Polymorphism and the free bicartesian closed category

Paolo Pistone, Luca Tranchini
Wilhelm-Schickard-Institut, Universität Tübingen,
Sand 13, D-72076, Tübingen (Germany)
paolo.pistone@uni-tuebingen.de
luca.tranchini@gmail.com

Abstract

We study two decidable fragments of System F, the polynomial and the Yoneda fragment, inducing two representations of the free bicartesian closed category.

The first fragment is freely generated by the encoding of finite polynomial functors (generalizing the usual embedding of finite products and coproducts) and describes a class of well-behaved polymorphic terms: unlike those in full System F, the terms typable in this fragment can be interpreted as ordinary natural transformations and are equivalent, up to permutations, to terms typable using a strictly predicative type discipline.

The second fragment is introduced to investigate the class of finite types, that is the types of System F which are isomorphic, modulo contextual equivalence, to a closed propositional type. The types of this fragment arise from a schema resembling the Yoneda isomorphism, and are shown to converge onto propositional types by a type rewriting approach.

1 Introduction

The question of whether two programs in a type system codify the same function can be answered in very different ways. One is to say that two programs are equivalent when they behave in the same way in the same contexts. While this equivalence has a simple and compact definition, it is often difficult to study. Other common ways of defining program equivalence are either syntactic, i.e. by describing a class of equations between well-typed terms (typically, \(\beta\) and \(\eta\) equations) or semantic, i.e. by considering equal those programs which are interpreted by the same entity in a given class of (usually categorical) models.

The simply-typed \(\lambda\)-calculus with finite products and coproducts, here \(\Lambda p\), is a good example of the interconnections between these approaches: \(\beta\eta\)-equivalence coincides with the equivalence generated by the interpretation of \(\Lambda p\) in bicartesian closed categories. In other words, \(\Lambda p\) corresponds, under \(\beta\eta\), to the free bicartesian closed category \(B\). Moreover, \(\beta\eta\)-equivalence and contextual equivalence coincide and are both decidable [33].

A translation which dates back to Russell [32] and Prawitz [29], and to which we will refer as the RP-translation, allows one to embed \(\Lambda p\) into System F. As is well-known, this translation maps \(\eta\)-equivalent terms of \(\Lambda p\) into polymorphic terms which are not equivalent modulo \(\beta\eta\), but only under stronger notions of equivalence (see [28, 18]).

Between \(\beta\eta\)-equivalence, which is decidable, and contextual equivalence, which is undecidable, System F admits a wide range of notions of equivalence. These arise from either denotational models (e.g. domain models [14] and realizability models [19]) or syntactic approaches (e.g. formal parametricity and bisimulations [28, 16, 27, 9]). Among these, the interpretation of polymorphic programs by dinatural transformations [4] provides a semantic notion of equivalence with a certain syntactic flavor, since dinaturality conditions can be described in a purely equational way (see...
However, the investigation of dinatural models is problematic in general, due to the well-known fact that dinatural transformations need not compose.

Goals of the paper

As it is observed in several places (including [37] and more recently [2]), the equivalences needed to map the $\eta$-rules of $\Lambda p$ into $F$ can be expressed in terms of ordinary natural transformations. These equivalences can be captured by a syntactic equational theory that we call the $\varepsilon$-theory. A natural question is whether the $\varepsilon$-theory provides a canonical and decidable notion of program equivalence for the polymorphic programs encoding finite products and coproducts, and whether a canonical interpretation for such programs can be defined in terms of natural transformations.

A related question is about type isomorphisms. Let a System F type be said finite when it is isomorphic, modulo contextual equivalence, to a closed type of $\Lambda p$. A finite type has a finite number of inhabitants, up to contextual equivalence. While the type isomorphisms holding under $\beta\eta$-equivalence in System F are known to be decidable and finitely axiomatizable [7], the larger classes of type isomorphisms holding under stronger equational theories are not yet well-understood. A second question motivating this work is thus whether one can find characterizations of the class of finite types of System F.

Contributions

In this paper we study two fragments of $F$ which correspond, modulo the $\varepsilon$-theory, to the free bicartesian closed category. By showing their equivalence with $\mathbb{B}$, we establish (1) the decidability of type inhabitation, (2) the decidability of the $\varepsilon$-theory and (3) the coincidence of the $\varepsilon$-theory with contextual equivalence in these fragments.

The polynomial fragment: naturality and atomization

The RP-translation allows one to map the binary operations $+$ and $\times$ onto binary operations defined over the types of System F. This mapping can be extended to all finite polynomial functors [12], that is, to the operations which transform a family of sets or types $(A_i)_{i \in I}$ indexed by a finite set $I$ into a set of the form

$$
\sum_{i \in I} \prod_{j \in g^{-1}(i)} A_{f(j)}
$$

(1)

determined by a diagram of finite sets $I \xrightarrow{f} A \xleftarrow{g} B$.

We obtain our first representation of $\mathbb{B}$ in $F$ by considering the fragment freely generated by the encoding of finite polynomial functors. This fragment, that we call the polynomial fragment (noted $\Lambda^{2\text{Poly}}$), provides a natural environment to answer our first question.

We show two properties which seem peculiar to the fragment $\Lambda^{2\text{Poly}}$: the existence of a canonical interpretation of polymorphic programs as ordinary natural transformations and the existence of a predicative description of $\Lambda^{2\text{Poly}}$ and of its program equivalence.

First, we introduce a refined type system for $\Lambda^{2\text{Poly}}$, tight to the specific form of the universally quantified types in this fragment. We show that all terms typable in this system yield syntactic natural transformations modulo the $\varepsilon$-theory.

Then we show that any term in $\Lambda^{2\text{Poly}}$ can be transformed, through $\varepsilon$-equations, into one whose type instantiations are all atomic. As a consequence, we obtain a faithful embedding of $\Lambda^{2\text{Poly}}$ into the atomic fragment $\Lambda^{2\text{at}}$ of System F [11].

