Reduction System for Extensional Lambda-mu Calculus

Koji Nakazawa and Tomoharu Nagai

Graduate School of Informatics, Kyoto University, Kyoto, Japan

Abstract. The $\Lambda\mu$-calculus is an extension of Parigot's $\lambda\mu$-calculus. For the untyped $\Lambda\mu$-calculus, Saurin proved some fundamental properties such as the standardization and the separation theorem. Nakazawa and Katsumata gave extensional models, called stream models, in which terms are represented as functions on streams. This paper introduces a conservative extension of the $\Lambda\mu$-calculus, called $\Lambda\mu_{\mathsf{cons}}$, from which the open term model is straightforwardly constructed as a stream model, and for which we can define a reduction system satisfying several fundamental properties such as confluence, subject reduction, and strong normalization.

1 Introduction

The $\lambda\mu$-calculus was originally introduced by Parigot [16] as a term assignment system for the classical natural deduction, and then a lot of studies have been devoted to the $\lambda\mu$-calculus from both sides of logic and computer science. An extension of the $\lambda\mu$-calculus was given by de Groote [6] to study continuation-passing-style translations for the calculus. As Saurin showed in [18,20], an untyped variant of this extension, called the $\Lambda\mu$-calculus, enjoys fundamental properties such as the standardization and the separation theorem. In particular, the latter does not hold for the original $\lambda\mu$-calculus as shown by David and Py [5].

For the untyped $\Lambda\mu$-calculus, Nakazawa and Katsumata [14] gave a extensional model, called *stream model*. The stream model is a simple extension of the λ-model and similar to Streicher and Reus' continuation semantics for the $\lambda\mu$-calculus [22]. The stream model naturally reflects the idea that the $\Lambda\mu$-terms represent functions on streams. Nakazawa and Katsumata showed the soundness and gave an algebraic characterization for the stream model, but they have not discussed on completeness.

Regarding types, some type assignment systems for the $\Lambda\mu$-calculus has been introduced. Pagani and Saurin [15,21] gave a type system for the $\Lambda\mu$-calculus as a stream calculus, and Gaboardi and Saurin [10] proposed its extension with recursive types. De'Liguoro [8] gave an intersection type system and filter models for the $\Lambda\mu$-calculus, based on the stream model. However, the results on the stream model in [14] have not been adapted to typed calculi.

The main results of this paper are the following: (1) An extension $\Lambda\mu_{\mathsf{cons}}$ of the $\Lambda\mu$-calculus and its reduction system are proposed. The calculus $\Lambda\mu_{\mathsf{cons}}$ induces a term model as a stream model, and hence it is sound and complete with respect to the stream model. It is proved that the reduction on the untyped $\Lambda\mu_{\mathsf{cons}}$ is confluence. (2) A type assignment for $\Lambda\mu_{\mathsf{cons}}$ based on de'Liguoro's type system is proposed, and subject reduction and strong normalization of the reduction on the typed $\Lambda\mu_{\mathsf{cons}}$ are proved.

In Section 2 and 3, we define the equational theory and the reduction system for the untyped $\Lambda\mu_{\mathsf{cons}}$, and prove confluence. The calculus $\Lambda\mu_{\mathsf{cons}}$ explicitly contains stream expressions, and this extension is similar to the $\lambda\mu$-calculus of Streicher and Reus [22]. The reduction system proposed in this paper avoids the expansion rule in the $\Lambda\mu$-calculus, called (*fst*) in [18,21], and adopts a new rule (exp), which we can define with the new explicit stream expressions. The reduction system is confluent for whole of the untyped $\Lambda\mu_{\mathsf{cons}}$ including open terms in contrast to the $\Lambda\mu$-calculus in [21], where confluence holds for only the stream closed terms.

In Section 4 and 5, a typed variant of $\Lambda\mu_{\mathsf{cons}}$ is proposed, and subject reduction and strong normalization are proved. Following the structure of the stream model and de'Liguoro's type system [8], our type system restricts functional types to those from stream types to term types. For types of streams, we adopt (restricted forms of) recursive types for finiteness of the calculus, similarly to Gaboardi and Saurin's type system [10].

In Section 6, we discuss on the relationship with the existing calculi, such as the extended stack calculus [3], the untyped $\Lambda\mu$-calculus, and some type systems in [16,15,21,10]. In particular, we will see that the untyped $\Lambda\mu_{\mathsf{cons}}$ is conservative over the untyped $\Lambda\mu$-calculus (and hence over the λ-calculus), and inherits the separation theorem from $\Lambda\mu$.

2 $\Lambda\mu_{\mathsf{cons}}$

2.1 Definition of Untyped $\Lambda\mu_{\mathsf{cons}}$

As in [18], we adopt the notation $t\alpha$ to denote the named term $[\alpha]t$ in the original $\lambda\mu$-calculus, that can be read as a function application of t to a stream α. We use the constructors car and cdr to represent the head and the tail of a stream, respectively.

Definition 1 (Untyped $\Lambda\mu_{\mathsf{cons}}$). Suppose to have two sorts of variables: term variables, denoted by $x, y, \cdots$, and stream variables, $\alpha, \beta, \cdots$.

The *terms* and the *streams* of $\Lambda\mu_{\mathsf{cons}}$ are defined as

$$t, u ::= x \mid \lambda x.t \mid tu \mid \mu\alpha.t \mid tS \mid \mathsf{car}S \qquad S ::= \alpha \mid t :: S \mid \mathsf{cdr}S$$

The sets of terms and streams are denoted by Tm and St, respectively. Occurrences of x in $\lambda x.t$ and α in $\mu\alpha.t$ are considered to be bound. A variable occurrence which is not bound is called *free*. A term which contains no free stream variables is called *stream closed*. The size of t (and S) is defined as usual, and it is denoted by $|t|$ (and $|S|$, respectively).

The axiom schema of $\Lambda\mu_{\mathsf{cons}}$ are the following:

$$
\begin{array}{rcll}
(\lambda x.t)u &=& t[x := u] & (\beta_T)\\
(\mu\alpha.t)S &=& t[\alpha := S] & (\beta_S)\\
\lambda x.tx &=& t & (\eta_T)\\
\mu\alpha.t\alpha &=& t & (\eta_S)\\
(\mathsf{car}S) :: (\mathsf{cdr}S) &=& S & (\mathsf{surj})\\
\mathsf{car}(t :: S) &=& t & (\mathsf{car})\\
\mathsf{cdr}(t :: S) &=& S & (\mathsf{cdr})\\
t(u :: S) &=& (tu)S & (\mathsf{assoc})
\end{array}
$$

where, t contains no free x in (η_T), and t contains no free α in (η_S). The congruence relation $=_{\Lambda\mu_{\mathsf{cons}}}$ is defined from the above axiom schema.

We write $\mathsf{cdr}^i S$ to denote the i-time application of cdr to S for $i \geq 0$, and use the abbreviation $\mathsf{cadr}^i S \equiv \mathsf{car}(\mathsf{cdr}^i\, S)$.