The Yoneda fragment: rewriting System F types into propositional types

In the second part of the paper we address type isomorphisms. Our analysis of $\Lambda^{2\text{Poly}}$ shows that all types in this fragment are isomorphic, up to $\varepsilon$-equivalence, to propositional types. However, finite System F types are not restricted to those of $\Lambda^{2\text{Poly}}$. 
t, u := x | λx. t | tu | ΛX.t | tA | ⟨t, u⟩ | π^A_i t | * | i_A t | δ_A(t, y, u_1, y, u_2) | ξ_A t \quad (i = 1, 2, A \text{ type})

Figure 1: Terms of the full polymorphic λ-calculus.

\[ \begin{array}{c}
\Gamma, x : A \vdash x : A \\
\Gamma, x : A \vdash t : B \\
\Gamma \vdash \lambda x : A t : A \rightarrow B \\
\Gamma \vdash tu : B \\
\Gamma \vdash t : A \rightarrow B \\
\Gamma \vdash x : A \\
\Gamma \vdash \forall X. A \\
\Gamma \vdash t : B : A[B/X] \\
\Gamma \vdash \lambda \xi : A_1 \times A_2 \xi t : A_1 \\
\Gamma \vdash \iota t : A_1 + A_2 \\
\Gamma \vdash \delta_C(t, y, u_1, y, u_2) : C \\
\Gamma \vdash * : 1 \\
\Gamma \vdash t : 0 \\
\Gamma \vdash \xi_A t : A
\end{array} \]

Figure 2: Typing rules for A2p

Our second fragment \( \Lambda_2_{\text{Yon}} \), that we call the \textit{Yoneda fragment}, arises from the observation that the isomorphism between a finite polynomial functor and its second order translation can be proved from standard isomorphisms of \( \Lambda_p \) plus the isomorphism \( A \equiv \forall X.(A \rightarrow X) \rightarrow X \). This isomorphism, which holds under the \( \varepsilon \)-theory but not under the \( \beta\eta \)-theory, is an instance of the following \textit{Yoneda schema}:

\[ F[A/X] \equiv \forall X.(A \rightarrow X) \rightarrow F \tag{YS} \]

where \( X \) is not free in \( A \) and occurs positively in \( F \). As the universal quantifier corresponds, modulo the \( \varepsilon \)-theory, to an end (see [22]), the schema \( \text{(YS)} \) translates the Yoneda isomorphism \( F(a) \equiv \int \text{Hom}(\text{Hom}(a, x), F(x)) \).

When read from left to right, the schema \( \text{(YS)} \) yields a type-rewriting rule eliminating a quantifier. We show that all \( \Lambda_2_{\text{Yon}} \)-types converge to propositional types by means of a rewriting which generalizes the Yoneda schema and replaces a second order quantifier by a finite polynomial functor. Hence we establish that any closed type in \( \Lambda_2_{\text{Yon}} \) is finite. The fragment \( \Lambda_2_{\text{Yon}} \) does not capture all finite types of System F, and we discuss some natural extension of our rewriting in the concluding section.

Related work

A clear description of the connection between the second order codings of finite data types and the (di)naturality conditions is in [13]. This topic has recently attracted new attention due to [2], where such naturality conditions are described at the level of \textit{propositional identity} using ideas from Homotopy Type Theory.

The “Yoneda restriction” of System \( \Lambda_2_{\text{Yon}} \) can be related to other approaches in the literature. A similar restriction was exploited by the first author to describe a decidable theory of program equivalence over a fragment of Second Order Multiplicative Linear Logic [20]. A related restriction appears in [6] to describe a class of polymorphic types for which program equivalence can be characterized by a finite testing.

The atomization property of \( \Lambda_2_{\text{poly}} \) is a generalization of the \textit{instantiation overflow} property investigated in [10] for \( \Lambda_2_{\text{at}} \). A characterization of the System F types satisfying this property is in [22]. We discuss this connection in some more detail in Section 5.

2 Preliminaries

Let \( \Lambda_2 \) be the \textit{full polymorphic \( \lambda \)-calculus}, whose types are generated from a countable set \( V \) of variables, the constants 0, 1 and the connectives \( \rightarrow, +, \times \), as well as second order quantification \( \forall \),
Figure 3: RP-translation of types and terms.

and whose terms are generated by the grammar in Fig. 1. The typing rules are in Fig. 2. The second-order \( \lambda \)-calculus \( \Lambda 2 \), i.e. System \( F \) \ref{fn:13}, is the sub-system of \( \Lambda 2 p \) obtained by restricting type constructors to \( \to, \forall \) and term constructors to \( \lambda, \Lambda, t u, t A \). The full simply typed \( \lambda \)-calculus \( \Lambda p \) is the sub-system obtained by restricting type constructors to 0, 1, \( \to \), \( \times \) and term constructors to \( \lambda, t u, \langle t, u \rangle, \pi^C, \iota_i, \delta_A, \ast, \xi_A \).

For any type system \( S \), we write \( T(S) \) for the set of its types, and \( \Gamma \vdash S t : A \) to indicate that the judgement \( \Gamma \vdash t : A \) is derivable in \( S \).

In Fig. 3 we recall the RP-translation \([\cdot]^\ast\) of \( \Lambda 2 p \) into \( \Lambda 2 \). It is easily checked by induction that \( \Gamma \vdash \Lambda 2 p t : A \) implies \( \Gamma^\ast \vdash \Lambda 2 t^\ast : A^\ast \).

2.1 Theories of program equivalence

We let \( S \) indicate any among \( \Lambda 2 p, \Lambda 2, \Lambda p \) or a fragment of these.