Example 1. 1. The usual μ-rule $(\mu\alpha.t)u = \mu\alpha.t[[]\alpha := []u\alpha]$ is admissible as follows, where the special substitution $t[[]\alpha := []u\alpha]$ recursively replaces subterm occurrences of the form $v\alpha$ in t with $(vu)\alpha$.

$$
\begin{array}{rll}
(\mu\alpha.t)u &=_{\Lambda\mu_{\mathsf{cons}}} \mu\beta.(\mu\alpha.t)u\beta & (\eta_S)\\
&=_{\Lambda\mu_{\mathsf{cons}}} \mu\beta.t[\alpha := u :: \beta] & (\mathsf{assoc}, \beta_S)\\
&=_{\Lambda\mu_{\mathsf{cons}}} \mu\alpha.t[[]\alpha := []u\alpha] & (\mathsf{assoc}).
\end{array}
$$

2. By a fixed-point combinator Y in the λ-calculus, the n-th function on streams is defined as

$$\mathsf{nth} \equiv Y(\lambda f.\mu\alpha.\lambda n.\mathsf{ifzero}\ n\ \mathsf{then}\ (\mathsf{car}\,\alpha)\ \mathsf{else}\ f(\mathsf{cdr}\,\alpha)(\mathsf{pred}\,n)),$$

where ifzero and pred are defined on the Church numerals, and then we have $\mathsf{nth}\,S\,\overline{k} =_{\Lambda\mu_{\mathsf{cons}}} \mathsf{cadr}^k S$, where $\overline{k}$ is the Church numeral representing k.

The calculus $\Lambda\mu_{\mathsf{cons}}$ is a natural extension of the $\Lambda\mu$-calculus, and it is also close to the $\lambda\mu$-calculus in [22], which explicitly has the expressions for continuations but no cdr operator. The main difference from these existing calculi is the surjectivity axiom (surj). We will discuss the relationship with existing calculi in Section 6.

2.2 Stream Models for Untyped $\Lambda\mu_{\mathsf{cons}}$

The stream models are defined as in [14]. We use $\overline{\lambda}$ to denote the meta-level function abstraction.

Definition 2. The set $\mathcal{S}$ is called a *stream set* on a set $\mathcal{D}$ if there is a bijective mapping $(::)$ from $\mathcal{D} \times \mathcal{S}$ to $\mathcal{S}$. For a stream set $\mathcal{S}$, the inverse of $(::)$ is denoted by $\langle \mathsf{Car}, \mathsf{Cdr} \rangle$.

Definition 3. A *stream model* consists of
a non-empty set $\mathcal{D}$ and a stream set $\mathcal{S}$ on $\mathcal{D}$,
a subset $[\mathcal{S} \to \mathcal{D}]$ of the set of functions from $\mathcal{S}$ to $\mathcal{D}$,
$\Psi : [\mathcal{S} \to \mathcal{D}] \to \mathcal{D}$ a bijective mapping,
such that the meaning function $[\![\cdot]\!]_\rho$ can be defined for any function ρ from term variables to $\mathcal{D}$ and stream variables to $\mathcal{S}$ as follows, where $d\star s$ denotes $\Psi^{-1}(d)(s)$.

$$\begin{aligned}
[\![x]\!]_\rho &= \rho(x) & [\![\alpha]\!]_\rho &= \rho(\alpha)\\
[\![\lambda x.t]\!]_\rho &= \Psi(\overline{\lambda} s \in \mathcal{S}.[\![t]\!]_{\rho[x\mapsto \mathsf{Car}\, s]}\star(\mathsf{Cdr}\, s)) & [\![t :: S]\!]_\rho &= [\![t]\!]_\rho :: [\![S]\!]_\rho\\
[\![tu]\!]_\rho &= \Psi(\overline{\lambda} s \in \mathcal{S}.[\![t]\!]_\rho\star([\![u]\!]_\rho :: s)) & [\![\mathsf{cdr} S]\!]_\rho &= \mathsf{Cdr}[\![S]\!]_\rho\\
[\![\mu\alpha.t]\!]_\rho &= \Psi(\overline{\lambda} s \in \mathcal{S}.[\![t]\!]_{\rho[\alpha\mapsto s]}) & &\\
[\![tS]\!]_\rho &= [\![t]\!]_\rho\star[\![S]\!]_\rho & &\\
[\![\mathsf{car} S]\!]_\rho &= \mathsf{Car}[\![S]\!]_\rho & &
\end{aligned}$$

The set $\mathsf{Tm}/=_{\Lambda\mu_{\mathsf{cons}}}$ is a stream model, which we call *open term model.*

Proposition 1 (Open term model). *Let $[t]$ and $[S]$ be the equivalence classes of t and S with respect to $=_{\Lambda\mu_{\mathsf{cons}}}$, and define*

$$\mathcal{D} = \{[t] \mid t \in \mathsf{Tm}\} \qquad \mathcal{S} = \{[S] \mid S \in \mathsf{St}\} \qquad [\mathcal{S} \to \mathcal{D}] = \{f_{[t]} \mid t \in \mathsf{Tm}\},$$

where $f_{[t]}$ denotes the function $\overline{\lambda}[S] \in \mathcal{S}.[tS]$. Ψ is defined as $\Psi(f_{[t]}) = [t]$. Then, these give a stream model with the meaning function given by $[\![t]\!]_\rho = [t\theta_\rho]$, where θ_ρ is the substitution such that $\theta_\rho(x) = u$ for $\rho(x) = [u]$ and $\theta_\rho(\alpha) = S$ for $\rho(\alpha) = [S]$.

Proof. Straightforward. Note that $[t\theta_\rho]$ is independent of the choice of θ_ρ.

Then, the following is easy to show.

Theorem 1 (Soundness and completeness). *For any t and u, $t =_{\Lambda\mu_{\mathsf{cons}}} u$ holds if and only if $[\![t]\!]_\rho = [\![u]\!]_\rho$ holds for any stream model and ρ.*

Some properties of the stream models are shown in [14]. One of them guarantees existence of a non-trivial stream model, which gives a semantical proof of the consistency of the equational theory of $\Lambda\mu_{\mathsf{cons}}$.

Proposition 2 ([14]). *For any pointed CPO D, there exists a stream model D^S_∞ into which D can be embedded.*

Corollary 1 (Consistency). *There exist closed $\Lambda\mu_{\mathsf{cons}}$-terms t and u such that $t =_{\Lambda\mu_{\mathsf{cons}}} u$ does not hold.*

3 Reduction System

3.1 Reduction for Untyped $\Lambda\mu_{\mathsf{cons}}$

Definition 4. 1. The one-step reduction $\to$ on terms and streams of $\Lambda\mu_{\mathsf{cons}}$ is the least compatible relation satisfying the following axioms.

$$
\begin{array}{rll}
(\mu\alpha.t)u \to \mu\alpha.t[\alpha := u :: \alpha] & & (\beta_T) \\
(\mu\alpha.t)S \to t[\alpha := S] & & (\beta_S) \\
\lambda x.t \to \mu\alpha.t[x := \mathsf{car}\alpha](\mathsf{cdr}\alpha) & & (\mathsf{exp}) \\
t(u :: S) \to tuS & & (\mathsf{assoc}) \\
\mathsf{car}(u :: S) \to u & & (\mathsf{car}) \\
\mathsf{cdr}(u :: S) \to S & & (\mathsf{cdr}) \\
\mu\alpha.t\alpha \to t & (\alpha \notin FV(t)) & (\eta_S) \\
(\mathsf{car}S) :: (\mathsf{cdr}S) \to S & & (\eta_{::}) \\
t(\mathsf{car}S)(\mathsf{cdr}S) \to tS & & (\eta'_{::})
\end{array}
$$

Here, α in (exp) is a fresh stream variable. The relation $\to^*$ is the reflexive transitive closure of $\to$, the relation $\to^+$ is the transitive closure of $\to$, and the relation $\to^=$ is the reflexive closure of $\to$.

2. The relations $\to_{\mathsf{B}}$ and $\to_{\mathsf{E}}$ are defined as the least compatible relations satisfying the following axioms, respectively.
$\to_{\mathsf{B}}$: (β_T), (β_S), (exp), (assoc), (car), and (cdr)
$\to_{\mathsf{E}}$: (η_S), $(\eta_{::})$, $(\eta'_{::})$, (car), and (cdr)
We also use $\to_{\mathsf{B}}^*$, $\to_{\mathsf{E}}^+$, and so on.

Note that the $\to_{\mathsf{B}}$-normal forms are characterized by

$$t ::= a \mid \mu\alpha.t \qquad\qquad a ::= x \mid \mathsf{cadr}^n\alpha \mid at \mid a(\mathsf{cdr}^n\alpha).$$

We can easily see that the usual β- and η-rules in the λ-calculus are derivable, that is, $(\lambda x.t)u \to^* t[x := u]$ and $\lambda x.tx \to^* t$ for $x \notin FV(t)$ hold. Hence, the $\beta\eta$-reduction of the λ-calculus and the reduction of Parigot's $\lambda\mu$-calculus including the renaming and the η-rules for μ-abstractions can be simulated in $\Lambda\mu_{\mathsf{cons}}$. Furthermore, the following holds.

Proposition 3. *The equivalence closure of $\to$ coincides with $=_{\Lambda\mu_{\mathsf{cons}}}$.*

It is known that naïvely adding the η-rule $\lambda x.tx \to_\eta t$ to the $\lambda\mu$-calculus destroys confluence [5]. The counterexample is $t = \lambda x.(\mu\alpha.y\beta)x$, and then

$$t \to_\eta \mu\alpha.y\beta, \qquad\qquad t \to_\beta \lambda x.\mu\alpha.y\beta.$$

In order to recover confluence, the rule called (*fst*) in [21] has been proposed as

$$\mu\alpha.t \to \lambda x.\mu\alpha.t[[]\alpha := []x\alpha] \qquad\qquad (\mathit{fst}).$$

It seems natural since it means the surjectivity of the bound variable α. However, if we consider type systems, it induces a type dependent reduction for subject reduction and strong normalization.

Alternatively, by the explicit stream syntax in $\Lambda\mu_{\mathsf{cons}}$, we can define the new rule (exp), and the above critical pair is solved as

$$\lambda x.\mu\alpha.y\beta \to_{\mathsf{exp}} \mu\gamma.(\mu\alpha.y\beta)(\mathsf{cdr}\gamma) \to_{\beta_S} \mu\gamma.y\beta (= \mu\alpha.y\beta).$$

The reduction system with (exp) will be adapted to the typed $\Lambda\mu_{\mathsf{cons}}$ without any restriction of types, and subject reduction and strong normalization will be proved.

3.2 Confluence

We prove confluence of $\to$ by (1) confluence of $\to_{\mathsf{B}}$, (2) confluence of $\to_{\mathsf{E}}$, and (3) commutativity of them. In contrast to the $\Lambda\mu$-calculus [21], the result is not restricted to stream closed terms, and hence the Church-Rosser theorem directly follows from the confluence.

Proposition 4. *$\to_{\mathsf{B}}$ and $\to_{\mathsf{E}}$ are respectively confluent.*

Proof. (B) By a generalized notion of complete development, which is independently introduced in [7,11]. We define the mapping $(\cdot)^\dagger$ as follows.

$$\begin{aligned}
x^\dagger &= x & \alpha^\dagger &= \alpha \\
(\lambda x.t)^\dagger &= \mu\alpha.t^\dagger[x := \mathsf{car}\alpha](\mathsf{cdr}\alpha) & (\mathsf{cdr}(t :: S))^\dagger &= S^\dagger \\
(\mu\alpha.t)^\dagger &= \mu\alpha.t^\dagger & (\mathsf{cdr}S)^\dagger &= \mathsf{cdr}S^\dagger \quad \text{(otherwise)} \\
((\mu\alpha.t)u)^\dagger &= \mu\alpha.t^\dagger[\alpha := u^\dagger :: \alpha] & (t :: S)^\dagger &= t^\dagger :: S^\dagger \\
(tu)^\dagger &= t^\dagger u^\dagger \quad \text{(otherwise)} \\
((\mu\alpha.t)S)^\dagger &= t^\dagger[\alpha := S^\dagger] \\
((\mu\alpha.t)uS)^\dagger &= t^\dagger[\alpha := u^\dagger :: S^\dagger] \\
(t(u :: S))^\dagger &= t^\dagger u^\dagger S^\dagger \quad (t \neq \mu\text{-abst.}) \\
(tS)^\dagger &= t^\dagger S^\dagger \quad \text{(otherwise)} \\
(\mathsf{car}(t :: S))^\dagger &= t^\dagger \\
(\mathsf{car}S)^\dagger &= \mathsf{car}S^\dagger \quad \text{(otherwise)}
\end{aligned}$$

Then, we can prove that $t \to_{\mathsf{B}} u$ implies $u \to_{\mathsf{B}}^* t^\dagger \to_{\mathsf{B}}^* u^\dagger$, from which the confluence follows. The only non-trivial point is that we exceptionally define $((\mu\alpha.t)uS)^\dagger = t^\dagger[\alpha := u^\dagger :: S^\dagger]$ (not $(\mu\alpha.t^\dagger[\alpha := u^\dagger :: \alpha])S^\dagger$), since we have to show that $((\mu\alpha.t)(u :: S))^\dagger \to_{\mathsf{B}}^* ((\mu\alpha.t)uS)^\dagger$, the left-hand side of which is $t^\dagger[\alpha := (u :: S)^\dagger] = t^\dagger[\alpha := u^\dagger :: S^\dagger]$.

(E) Since $\to_{\mathsf{E}}$ is clearly strongly normalizing, it is sufficient to prove local confluence. It is straightforward.

In order to prove commutativity of B and E, we consider the following restricted E-reduction.

Definition 5. The relation $\to_{\mathsf{E}^-}$ is the least compatible relation satisfying (η_S), (car), (cdr), and the restricted forms of $(\eta_{::})$ and $(\eta'_{::})$ as follows.

$$(\mathsf{car}\,\mathsf{cdr}^n\alpha) :: (\mathsf{cdr}\,\mathsf{cdr}^n\alpha) \to \mathsf{cdr}^n\alpha \qquad (\eta_{::}^-)$$
$$t(\mathsf{car}\,\mathsf{cdr}^n\alpha)(\mathsf{cdr}\,\mathsf{cdr}^n\alpha) \to t(\mathsf{cdr}^n\alpha) \qquad (\eta_{::}'^-)$$

The relation $\to_{\mathsf{cdr}}$ is the least compatible relation satisfying (cdr).