Definition 2.1 (theories). A well-typed equation in \( S \) is an expression of the form \( \Gamma \vdash S t \approx u : A \), such that \( \Gamma \vdash S t, u : A \). A theory \( T \) over \( S \) is a set of well-typed equations \( \Gamma \vdash S t \approx_T u : A \) over \( S \), closed with respect to usual congruence rules like

\[
\begin{align*}
\Gamma, x : A \vdash S t & \approx_T u : B \\
\Gamma \vdash S \lambda x.t & \approx_T \lambda x.u : A \to B 
\end{align*}
\]

To describe usual theories it is useful to introduce contexts:

Definition 2.2. Contexts \( C \) are defined by the same grammar as terms, plus the constructor \([\cdot]\).

If \( C \) is a context and \( t \) is a term, we let \( C[t] \) be the term obtained by variable-binding substitution of \( t \) for \([\cdot]\) in \( C \).

If \( S \) is any of \( \Lambda 2 p, \Lambda 2, \Lambda p \), we let \( C : (\Gamma \vdash S A) \Rightarrow (\Gamma' \vdash S A') \) when for all \( \Gamma \vdash S t : A \), \( \Gamma' \vdash S C[t] : A' \).\footnote{One can deduce explicit typing rules for such judgements from the typing rules of \( S \), see for instance \cite{17}}

We let \( C : A \vdash S B \) be a shorthand for \( C : (\vdash S A) \Rightarrow (\vdash S B) \) and we indicate by \( C : A \vdash S B \) that, for some \( \Gamma, C : A \vdash S B \).

Let \( C \circ D \) be shorthand for \( C[D] \). It is clear that if \( D : (\Gamma \vdash S A) \Rightarrow (\Gamma' \vdash S A') \) and \( C : (\vdash S A) \Rightarrow (\vdash S A') \), then \( C \circ D : (\Gamma \vdash S A) \Rightarrow (\vdash S A') \).

The following families of contexts will be used in the next sections:

Definition 2.3. (elimination and introduction contexts). Let \( C \in T(\Lambda 2) \) be \( C = \forall \bar{Y}_1.C_1 \to \forall \bar{Y}_2.C_2 \to \cdots \to \forall \bar{Y}_n.C_n \to \forall \bar{C}_{n+1}.Z \). For all sequence of \( m \geq n \) distinct variables \( \bar{z} = z_1, \ldots, z_m \), let \( \Sigma^C_\bar{z} = \{ z_1 : C_1, \ldots, z_n : C_n \} \). We define the contexts \( E_1^C : (\vdash \Lambda 2 C) \Rightarrow (\vdash \Sigma^C_\bar{z} \vdash \Lambda 2 Z) \) and \( I_n^C : (\Sigma^C_\bar{z} \vdash \Lambda 2 Z) \Rightarrow (\vdash \Lambda 2 C) \) by induction on \( C \) as follows:

- if \( C = Z \) and we let \( E_1^Z = [\ ] \) and \( I_n^Z = [\ ] \);
- if \( C = D \to E \), then \( E_1^{Z,E} = E_1^D[[\ ] ] z \) and \( I_n^{Z,E} = \lambda z. I_n^D \);
- if \( C = \forall Y.C' \), then \( E_1^C = E_1^{\bar{Y}C'} \) and \( I_n^C = \Lambda Y. I_n^{\bar{Y}C'} \).
\[(\lambda x.t)u \cong t[u/x] \quad (\Lambda X.t)B \cong t[B/X] \quad \left(\pi^1_{\eta}(t_1, t_2) \cong t_i\right)_{i=1,2} \quad \left(\delta_{C}(t_1, y, u_1, y, u_2) \cong u_{i}[t/y]\right)_{i=1,2}\]

\[
\begin{align*}
\Gamma \vdash t : A &\rightarrow B \\
\Gamma \vdash t \cong \lambda x.t x : A &\rightarrow B \\
\Gamma \vdash t : A \times B &\rightarrow C \\
\Gamma, \Delta \vdash C[t] &\cong \delta_{C}(t, y, C[y, y], y, C[y, y]) : C \\
\Gamma \vdash t : \forall X.A &\rightarrow \forall X.A \\
\Gamma \vdash u : 0 &\rightarrow \Gamma A \\
\Gamma \vdash C[u] &\cong \xi_{A!u} : A
\end{align*}
\]

Figure 4: \(\beta\) and \(\eta\)-rules for \(\Lambda 2p\).

The \(\beta, \eta\) and \(\beta\eta\)-theories for \(S\) are the smallest theories generated by the \(\beta\) and \(\eta\)-rules for \(\Lambda 2p\) recalled in Fig. 4.

**Remark 2.1.** While the embedding \([\_]^\ast\) preserves the \(\beta\)-rules, it does not preserve the \(\eta\)-rules. For instance, let \(t = \lambda x.\lambda y.\delta(y, z.z, z.z)x\) and \(u = \lambda x.\lambda y.\delta(y, z.z, z.z)\) be two closed terms of type \(D = A \rightarrow ((A \rightarrow C) + (A \rightarrow C)) \rightarrow C\). Then \(\vdash_{\Lambda p} t \cong_{\eta} u : D\) but \(t^\ast\) and \(u^\ast\) have distinct \(\beta\eta\)-normal forms, as shown by a simple calculation.

We recall the definition of contextual equivalence, here limited to closed terms:

**Definition 2.4** (contextual equivalence). Let \(S\) be any among \(\Lambda p\), \(\Lambda 2p\) and its fragments, and \(S'\) be any among \(\Lambda 2\) and its fragments.

i. If \(\vdash_{S} t, u : A\) we let \(t \cong_{ctx} u\) when for all context \(C : A \vdash S 1 + 1\), \(C[t] \cong_{\beta} C[u]\).

ii. If \(\vdash_{S'} t, u : A\), we let \(t \cong_{ctx} u\) when for all context \(C : A \vdash S 1 + 1\), \(\forall X.X \rightarrow X \rightarrow X, C[t] \cong_{\beta} C[u]\).

While contextual equivalence is undecidable in \(\Lambda 2p\) and \(\Lambda 2\), the following result was recently established and will play a central role in our results:

**Theorem 2.1** (\[BB\]). The theory \(\cong_{ctx}\) over \(\Lambda p\) is decidable and coincides with the \(\beta\eta\)-theory.