The relation $\to_{\mathsf{E}^-}$ is introduced to show a variant of strong commutativity, that is Lemma 2.2. Note that $t \to_{\mathsf{E}^-} t'$ does not necessarily imply $t[\alpha := S] \to_{\mathsf{E}^-} t'[\alpha := S]$ due to the restriction. Instead, we have the following lemma.

Lemma 1. *1. Any S is reduced by $\to_{\mathsf{cdr}}$ to a term of the form either $t' :: S'$ or $\mathsf{cdr}^n\alpha$ for some $n \geq 0$.*

2. For any S, there exists S' such that $S \to_{\mathsf{cdr}}^* S'$ and $(\mathsf{car}S) :: (\mathsf{cdr}S) \to_{\mathsf{E}^-}^* S'$.

3. If $t \to_{\mathsf{E}^-} t'$, then $t[\alpha := S] \to_{\mathsf{E}^-} u$ and $t'[\alpha := S] \to_{\mathsf{cdr}}^* u$ for some u.

Proof. 1. By induction on S.

2. By 1, there exists S' such that $S \to_{\mathsf{cdr}}^* S'$ and S' is of the form $t'_0 :: S'_0$ or $\mathsf{cdr}^n\alpha$. In the former case, we have $(\mathsf{car}S) :: (\mathsf{cdr}S) \to_{\mathsf{cdr}}^* (\mathsf{car}(t'_0 :: S'_0)) :: (\mathsf{cdr}(t'_0 :: S'_0)) \to_{\mathsf{car},\mathsf{cdr}}^* t'_0 :: S'_0$. In the latter case, we have $(\mathsf{car}S) :: (\mathsf{cdr}S) \to_{\mathsf{cdr}}^* (\mathsf{car}\,\mathsf{cdr}^n\alpha) :: (\mathsf{cdr}\,\mathsf{cdr}^n\alpha) \to_{\mathsf{E}^-} \mathsf{cdr}^n\alpha$.

3. By induction on $t \to_{\mathsf{E}^-} t'$. Consider the case of $\mathsf{car}\,\mathsf{cdr}^n\alpha :: \mathsf{cdr}\,\mathsf{cdr}^n\alpha \to_{\mathsf{E}^-} \mathsf{cdr}^n\alpha$ by $(\eta_{::}^-)$. By 2, there exists S' such that $(\mathsf{car}\,\mathsf{cdr}^n S) :: (\mathsf{cdr}\,\mathsf{cdr}^n S) \to_{\mathsf{E}^-}^* S'$ and $(\mathsf{cdr}^n S) \to_{\mathsf{cdr}}^* S'$. The case of $\eta_{::}'^-$ is similarly proved, and the other cases are straightforward.

Lemma 2. *The following commuting diagrams hold.*

1. $$\begin{tikzcd} \cdot \arrow[r, "\mathsf{cdr}"] \arrow[d, "\mathsf{E}^-"'] & \cdot \arrow[d, dotted, "="', "\mathsf{E}^-"] \\ \cdot \arrow[r, dotted, "="', "\mathsf{cdr}"'] & \exists\,\cdot \end{tikzcd}$$

2. $$\begin{tikzcd} \cdot \arrow[rr, "\mathsf{B}"] \arrow[d, "\mathsf{E}^-"'] & & \cdot \arrow[d, dotted, "*"', "\mathsf{E}^-"] \\ \cdot \arrow[r, dotted, "="', "\mathsf{B}"'] & \cdot \arrow[r, dotted, "*"', "\mathsf{cdr}"'] & \exists\,\cdot \end{tikzcd}$$

3. $$\begin{tikzcd} \cdot \arrow[r, "\mathsf{cdr}", "*"'] \arrow[d, "\mathsf{E}"', "*"] & \cdot \arrow[d, dotted, "*"', "\mathsf{E}^-"] \\ \cdot \arrow[r, dotted, "*", "\mathsf{cdr}"'] & \exists\,\cdot \end{tikzcd}$$

Proof. 1. By induction on the size of terms and streams.

2. By induction on the terms and streams. We only show the case of $(\mu\alpha.t)S \to_{\mathsf{B}} t[\alpha := S]$ and $(\mu\alpha.t)S \to_{\mathsf{E}^-} (\mu\alpha.t')S$ where $t \to_{\mathsf{E}^-} t'$. By Lemma 1, there exists u such that $t[\alpha := S] \to_{\mathsf{E}^-}^* u$ and $(\mu\alpha.t')S \to_{\mathsf{B}} t'[\alpha := S] \to_{\mathsf{cdr}}^* u$.

3. By induction on the length of $\to_{\mathsf{E}}^*$. By 1, $\to_{\mathsf{E}^-}^*$ and $\to_{\mathsf{cdr}}^*$ commute, so it is sufficient to consider each step of $(\eta_{::})$ and $(\eta'_{::})$ which is not restricted. It is proved since $t \to_{\mathsf{E}} t'$ implies that there exists u such that $t \to_{\mathsf{E}^-}^* u$ and $t' \to_{\mathsf{cdr}}^* u$, that is proved by Lemma 1.2.

If we consider the full E-reduction, $\to^=$ in 1 and 2 does not necessarily hold.

Proposition 5. *$\to^*_{\mathsf{B}}$ and $\to^*_{\mathsf{E}}$ commute.*

Proof. By Lemma 2.3, the left triangle in the following diagram commute. By 1 and 2 of Lemma 2, $\to^*_{\mathsf{B}}$ and $\to^*_{\mathsf{E}^-}$ commute, and hence $\to^*_{\mathsf{B}}$ and $\to^*_{\mathsf{E}}$ commute.

$$\begin{tikzcd}
\cdot \arrow[rr, "\mathsf{B}", "*"'] \arrow[d, "\mathsf{E}"', "*"] \arrow[dr, dotted, "\mathsf{E}^-", "*"'] & & \cdot \arrow[d, dotted, "\mathsf{E}^-", "*"'] \\
\cdot \arrow[r, dotted, "*", "\mathsf{cdr}"'] & \cdot \arrow[r, dotted, "*", "\mathsf{B}"'] & \cdot
\end{tikzcd}$$

Theorem 2 (Confluence). *The reduction $\to$ is confluent.*

Proof. It follows from Proposition 4 and 5.

Corollary 2 (Church-Rosser theorem). *If $t =_{\Lambda\mu_{\mathsf{cons}}} t'$ holds, then there exists u such that $t \to^* u$ and $t' \to^* u$.*

The Church-Rosser theorem gives a syntactic proof of consistency of the equational logic of $\Lambda\mu_{\mathsf{cons}}$, since, for example, $\mu\alpha.\mathsf{car}\alpha$ and $\mu\alpha.\mathsf{cadr}\alpha$ are different normal forms.

Corollary 3 (Consistency of $\Lambda\mu_{\mathsf{cons}}$). *There exists two closed $\Lambda\mu_{\mathsf{cons}}$-terms t and u such that $t =_{\Lambda\mu_{\mathsf{cons}}} u$ does not hold.*

4 Typed $\Lambda\mu_{\mathsf{cons}}$

We will give a type assignment system for $\Lambda\mu_{\mathsf{cons}}$, inspired by de'Liguoro's type system for the $\Lambda\mu$-calculus [8], and adopting recursive types to represent types for streams like [10].

4.1 Definition of Typed $\Lambda\mu_{\mathsf{cons}}$

The types of streams will be introduced as non-empty lists of types of individual data such as $[\delta_0, \delta_1]$, which is a special case of the recursive types, and which is just an abbreviation for $\mu\chi.\delta_0 \times \delta_1 \times \chi$. The following axiomatization for the equivalence on the types are borrowed from the well-known results for recursively defined trees in [17,12,1,2].

Definition 6 (Typed $\Lambda\mu_{\mathsf{cons}}$). The types consist of two sorts, *term types* and *stream types*, which are inductively defined as

$$\delta ::= X \mid \sigma \to \delta \qquad\qquad \sigma ::= [\delta_0, \cdots, \delta_{n-1}] \mid \delta \times \sigma,$$

where X ranges over the base types, and $[\delta_0, \cdots, \delta_{n-1}]$ is a non-empty finite list of types. The relation $\sim$ on the types is defined as the least congruence relation satisfying the following.

$$\frac{}{[\delta_0, \cdots, \delta_{n-1}] \sim \delta_0 \times [\delta_1, \cdots \delta_{n-1}, \delta_0]}\ (\mathrm{Fld}) \qquad \frac{\delta_0 \times \cdots \delta_{n-1} \times \sigma \sim \sigma}{[\delta_0, \cdots, \delta_{n-1}] \sim \sigma}\ (\mathrm{Ctr})$$

Note that $\sim$ is also defined on the term types as $\sigma \to \delta \sim \sigma' \to \delta'$ if $\sigma \sim \sigma'$ and $\delta \sim \delta'$.

A term context Γ and a stream context Δ are finite lists of pairs of the form $(x : \delta)$ and $(\alpha : \sigma)$, respectively, in which each variable occurs at most once.

The typing rules of $\Lambda\mu_{\mathsf{cons}}$ are the following.

$$\frac{}{\Gamma, x : \delta \mid \Delta \vdash x : \delta} \qquad \frac{}{\Gamma \mid \Delta, \alpha : \sigma \vdash \alpha : \sigma} \qquad \frac{\Gamma \mid \Delta \vdash S : \delta \times \sigma}{\Gamma \mid \Delta \vdash \mathsf{car}\, S : \delta}$$

$$\frac{\Gamma, x : \delta \mid \Delta \vdash t : \sigma \to \delta'}{\Gamma \mid \Delta \vdash \lambda x.t : \delta \times \sigma \to \delta'} \qquad \frac{\Gamma \mid \Delta \vdash t : \delta \times \sigma \to \delta' \quad \Gamma \mid \Delta \vdash u : \delta}{\Gamma \mid \Delta \vdash tu : \sigma \to \delta'}$$

$$\frac{\Gamma \mid \Delta, \alpha : \sigma \vdash t : \delta}{\Gamma \mid \Delta \vdash \mu\alpha.t : \sigma \to \delta} \qquad \frac{\Gamma \mid \Delta \vdash t : \sigma \to \delta \quad \Gamma \mid \Delta \vdash S : \sigma}{\Gamma \mid \Delta \vdash tS : \delta}$$

$$\frac{\Gamma \mid \Delta \vdash t : \delta \quad \Gamma \mid \Delta \vdash S : \sigma}{\Gamma \mid \Delta \vdash t :: S : \delta \times \sigma} \qquad \frac{\Gamma \mid \Delta \vdash S : \delta \times \sigma}{\Gamma \mid \Delta \vdash \mathsf{cdr}, S : \sigma}$$

$$\frac{\Gamma \mid \Delta \vdash t : \delta \quad \delta \sim \delta'}{\Gamma \mid \Delta \vdash t : \delta'} \qquad \frac{\Gamma \mid \Delta \vdash S : \sigma \quad \sigma \sim \sigma'}{\Gamma \mid \Delta \vdash S : \sigma'}$$

The relation $\Gamma \mid \Delta \vdash t_1 = t_2 : \delta$ means $\Gamma \mid \Delta \vdash t_i : \delta$ $(i = 1, 2)$ and $t_1 =_{\Lambda\mu_{\mathsf{cons}}} t_2$.

We consider the restricted recursive types only for finiteness of the type system, and the choice of the equivalence $\sim$ is not essential for the following discussion. We can adopt the equivalence defined by only the fold/unfold axiom as in [10]. Indeed, the discussion in the following sections can be done in more general setting in which types of streams are represented as infinite product types, called *expanded types.* Some notions such as the stream models and the reducibility predicate for the strong normalization proof will be defined on the expanded types.

Definition 7. 1. The *expanded types* are defined by

$$\delta ::= X \mid \sigma \to \delta \qquad\qquad \sigma ::= \Pi_{i \in \mathbb{N}} \delta_i.$$

We also use the notation $\delta \times \Pi_{i \in \mathbb{N}} \delta'_i$, which is straightforwardly defined.

2. Given a stream type σ and $i \in \mathbb{N}$, we define $(\sigma)_i$ by

$$([\delta_0, \cdots, \delta_{n-1}])_i = \delta_{i \bmod n} \qquad\qquad (\delta \times \sigma)_i = \begin{cases} \delta & (i = 0) \\ (\sigma)_{i-1} & (i > 0), \end{cases}$$

where $i \bmod n$ denotes the remainder of the division of i by n. We also define the function $(\sigma)_i$ for the expanded types as $(\Pi_{j \in \mathbb{N}} \delta_j)_i = \delta_i$.

3. The *expansion* of the types is defined as follows.

$$\langle\!| X |\!\rangle = X \qquad \langle\!| \sigma \to \delta |\!\rangle = \langle\!| \sigma |\!\rangle \to \langle\!| \delta |\!\rangle \qquad \langle\!| \sigma |\!\rangle = \Pi_{i \in \mathbb{N}} \langle\!| (\sigma)_i |\!\rangle$$

Note that the relation $\sqsubset$ on expanded types defined as $\sigma, \delta \sqsubset \sigma \to \delta$ and $\delta_i \sqsubset \Pi_{i \in \mathbb{N}} \delta_i$ is a well-founded order, and we use the induction on this order.

Proposition 6. $\delta \sim \delta'$ *iff* $\langle\!|\delta|\!\rangle = \langle\!|\delta'|\!\rangle$.

It follows from the completeness of the axiomatization, for example, in [1].

Example 2. In [14], SCL is proposed as a combinatory calculus which is equivalent to the $\Lambda\mu$-calculus. However, some combinators of SCL are not typable in the original typed $\lambda\mu$-calculus. On the other hand, the SCL combinators are typable in $\Lambda\mu_{\mathsf{cons}}$ such as

$$\begin{array}{ll}(\mathsf{K}_1) & \cdot \mid \cdot \vdash \lambda x.\mu\alpha.x : \delta \times \sigma \to \delta \\ (\mathsf{W}_1) & \cdot \mid \cdot \vdash \lambda x.\mu\alpha.x\alpha\alpha : (\sigma \to \sigma \to \delta) \times \sigma \to \delta\end{array}$$

for any term type δ and any stream type σ.

We will discuss the related typed calculi in Section 6 and in [13].

4.2 Stream Models for Typed $\Lambda\mu_{\mathsf{cons}}$

In [13], it is shown that the stream models are adapted to the typed $\Lambda\mu_{\mathsf{cons}}$, and we briefly introduce the results.