### 2.2 Syntactic categories and functors

Contexts provide a simple way to define syntactic categories:

**Definition 2.5** (syntactic category). Let \(T\) be a theory of \(S\) containing the \(\beta\eta\)-theory. The category \(\mathcal{C}_{T}(S)\) is defined as follows: the objects are the types of \(S\) and the arrows from \(A\) to \(B\) are the \(T\)-equivalence classes of contexts \(C : A \vdash_{S} 1 + 1\) with identity \([\ ]\) and composition given by context composition. The category \(\mathcal{C}_{T}^{0}(S)\) is the subcategory of \(\mathcal{C}_{T}(S)\) whose arrows are \(T\)-equivalence classes of contexts \(C : A \vdash_{S} 1 + 1\) with identity \([\ ]\) and composition given by context composition.

The category \(\mathcal{B} = \mathcal{C}_{\beta\eta}(\Lambda p)\) is the free bicartesian closed category.

For all \(X \in \mathcal{V}\), let \(P_{X}\) be the set of types in which \(X\) occurs only in positive position and \(N_{X}\) be the set of types in which \(X\) occurs only in negative position.

**Definition 2.6** (syntactic functors). Let \(A \in P_{X} \cup N_{X}\). For all context \(C\) we let \(A^\times(C)\) be the context defined by induction on \(A\) as follows:

- if \(A = Y \neq X\), then \(A^\times(C) = \{\}\);
- if \(A = X\), then \(A^\times(C) = C\);
- if \(A = A_{1} \rightarrow A_{2}\), then \(A^\times(C) = \lambda y.(A_{1}^\times(C))\left[xA_{1}^\times(C)[y]\right]\).

\(^2\)We omitted type informations when these can be guessed by inspecting the terms.
• if $A = \forall Y. B$, then $A^\gamma(C) = AY. B^\gamma(C)[zY]$.

**Proposition 2.2.** Let $T$ be a theory of $S$ including the $\eta$-theory. Then, for all $A \in P_X$ (resp. $A \in N_X$), $A^\nu : C_T(S) \rightarrow C_T(S)$ (resp. $A^\nu : C_T(S)^{op} \rightarrow C_T(S)$).

In the Proposition above one can replace $C_T(S)$ by $C^0_T(S)$.

We call two types $A, B \in T(S)$ T-isomorphic, written $A \equiv_T B$, if there is an isomorphism between $A$ and $B$ in $C^0_T(S)$. We stress the dependency of type isomorphisms on a theory $T$. For instance the $\eta$-isomorphisms for $\Lambda 2$ do not coincide with $\text{ctx}$-isomorphisms (a crucial aspect in Section 5).

### 3 Finite polynomial functors

The essence of the RP-translation is a mapping of the type constructors $1, 0, \times, +$, viewed as functors $\mathbb{C}_{\beta\eta}(\Lambda p)^n \rightarrow \mathbb{C}_{\beta\eta}(\Lambda p)$, onto certain functors $\mathbb{C}_{\beta\eta}(\Lambda 2)^n \rightarrow \mathbb{C}_{\beta\eta}(\Lambda 2)$ definable in terms of $\rightarrow$ and $\forall$. This mapping extends straightforwardly to all “finite polynomials”. These are elegantly described by the theory of finite polynomial functors [12], that we shortly recall.

**Definition 3.1** (finite polynomial functor). A finite polynomial functor (abbreviated f.p.f.) is a diagram in $\text{FinSet}$ of shape

$$\mathcal{I} \xleftarrow{f} A \xrightarrow{g} B \xrightarrow{h} \mathcal{J}$$

The diagram $(f, g, h)$ yields a functor $\text{Set}^\mathcal{I} \rightarrow \text{Set}^\mathcal{J}$ given by

$$(X_i)_{i \in I} \mapsto \left( \sum_{i \in h^{-1}(j)} \prod_{i \in g^{-1}(i)} X_j(i) \right)_{j \in \mathcal{J}}$$

We call the set $\mathcal{I}$ the base of the functor. We will restrict attention to finite polynomial functors $\text{FinSet}^\mathcal{I} \rightarrow \text{FinSet}$. As $\mathcal{J}$ is a singleton we can omit the constant arrow $h$.

**Example 3.1.** The finite polynomial functor $3 \xrightarrow{f} 4 \xrightarrow{g} 2$, where $2 = \{0, 1\}$, $3 = 2 \cup \{3\}$, $4 = 3 \cup \{4\}$ and with $f : \{1, 3 \rightarrow 1; 2 \rightarrow 2; 4 \rightarrow 3\}$ and $g : \{1, 2 \rightarrow 1; 3, 4 \rightarrow 2\}$, maps each $(X_i)_{i \in 3}$ onto $(X_1 \times X_2) + (X_1 \times X_3)$. The finite polynomial functor $2 \xrightarrow{id_2} 2 \xrightarrow{id_2} 2$ corresponds to the coproduct $(X_i)_{i \in 2} \mapsto X_0 + X_1$. The finite polynomial functor $2 \xrightarrow{id_2} 2 \xrightarrow{eq_0} 1$, where $1 = \{0\}$ and $c_0$ is constant, corresponds to the product $(X_i)_{i \in 2} \mapsto X_0 \times X_1$.

### 3.1 Finite polynomial functors in $\Lambda p$ and $\Lambda 2$

We show how finite polynomial functors yield functors over $\mathbb{C}_{\beta\eta}(\Lambda p)$ and $\mathbb{C}_{\beta\eta}(\Lambda 2)$.

**Remark 3.1.** We will consider $\Lambda p$-types up to the associativity $\beta\eta$-isomorphisms $(A + B) + C \equiv_{\beta\eta} A + (B + C)$ and $(A \times B) \times C \equiv_{\beta\eta} A \times (B \times C)$. Given a finite linearly ordered set $I = \{i_1 < \cdots < i_k\}$ and a family of types $(A_i)_{i \in I}$, it thus makes sense to speak of the $I$-indexed sums $\sum_{i \in I} A_i$ (equal to $0$ if $I = \emptyset$) and product $\prod_{i \in I} A_i$ (equal to 1 if $I = \emptyset$).