A stream model for the typed $\Lambda\mu_{\mathsf{cons}}$ consists of

- family of sets $\mathcal{A}^\delta$ and $\mathcal{A}^\sigma$ indexed by the expanded types
- an operation $(\star) : \mathcal{A}^{\sigma\to\delta} \times \mathcal{A}^\sigma \to \mathcal{A}^\delta$ for each σ and δ such that

$$\forall f, g \in \mathcal{A}^{\sigma\to\delta}.[\forall s \in \mathcal{A}^\sigma.[f\star s = g\star s] \Rightarrow f = g].$$

- a bijection $(::) : \mathcal{A}^\delta \times \mathcal{A}^\sigma \to \mathcal{A}^{\delta\times\sigma}$ for each δ and σ, the inverse of which consists of the projection functions $\langle \mathsf{Car}, \mathsf{Cdr}\rangle$.
- a meaning function $[\![\cdot]\!]$ such that $[\![\lambda x^{\delta'}.t^{\sigma\to\delta}]\!]_\rho \in \mathcal{A}^{\delta'\times\sigma\to\delta}$ and $[\![\lambda x^{\delta'}.t^{\sigma\to\delta}]\!]_\rho\star s = [\![t]\!]_{\rho[x\mapsto\mathsf{Car}(s)]}\star\mathsf{Cdr}(s)$ for any $s \in \mathcal{A}^{\delta'\times\sigma}$, and so on.

In particular, a stream model is called *full* if $\mathcal{A}^{\sigma\to\delta}$ is the whole function space from $\mathcal{A}^\sigma$ to $\mathcal{A}^\delta$ for any σ and δ, and $\mathcal{A}^\sigma$ is $\Pi_{i\in\mathbb{N}}\mathcal{A}^{(\sigma)_i}$ for any σ.

The typed $\Lambda\mu_{\mathsf{cons}}$ is sound and complete with respect to the stream model. Furthermore, we can show the following property, corresponding to Friedman's theorem [9]: the extensional equality in $\lambda^{\to}$ is characterized by an arbitrary individual full type hierarchy with infinite domains for base types. This theorem is proved by giving the logical relation on the stream models between the open term model and the full stream model.

Theorem 3 (Friedman's theorem for $\Lambda\mu_{\mathsf{cons}}$, [13]). *Suppose that a stream model $\mathcal{F}$ is full and all of $\mathcal{F}^X$ are infinite. Then, for any closed typable t and u, $t =_{\Lambda\mu_{\mathsf{cons}}} u$ holds if and only if $[\![t]\!]^{\mathcal{F}} = [\![u]\!]^{\mathcal{F}}$ holds.*

5 Reduction System for Typed $\Lambda\mu_{\mathsf{cons}}$

In this section, we show two fundamental properties of the reduction on the typed $\Lambda\mu_{\mathsf{cons}}$: subject reduction and strong normalization.

5.1 Subject Reduction

We omit the proof of the subject reduction since it is straightforwardly proved using the usual generation lemma modulo $\sim$.

Theorem 4 (Subject reduction). *If $\Gamma \mid \Delta \vdash t : \delta$ and $t \to u$ hold, then we have $\Gamma \mid \Delta \vdash u : \delta$.*

5.2 Strong Normalization

First, we prove the strong normalization of $\to_{\mathsf{B}}$ by the usual reducibility, and then extend it to the full reduction $\to$. The set of terms and streams which are strongly normalizable with respect to $\to_{\mathsf{B}}$ are denoted by $\mathsf{SN_T}$ and $\mathsf{SN_S}$, respectively. Moreover, the applicative contexts are defined as $C ::= [] \mid Ct \mid CS$, and $\mathsf{SN_C}$ is the set of the applicative contexts in which $t \in \mathsf{SN_T}$ and $S \in \mathsf{SN_S}$.

Definition 8. The predicates Red indexed by the expanded types are defined as

$\mathsf{Red}_X = \mathsf{SN_T}$,

$t \in \mathsf{Red}_{\sigma \to \delta}$ iff, for any $S \in \mathsf{Red}_\sigma$, $tS \in \mathsf{Red}_\delta$,

$S \in \mathsf{Red}_\sigma$ iff, for any $n \geq 0$, $\mathsf{cadr}^n S \in \mathsf{Red}_{(\sigma)_n}$.

For (not expanded) types, Red_δ and Red_σ mean $\mathsf{Red}_{\langle\!|\delta|\!\rangle}$ and $\mathsf{Red}_{\langle\!|\sigma|\!\rangle}$, respectively.

Note that, $S \in \mathsf{Red}_{\delta\times\sigma}$ iff $\mathsf{car}\, S \in \mathsf{Red}_\delta$ and $\mathsf{cdr}\, S \in \mathsf{Red}_\sigma$ by the definition.

Lemma 3. *1. $\mathsf{Red}_\delta \subseteq \mathsf{SN_T}$ and $\mathsf{Red}_\sigma \subseteq \mathsf{SN_S}$ hold.*

2. For any $C \in \mathsf{SN_C}$, $C[x] \in \mathsf{Red}_\delta$ and $C[\mathsf{cadr}^n \alpha] \in \mathsf{Red}_\delta$ hold.

3. $\alpha \in \mathsf{Red}_\sigma$ holds.

Proof. They are simultaneously proved by induction on the expanded types.

Lemma 4. *For any expanded types δ, σ, and any applicative context C, the following hold.*

1. *For any $u \in \mathsf{SN_T}$, $C[\mu\alpha.t[\alpha := u :: \alpha]] \in \mathsf{Red}_\delta$ implies $C[(\mu\alpha.t)u] \in \mathsf{Red}_\delta$.*
2. *For any $S \in \mathsf{SN_S}$, $C[t[\alpha := S]] \in \mathsf{Red}_\delta$ implies $C[(\mu\alpha.t)S] \in \mathsf{Red}_\delta$.*
3. *$C[\mu\alpha.t[x := \mathsf{car}\alpha](\mathsf{cdr}\alpha)] \in \mathsf{Red}_\delta$ implies $C[\lambda x.t] \in \mathsf{Red}_\delta$.*
4. *For any $S \in \mathsf{SN_S}$, if $C[t] \in \mathsf{Red}_\delta$ implies $C[\mathsf{car}(t :: S)] \in \mathsf{Red}_\delta$.*
5. *For any $t \in \mathsf{SN_T}$, if $C[\mathsf{cadr}^n S] \in \mathsf{Red}_\delta$ implies $C[\mathsf{cadr}^{n+1}(t :: S)] \in \mathsf{Red}_\delta$.*

Proof. We give only the proof of 2, and the others are proved similarly. In this proof, $\#t$ for $t \in \mathsf{SN_T}$ denotes the maximum length of reduction sequences from t, and $\#S$ for $S \in \mathsf{SN_S}$ is similarly defined.

First, we show that, for any $S \in \mathsf{SN_S}$, $C[t[\alpha := S]] \in \mathsf{SN_T}$ implies $C[(\mu\alpha.t)S] \in \mathsf{SN_T}$, by induction on the triple $\langle \#S, |S|, \#C[t[\alpha := S]]\rangle$ with the lexicographical order. It is sufficient to show that $u \in \mathsf{SN_T}$ for any u such that $C[(\mu\alpha.t)S] \to_{\mathsf{B}} u$.