Indexed sums and products come with constructors $i_*^k$ (for $i = 1, \ldots, k$) and $\langle t_1, \ldots, t_k \rangle$ and destructors $\delta_i^k : \langle t, (x \cdot u_i)_{i = 1, \ldots, k} \rangle$ and $\pi_i^k$ (for $i = 1, \ldots, k$), with obvious $\beta$ and $\eta$-rules. All these operators can be defined explicitly from the terms of $\Lambda p$ (and their $\beta$ and $\eta$-rules derived from those of $\Lambda p$), by fixing a representative of each associativity-classes of types.

To describe polynomials in terms of indexed sums and products in $\Lambda p$, it is convenient to replace finite sets with finite linear orders. Let $\text{FinLin}$ be the category of finite linear orders and monotone functors.
Definition 3.2. An ordered f.p.f. is a diagram $I \xrightarrow{f} A \xrightarrow{g} B \xrightarrow{h} J$ in FinLin.

Observe that the linear order of $A$ induces unique linear orders on the sets $g^{-1}(i)$, for $i \in B$. It is straightforward that an ordered f.p.f. yields a functor $\text{FinLin}^I \to \text{FinLin}^J$.

Proposition 3.1. Let $I \xrightarrow{f} A \xrightarrow{g} B$ be an ordered f.p.f. Then there exists a functor $\mathfrak{P}_{f,g} : \mathbb{C}_{\beta\eta}(\text{Ap})^I \to \mathbb{C}_{\beta\eta}(\text{Ap})$ given by

$$\mathfrak{P}_{f,g}(A_i) := \sum_{i \in B} \prod_{j \in g^{-1}(i)} A_{f(j)}$$

and $\mathfrak{P}_{f,g}(C_i) = \delta^k(\{1\}, (x, \langle C_{f(a_1)}[x], \ldots, C_{f(a_k)}[x] \rangle)_{i=1}^{\ldots, a})$.

To describe finite polynomials in $\Lambda^2$ we adopt the following exponential notation:

Definition 3.3. Let $I = \{i_1 \leq \cdots \leq i_k\}$ be a finite linear order and $A_i$ be an $I$-indexed family of $\Lambda^2$-types. For all $\Lambda^2$-type $B$, we let

$$\text{EXP}_{\text{poly}}^B(A_i) = A_{i_1} \to \cdots \to A_{i_k} \to B$$

Proposition 3.2. Let $I \xrightarrow{f} A \xrightarrow{g} B$ be an ordered f.p.f. Then there exists a functor $\mathfrak{U}_{f,g} : \mathbb{C}_{\beta\eta}(\Lambda^2)^I \to \mathbb{C}_{\beta\eta}(\Lambda^2)$ given by

$$\mathfrak{U}_{f,g}(A_i) := \forall X. \exp^{X}_{\text{poly}}(\exp^{X}_{g^{-1}(i)}(A_{f(i)}))$$

where for all $i \in B$, $f_i : g^{-1}(i) \to I$ is the restriction of $f$ to $g^{-1}(i)$, $X$ is a fresh variable, and

$$\mathfrak{U}_{f,g}(A_i) = \Lambda X. \lambda y_1 \ldots y_k. \{X, R_1 \ldots R_k \mid k = \text{card}(B) \text{ and } R_i = \lambda z_1 \ldots z_k. y_i(C_{f(a_k)}[z_1]) \ldots (C_{f(a_k)}[z_k]) \text{ with } k_i = \text{card}(g^{-1}(i)) \text{ and } g^{-1}(i) = \{a_1 < \cdots < a_k\}.$$  

Remark 3.2. When $\mathfrak{P}_{f,g}$ is a binary product or coproduct (see Example 3.1), then for all indexed family of types $(A_i)_{i \in I}$, the type $\mathfrak{U}_{f,g}(A_i)$ coincides with $(\mathfrak{P}_{f,g}(A_i))^*$, that is, $\mathfrak{U}_{id_2, id_2}(A_i) = (A_1 + A_2)^*$ and $\mathfrak{U}_{id_2, co}(A_i) = (A_1 \times A_2)^*$.

This is however not true in general. Consider $\mathfrak{P}_{f,g}(A_i) = (A_1 \times A_2) + (A_1 \times A_3)$. $\mathfrak{P}_{f,g}$ and $\mathfrak{U}_{f,g}$ are given by the diagram $3 \xrightarrow{f} 4 \xrightarrow{g} 2$. While $\mathfrak{U}_{f,g}(A_i) = \forall X. (A_1 \to A_2 \to X) \to (A_1 \to A_3 \to X) \to X$, $(\mathfrak{P}_{f,g}(A_i))^*$ is the type $\forall X. \forall Y. ((A_1 \to A_2 \to Y) \to (A_1 \to A_3 \to Y) \to (A_1 \to Y) \to X) \to X$. Nevertheless, when all $A_i$ are in $\Lambda^2_{\text{poly}}$, the types $(\mathfrak{P}_{f,g}(A_i))^*$ are in $\Lambda^2_{\text{poly}}$, as the latter is closed under substitution.

The relationship between the $\Lambda^2$-types $\mathfrak{U}_{f,g}(A_i)$ and $(\mathfrak{P}_{f,g}(A_i))^*$ will be clarified in the next subsection by extending the $\beta\eta$-theory of $\Lambda^2$. As a preliminary observation we establish the following connection between the functors $\mathfrak{P}_{f,g}$ and $\mathfrak{U}_{f,g}$ in $\Lambda^2$:

Lemma 3.3. For all ordered f.p.f. $I \xrightarrow{f} A \xrightarrow{g} B$, $\mathfrak{U}_{f,g}(A_i) = \beta\eta \forall X. (\mathfrak{P}_{f,g}(A_i) \to X) \to X$.