Case $C[(\mu\alpha.t)S] \to_{\mathsf{B}} C[t[\alpha := S]]$. $C[t[\alpha := S]] \in \mathsf{SN_T}$ is the assumption.

Case $C[(\mu\alpha.t)S] \to_{\mathsf{B}} C'[(\mu\alpha.t')S]$. We have $C[t[\alpha := S]] \to_{\mathsf{B}} C'[t'[\alpha := S]]$, and hence $C'[(\mu\alpha.t')S] \in \mathsf{SN_T}$ follows from the induction hypothesis since $\#C[t[\alpha := S]] > \#C'[t'[\alpha := S]]$.

Case $C[(\mu\alpha.t)S] \to_{\mathsf{B}} C[(\mu\alpha.t)S']$. It follows from the induction hypothesis since $\#S > \#S'$.

Case $C[(\mu\alpha.t)(t_0 :: S_0)] \to_{\mathsf{B}} C[(\mu\alpha.t)t_0S_0]$ by (assoc). Since $\#(t_0 :: S_0) \geq \#S_0$ and $|t_0 :: S_0| > |S_0|$, we have $C[(\mu\alpha.t[\alpha := t_0 :: \alpha])S_0] \in \mathsf{SN}_{\mathsf{T}}$ by the induction hypothesis. Since $t_0 \in \mathsf{SN}_{\mathsf{T}}$, we have $C[(\mu\alpha.t)t_0S_0] \in \mathsf{SN}_{\mathsf{T}}$ by 1.

Secondly, the lemma is proved by induction on the expanded types. The base case is shown above, and the induction steps are straightforward.

Definition 9. $\mathsf{Red}_{\Gamma|\Delta}$ denotes the set of substitutions θ such that $\theta(x) \in \mathsf{Red}_\delta$ for any $x : \delta \in \Gamma$ and $\theta(\alpha) \in \mathsf{Red}_\sigma$ for any $\alpha : \sigma \in \Delta$.

Lemma 5. *If $\Gamma \mid \Delta \vdash t : \delta$ and $\theta \in \mathsf{Red}_{\Gamma|\Delta}$, then we have $t\theta \in \mathsf{Red}_\delta$.*

Proof. By induction on the derivation of $\Gamma \mid \Delta \vdash t : \delta$, using Lemma 4. Note that $\delta \sim \delta'$ implies $\mathsf{Red}_\delta = \mathsf{Red}_{\delta'}$.

Proposition 7. *Every typable term is in SN_{T}.*

Proof. By 2 and 3 of Lemma 3, the identity substitution θ is in $\mathsf{Red}_{\Gamma|\Delta}$ for any Γ and Δ. Hence, by Lemma 3.1 and Lemma 5, we have $t = t\theta \in \mathsf{Red}_\delta \subseteq \mathsf{SN}_{\mathsf{T}}$.

Theorem 5 (Strong normalization). *Every typable term is strongly normalizing with respect to $\to$.*

Proof. For any reduction sequence, we can postpone any E-reduction, that is, we can prove that $t \to_{\mathsf{E}} \cdot \to_{\mathsf{B}} u$ implies $t \to_{\mathsf{B}}^{+} \cdot \to_{\mathsf{E}}^{*} u$. Since $\to_{\mathsf{E}}$ is strongly normalizable, if we have an infinite sequence of $\to$, we can construct an infinite sequence of $\to_{\mathsf{B}}$, that contradicts Proposition 7.

6 Related Work

In this section, we discuss the relationship between $\Lambda\mu_{\mathsf{cons}}$ and the existing related systems such as the stack calculus in [4,3], the untyped $\Lambda\mu$-calculus in [18], Parigot's original typed $\lambda\mu$-calculus [16], Pagani and Saurin's $\Lambda_{\mathcal{S}}$ in [15], and Gaboardi and Saurin's $\Lambda_{\mathcal{S}}$ in [10].

6.1 Extended Stack Calculus

The calculus $\Lambda\mu_{\mathsf{cons}}$ can be seen as an extension of the nil-free fragment of the extended stack calculus in [3]. The stack calculus contains neither term variables, λ-abstractions, nor term applications, but they can be simulated. It is straightforward to see that the reduction of $\Lambda\mu_{\mathsf{cons}}$ is conservative over the stack calculus, that is, for terms t and u of the extended stack calculus without nil, $t \to^* u$ in the stack calculus if and only if $t \to^* u$ in $\Lambda\mu_{\mathsf{cons}}$. Moreover, our type system can be adapted to the extended stack calculus without nil in a straightforward way. The discussion in this paper on the stream models for the untyped and typed variants of $\Lambda\mu_{\mathsf{cons}}$ can be adapted to the extended stack calculus.

6.2 Untyped $\Lambda\mu$-Calculus

The calculus $\Lambda\mu_{\mathsf{cons}}$ is conservative over the $\Lambda\mu$-calculus.

Proposition 8 (Conservativity over $\Lambda\mu$). *For any $\Lambda\mu$-terms t and u, $t = u$ holds in $\Lambda\mu_{\mathsf{cons}}$ if and only if $t = u$ holds in $\Lambda\mu$.*

Corollary 4. *For any $\Lambda\mu$-terms t and u, $t = u$ holds in the $\Lambda\mu$-calculus if and only if $[\![t]\!]_\rho = [\![u]\!]_\rho$ for any stream model $\mathcal{A}$ and ρ.*

Saurin [18] proved the separation theorem of the $\Lambda\mu$-calculus. By the conservativity, $\Lambda\mu_{\mathsf{cons}}$ inherits the separation theorem from $\Lambda\mu$ for stream closed terms. The *canonical normal forms* in the $\Lambda\mu$-calculus are defined as terms which are η-normal and contain no subterm of the form either $(\lambda x.t)u$, $(\lambda x.t)\beta$, $(\mu\alpha.t)u$, or $(\mu\alpha.t)\beta$. The *stream applicative contexts* are defined as $\mathcal{C} ::= [] \mid \mathcal{C}t \mid \mathcal{C}\alpha$.

Theorem 6 (Separation theorem for $\Lambda\mu$, [18]). *Let $\Lambda\mu$-terms t_1 and t_2 be closed canonical normal forms. If $t_1 \neq t_2$ in $\Lambda\mu$, then there exists a stream applicative context $\mathcal{C}$ such that $\mathcal{C}[t_1] \to^* \lambda xy.x$ and $\mathcal{C}[t_2] \to^* \lambda xy.y$ hold in $\Lambda\mu$.*

By this theorem, the separation theorem for $\Lambda\mu_{\mathsf{cons}}$ is proved.

Theorem 7 (Separation theorem for $\Lambda\mu_{\mathsf{cons}}$). *Let $\Lambda\mu_{\mathsf{cons}}$-terms t_1 and t_2 be distinct closed normal forms. For any normal u_1 and u_2, there exists a stream applicative context $\mathcal{C}$ such that $\mathcal{C}[t_1] \to^* u_1$ and $\mathcal{C}[t_2] \to^* u_2$ hold in $\Lambda\mu_{\mathsf{cons}}$.*

In [19], Saurin also gave an interpretation of the $\Lambda\mu$-calculus (and its generalization, called stream hierarchy) with a CPS translation into the λ-calculus with surjective pairs, called λSP, and proved the completeness of the CPS translation. The term model induced from λSP is a special case of the stream models with $D = S$, so his result can be seen as the completeness of $\Lambda\mu$ with respect to the stream model.