Proof. $\forall X. \exp^{X}_{\text{poly}}(\exp^{X}_{g^{-1}(i)}(A_{f(i)})) = \beta\eta \forall X. \exp^{X}_{\text{poly}}(\prod_{j \in g^{-1}(i)} A_{f(j)}) \to X \equiv \beta\eta \forall X. (\prod_{i \in B} \prod_{j \in g^{-1}(i)} A_{f(j)} \to X) \equiv \beta\eta \forall X. (\sum_{i \in B} \prod_{j \in g^{-1}(i)} A_{f(j)} \to X) \to X$.  

3.2 The polynomial fragment of $\Lambda^2$

The fragment $\Lambda^2_{\text{poly}}$ is generated by the smallest set of $\Lambda^2$-types closed with respect to $\to$ and finite polynomial functors. Formally:

Definition 3.4 (System $\Lambda^2_{\text{poly}}$). The set $\text{Poly} \subseteq T(\Lambda^2)$ is defined inductively by (1) $\forall \subseteq \text{Poly}$, (2) if $A, B \in \text{Poly}$, then $A \to B \in \text{Poly}$ and (3) if $I \xrightarrow{f} A \xrightarrow{g} B$ is an ordered f.p.f. and $(A_i)_{i \in I} \in \text{Poly}^I$, then $\mathfrak{U}_{f,g}(A_i) \in \text{Poly}$. We let $\Lambda^2_{\text{poly}}$ be the fragment of $\Lambda^2$ with types $\text{Poly}$.
\[ \Gamma \vdash \Lambda_{2\text{poly}} \; t : A \quad (\Delta \vdash \Lambda_{2\text{poly}} \; u_i : A_i[C/X])_{i=1,\ldots,k} \quad C : (\Delta \vdash \Lambda_{2\text{poly}} \; C) \Rightarrow (\Sigma \vdash \Lambda_{2\text{poly}} \; D) \]

\[ \Gamma, \Sigma \vdash [tC u_1 \ldots u_n] \approx_{\varepsilon} tD(A_i^1(C)[u_1]) \ldots (A_i^k(C)[u_k]) : D \]

Figure 5: \(\varepsilon\)-rule for \(A = \forall X.A_1 \rightarrow \cdots \rightarrow A_k \rightarrow X\).

As the set Poly is closed by substitution, the fragment \(\Lambda_2\) is well-defined and closed with respect to \(\beta\) and \(\eta\)-rules. We now introduce the \(\varepsilon\)-theory for \(\Lambda_{2\text{poly}}\).

**Definition 3.5** (\(\varepsilon\)-theory). We let \(\approx_{\varepsilon}\) be the smallest theory over \(\Lambda_{2\text{poly}}\) containing \(\beta, \eta\) and all equations in Fig. 5, where \(A = \forall X.A_1 \rightarrow \cdots \rightarrow A_k \rightarrow X \in \text{Poly}\).

Similarly to the \(\eta\)-rules for sum types, the \(\varepsilon\)-rule allows to permute contexts within a polymorphic term. It is indeed not difficult to see that the \(\eta\)-rules translate into \(\varepsilon\)-rules through the second order embedding.

**Remark 3.3.** For readability, we will indicate that an equation \(t \approx_{\varepsilon} t'\) results from an application of the rule in Fig. 5 permuting context \(C\) by \(t \approx_{\varepsilon} t'\).

**Remark 3.4.** The \(\varepsilon\)-equations strictly extend the \(\beta\eta\)-theory. For instance, let \(C = \lambda y.[] : B \vdash \Lambda_{2\text{poly}} A \rightarrow B\). Then we can deduce the equation \(x : \forall Y.Y \rightarrow Y, z : B \vdash \lambda y.(xBz) \approx_{\varepsilon} x(A \rightarrow B)\lambda y.z : A \rightarrow B\) which does not hold under the \(\beta\eta\)-theory.

The following is a simple application of \(\varepsilon\)-theory, that will be generalized in Section 6.

**Lemma 3.4.** For all \(A \in \text{Poly}\), if \(X \not\in \text{FV}(A)\), \(A \equiv_{\varepsilon} \forall X.(A \rightarrow X) \rightarrow X\).

**Proof.** Let \(C = \Lambda X.\lambda f.f[] : A \vdash \Lambda_{2\text{poly}} B\) and \(D = [\ ] \lambda X.\lambda f.f([\ ]A\lambda x.x) \vdash \Lambda_{2\text{poly}} A\), with \(B = \forall X.(A \rightarrow X) \rightarrow X\). Then \(C \circ D \equiv_{\beta} [\ ]\) and \(D \circ C = \Lambda X.\lambda f.f([\ ]A\lambda x.x) \approx_{\varepsilon} \Lambda X.\lambda f.f([\ ]X \lambda x.fx) \approx_{\eta} [\ ]\).

Let \(p\text{Poly}\) be the set of types obtained by extending Poly with finite products and coproducts, and \(\Lambda_{2\text{pPoly}}\) the fragment of \(\Lambda_2\) generated by \(p\text{Poly}\). By taking \(\varepsilon\) as a theory of \(\Lambda_{2\text{pPoly}}\), we deduce the following from Lemma 3.3 and Lemma 3.4.

**Proposition 3.5.** In \(\Lambda_{2\text{pPoly}}\), \(\Psi_{f,g}(A_i) \equiv_{\varepsilon} \Upsilon_{f,g}(A_i)\).

From Remark 3.2 we can deduce that the RP-translation yields an embedding of \(\Lambda p\) into \(\Lambda_{2\text{poly}}\). By exploiting the \(\varepsilon\)-theory we can prove the following facts:

**Theorem 3.6.**

i. If \(\Gamma \vdash \Lambda p \; t \approx_{\beta\eta} u : A\) then \(\Gamma^* \vdash \Lambda_{2\text{poly}} \; t^* \approx_{\varepsilon} u^* : A^*\).

ii. For all \(\Lambda_2\)-type \(A\), there exists a \(\varepsilon\)-isomorphism between \(A\) and \(A^*\).

**Proof.** Claim i. can be deduced from the results in [28] and [13], as the \(\varepsilon\)-rules are particular instances of the dinaturation condition for System F terms. A more detailed argument can be found in [37]. The isomorphisms of Claim ii. are described in Appendix A.

**Remark 3.5.** While \((\Psi_{f,g}(A_i))^*\) and as \(\Upsilon_{f,g}(A_i)\) are not the same type (Remark 3.2), they are \(\varepsilon\)-isomorphic, since \(\Upsilon_{f,g}(A_i)^* \equiv_{\varepsilon} \Psi_{f,g}(A_i)^*\).
4 Polymorphic terms as natural transformations

The $\varepsilon$-equation of Fig. 5 reads informally as a naturality condition for polymorphic terms. For instance, let $\Upsilon_{f,g}(A_i)$ be the type $\forall X. (A \rightarrow B \rightarrow X) \rightarrow X$; then the equations

$$\left( \begin{array}{c} D[xC[ ]] \cong^D xD\lambda aD[[ ]\,ab]\end{array} \right)_{C,D\in\text{Poly},\, D: C\rightarrow \Lambda_{\text{Poly}}D}$$

express the fact that the family of contexts $(C^C : A \rightarrow B \rightarrow C \vdash^{x\Upsilon_{f,g}} C)_{C\in\text{Poly}}$, given by $C^X = x.X[ ]$, defines a natural transformation between the functor $A \rightarrow B \rightarrow X^X$ and the identity functor $X^X$:

$$\begin{array}{c} \Upsilon_{f,g}(A_i) \times (A \rightarrow B \rightarrow C) \xrightarrow{\varepsilon C[ ]} C \\
\Upsilon_{f,g}(A_i) \times (A \rightarrow B \rightarrow D) \xrightarrow{\varepsilon D[ ]} D \end{array}$$

In this section we show that the correspondence between polymorphic terms and natural transformations can be extended to all the fragment $\Lambda_{\text{Poly}}$. To obtain this we will need to describe a refined type system, based on the notion of strictly-positive type.

4.1 Strictly-positive types and the system $\Lambda^*_{\text{Poly}}$

The types of $\Lambda_{\text{Poly}}$ are easily described in terms of strictly positive types:

**Definition 4.1** (strictly-positive types). For any $X \in \forall$, we define the set $\text{SP}_X \subseteq \Gamma(\Lambda_{2p})$, whose elements are called strictly-positive in $X$, inductively as follows:

i. for all $Y \in \forall$, $Y \in \text{SP}_X$;

ii. if $A \in \text{SP}_X$ and $X \notin \text{FV}(B)$, then $B \rightarrow A \in \text{SP}_X$;

iii. if $A \in \text{SP}_X \cap \text{SP}_Y$ and $Y \neq X$, then $\forall Y.A \in \text{SP}_X$.

We let $\text{SSP}_X \subseteq \text{SP}_X$ be the set of those $A \in \text{SP}_X$ such that $X \in \text{FV}(A)$.

**Lemma 4.1.** $\forall X.A \in \text{Poly}$ iff for some $k \in \mathbb{N}$ and types $A_1, \ldots, A_k \in \text{Poly} \cap \text{SSP}_X$, $A = A_1 \rightarrow \cdots \rightarrow A_k \rightarrow X$.

**Remark 4.1.** When $A \in \text{SSP}_X$, the functor $A^X$ is given by a chain of eliminations followed by a chain of introductions, that is $A^X(C) = \text{In}_A^X \circ C \circ \text{El}_A^X$.

We introduce now a refined type system in which all judgements are made of strictly positive types. We let $\text{Poly}^\varepsilon$ be the set of types obtained by enriching $\text{Poly}$ with a countable set $\mathbb{C}$ of type constants $p, q, r, \ldots$.

**Definition 4.2.** Let $\alpha \subseteq f_{in} \forall$. We let $\text{SP}_\alpha \subseteq \text{Poly}^\varepsilon$ be defined by $A \in \text{SP}_\alpha$ if $\text{FV}(A) \subseteq \alpha$ and for all $X \in \alpha$, either $A \in \text{SP}_X$ or $X \notin \text{FV}(A)$. We let $\Gamma \in \text{SP}_\alpha$ indicate that for all type $A$ appearing in $\Gamma$, $A \in \text{SP}_\alpha$. We will also indicate by $\alpha \neq A$, that for all $X \in \alpha$, $X \notin \text{FV}(A)$.

By $\Gamma \vdash^\alpha t : A$ we indicate a judgement, called $\text{SP}_\alpha$-judgement, such that $\Gamma, A \in \text{SP}_\alpha$. For $X \in \forall$ and $\alpha \subseteq f_{in} \forall$, we let $\alpha + X$ and $\alpha - X$ be shorthands for $\alpha \cup \{X\}$ and $\alpha \setminus \{X\}$.

**Definition 4.3.** We let $\Lambda^*_{\text{Poly}}$ be the type system with types $\text{Poly}^\varepsilon$, and typing rules given in Fig. 5.
Remark 4.2. The systems $\Lambda^2_{\text{poly}}$ and $\Lambda^2_{\text{poly}}^*$ do not type the same terms. For instance, take $t = \Lambda X.\lambda z.x(X \to X) \lambda y.y z$; we have $x : \forall X.X \to X \vdash t : \forall X.X \to X$ but we cannot derive the same typing in $\Lambda^2_{\text{poly}}$. Indeed, to type the last abstraction $\Lambda X$ we need to derive the judgment $x : \forall X.X \to X, z : X \vdash (\lambda y.y)_z : X$, but this is not a $\text{SP}_{\{X\}}$-judgement, as $X \to X \notin \text{SP}_{\{X\}}$.