6.3 Type Assignment for $\Lambda\mu$-Calculus

On the related type systems, more detailed discussion is found in [13].

In the typed $\Lambda\mu_{\mathsf{cons}}$, only functional types from streams to individual data are considered, inspired by the type system of de'Liguoro [8]. However, every typable term in Pagani and Saurin's $\Lambda_{\mathcal{S}}$ [15,21] is also typable in $\Lambda\mu_{\mathsf{cons}}$. Therefore, every typable term in Parigot's propositional typed $\lambda\mu$-calculus [16] is also typable in $\Lambda\mu_{\mathsf{cons}}$.

Here, we show the translation from the $\lambda\mu$-calculus to $\Lambda\mu_{\mathsf{cons}}$, which is based on the same idea of translations in Saurin [21] and van Bakel et al. [24]. They show that their type systems correspond to the image of negative translations from the classical logic to the intuitionistic logic. The following translation corresponds to the continuation-passing-style translation of Thielecke [23].

Definition 10 (Negative translation). We fix a type variable O and an arbitrary stream type θ, and we write $\neg\sigma$ for $\sigma \to O$. The *negative translation* $\overline{(\cdot)}$ from the implicational formulas to term types in $\Lambda\mu_{\mathsf{cons}}$ is defined as

$$\overline{A} = \neg A^{\bullet} \qquad\qquad p^{\bullet} = \theta \qquad\qquad (A \to B)^{\bullet} = \neg A^{\bullet} \times B^{\bullet}.$$

Proposition 9. *If $\Gamma \vdash t : A; \Delta$ holds in the propositional typed $\lambda\mu$-calculus, then $\neg\Gamma^{\bullet}; \Delta^{\bullet} \vdash t : \neg A^{\bullet}$ holds in $\Lambda\mu_{\mathsf{cons}}$.*

Hence, every typable term in either $\lambda^{\to}$, $\lambda\mu$, or $\Lambda_{\mathcal{S}}$ is typable also in $\Lambda\mu_{\mathsf{cons}}$. On the other hand, due to the recursive stream types, there is a λ-term which is typable in $\Lambda\mu_{\mathsf{cons}}$, and not typable in $\lambda^{\to}$. An example of such terms is $x(y(zw))(y(zww))$, where zw and zww can have the same type in the typed $\Lambda\mu_{\mathsf{cons}}$ under the context $z : [X] \to X, w : X$.

Gaboardi and Saurin [10] proposed another type system $\Lambda_{\mathcal{S}}$ as an extension of the type system in [15,21], equipped with the recursive types and coercion operator from streams to terms, which enables to represent functions returning streams such as cdr. We can define a translation from $\Lambda\mu_{\mathsf{cons}}$ to $\Lambda_{\mathcal{S}}$ preserving typability.

Acknowledgments. The authors would like to thank Kentaro Kikuchi, Shogo Ehara, and the anonymous referees for their helpful comments.

References

1. Amadio, R., Cardelli, L.: Subtyping recursive types. ACM Transactions on Programming Languages and Systems 15(4), 575–631 (1993)
2. Ariola, Z.M., Klop, J.W.: Equational term graph rewriting. Fundamenta Informaticae 26(3-4), 207–240 (1996)
3. Carraro, A.: The untyped stack calculus and Böhm's theorem. In: 7th Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2012). Electric Proceedings in Theoretical Computer Science, vol. 113, pp. 77–92 (2012)
4. Carraro, A., Ehrhard, T., Salibra, A.: The stack calculus. In: 7th Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2012). Electric Proceedings in Theoretical Computer Science, vol. 113, pp. 93–108 (2012)
5. David, R., Py, W.: $\lambda\mu$-calculus and Böhm's theorem. The Journal of Symbolic Logic 66, 407–413 (2001)
6. de Groote, P.: A CPS-translation of the $\lambda\mu$-calculus. In: Tison, S. (ed.) CAAP 1994. LNCS, vol. 787, pp. 85–99. Springer, Heidelberg (1994)
7. Dehornoy, P., van Oostrom, V.Z.: Proving confluence by monotonic single-step upperbound functions. In: Logical Models of Reasoning and Computation, LMRC 2008 (2008)
8. de'Liguoro, U.: The approximation theorem for the $\Lambda\mu$-calculus. Mathematical Structures in Computer Science (to appear)
9. Friedman, H.: Equality between functionals. In: Parikh, R. (ed.) Logic Colloquium, pp. 22–37 (1973)
10. Gaboardi, M., Saurin, A.: A foundational calculus for computing with streams. In: 12th Italian Conference on Theoretical Computer Science (2010)

11. Komori, Y., Matsuda, N., Yamakawa, F.: A simplified proof of the church-rosser theorem. Studia Logica 101(1) (2013)
12. Milner, R.: A complete inference system for a class of regular behaviours. Journal of Computer and System Sciences 28, 439–466 (1984)
13. Nakazawa, K.: Extensional models of typed lambda-mu caclulus (2014) (unpublished manuscript), `http://www.fos.kuis.kyoto-u.ac.jp/~knak/papers/manuscript/typed-sm.pdf`
14. Nakazawa, K., Katsumata, S.: Extensional models of untyped Lambda-mu calculus. In: Geuvers, H., de'Liguoro, U. (eds.) Proceedings Fourth Workshop on Classical Logic and Computation (CL&C 2012). Electric Proceedings in Theoretical Computer Science, vol. 97, pp. 35–47 (2012)
15. Pagani, M., Saurin, A.: Stream associative nets and $\lambda\mu$-calculus. Research Report 6431, INRIA (2008)
16. Parigot, M.: $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In: Voronkov, A. (ed.) LPAR 1992. LNCS, vol. 624, pp. 190–201. Springer, Heidelberg (1992)
17. Salomaa, A.: Two complete axiom systems for the algebra of regular events. Journal of the Associations for Computing Machinery 13(1), 158–169 (1966)
18. Saurin, A.: Separation with streams in the $\Lambda\mu$-calculus. In: 20th Annual IEEE Symposium on Logic in Computer Science (LICS 2005), pp. 356–365 (2005)
19. Saurin, A.: A hierarchy for delimited continuations in call-by-name. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 374–388. Springer, Heidelberg (2010)
20. Saurin, A.: Standardization and Böhm trees for $\Lambda\mu$-calculus. In: Blume, M., Kobayashi, N., Vidal, G. (eds.) FLOPS 2010. LNCS, vol. 6009, pp. 134–149. Springer, Heidelberg (2010)
21. Saurin, A.: Typing streams in the $\Lambda\mu$-calculus. ACM Transactions on Computational Logic 11 (2010)
22. Streicher, T., Reus, B.: Classical logic, continuation semantics and abstract machines. Journal of Functional Programming 8(6), 543–572 (1998)
23. Thielecke, H.: Categorical structure of continuation passing style. PhD thesis, University of Edinburgh (1997)
24. van Bakel, S., Barbanera, F., de'Liguoro, U.: A filter model for the $\lambda\mu$-calculus. In: Ong, L. (ed.) Typed Lambda Calculi and Applications. LNCS, vol. 6690, pp. 213–228. Springer, Heidelberg (2011)