Observe that, unless $\Delta = \emptyset$, the category $L^\alpha$ is not closed (nor cartesian), since from $A, B \in \text{SP}_\alpha$ it need not follow $A \to B \in \text{SP}_\alpha$ (take $A = B = X$).

If $A \in \text{SP}_X$, for all $\alpha$ and $\Gamma \in \text{SP}_\alpha$, the map $A^\alpha$ yields a functor $A^\alpha : L^\alpha \to L^\alpha$. When a type variable $X$ is clear from the context, we will abbreviate $t[C/X]$ by $t^C$. Also, we will employ the following useful abbreviation:

Definition 4.6 (contextual composition). Given $\Gamma \vdash t : A$, where $\Gamma = \{x_1 : A_1, \ldots, x_n : A_n\}$ and given terms $\Delta \vdash u_i : A_i$ for some $\Delta$, and terms $u_i$, for $i = 1, \ldots, n$, we let

$$t \circ_m (u_1, \ldots, u_n) := t[u_1/x_1, \ldots, u_n/x_n]$$
We will also let \( \Gamma^\alpha(\mathcal{C}) = (A_1^\alpha(\mathcal{C})[x_1], \ldots, A_n^\alpha(\mathcal{C})[x_n]) \).

**Definition 4.7** (syntactic natural transformation). For all \( \Gamma, A \in \mathcal{SP}_\alpha \), a syntactic natural transformation from \( \Gamma^\alpha \) to \( A^\alpha \) is a term \( t \) such that \( \Gamma \vdash_{\Lambda_{2\text{poly}}}^\alpha t : A \) and for all \( \mathcal{C} : C \vdash_{\Lambda_{2\text{poly}}}^\Delta D \), \( \Gamma[C/X], \Delta \vdash A^\alpha(\mathcal{C})[t]^C \simeq \varepsilon t^D \circ_m \Gamma^\alpha(\mathcal{C}) : A[D/X] \).

**Theorem 4.4** (naturality). If \( \Gamma \vdash_{\Lambda_{2\text{poly}}}^\alpha t : A \), then \( t \) is a syntactic natural transformation from \( \Gamma^\alpha \) to \( A^\alpha \).

Proof. We argue by induction on a typing derivation of \( \Gamma \vdash_{\Lambda_{2\text{poly}}}^\alpha t : A \). We only consider the most significant case, that is the one of the extraction rule: \( \Gamma \vdash_{\Lambda_{2\text{poly}}}^\alpha t : A \) is deduced by

\[
\Gamma \vdash_{\Lambda_{2\text{poly}}}^\alpha t \colon \forall X.A \quad \left( \Gamma \vdash_{\Lambda_{2\text{poly}}}^\alpha u_i : A_i[B/X] \right)_{i=1,\ldots,k}, \text{ where } A = A_1 \to \cdots \to A_k \to Y. \text{ Let } B^\dagger = B[C/X] \text{ and } B^\dagger = B[D/X]. \text{ We must show }
\]

\[
\Gamma[C/X], \Delta \vdash_{\Lambda_{2\text{poly}}}^\alpha B^\alpha(\mathcal{C})[(tB_1 \ldots u_k)^C] \simeq \varepsilon (tB_1 \ldots u_k)^D \circ_m \Gamma^\alpha(\mathcal{C}) : B^\dagger
\]

By a \( \varepsilon \)-equation we can compute \( B^\alpha(\mathcal{C})[(tB_1 \ldots u_k)^C] \) \( \simeq \varepsilon (t^B \circ_m \Gamma^\alpha(\mathcal{C}))^B \simeq \varepsilon ((tB)^D \circ_m \Gamma^\alpha(\mathcal{C}))[(u_1 \ldots u_k)^C] \). Moreover, as \( A_i \in \mathcal{SP}_X \), for \( i = 1, \ldots, k \), we deduce that \( \alpha \neq A \). Hence, from the induction hypothesis we also have \( t^B \circ_m \Gamma^\alpha(\mathcal{C})[(u_1 \ldots u_k)^C] \simeq \varepsilon ((tB)^D \circ_m \Gamma^\alpha(\mathcal{C}))[(u_1 \ldots u_k)^C] = (tB^\dagger u_1 \ldots u_k)^D \circ_m \Gamma^\alpha(\mathcal{C}) \). \( \square \)

### 5 Atomization

A salient feature of the \( \varepsilon \)-equation in Fig. 4 is that it allows one to modify the type instantiations occurring in a term. In this section we exploit this fact to show that any term in \( \Lambda_{2\text{poly}} \) can be permuted in a term in which all type instantiations are atomic. Along with the fact that permutations between \( \Lambda_{2\text{poly}} \) translate into *atomic* permutations of their atomizations, this yields an equivalence-preserving translation of \( \Lambda_{2\text{poly}} \) into \( \Lambda_{2\text{nat}} \), the predicative fragment of System \( \Gamma \) which only admits atomic type instantiations.

Atomization relies on the following lemma:

**Lemma 5.1**. Let \( \mathfrak{U}_{f,g}(A_i) = \forall X.A = \forall X.A_1 \to \cdots \to A_k \to X. \) For all \( \mathcal{C} \in \mathcal{T}(\Lambda) \) there exists a context \( \mathcal{A}_{\mathcal{C}}^{X,A} : \forall X.A \vdash_{\Lambda_2} A[C/X] \) such that \( x : \forall X.A \vdash_{\Lambda_2} xC \simeq \varepsilon \mathcal{A}_{\mathcal{C}}^{X,A}[x] : A[C/X] \)

Proof. We let \( \mathcal{A}_{\mathcal{C}}^{X,A} = \lambda y_1 \ldots y_k. \mathcal{I}_{\mathcal{C}}^{X}[Z(A_1^{\mathcal{C}}(E_1^{\mathcal{C}})[y_1]) \ldots (A_k^{\mathcal{C}}(E_k^{\mathcal{C}})[y_k])] \), where \( Z \) is the rightmost variable of \( C \). We can compute then

\[
x \simeq_{\eta} \lambda y_1 \ldots y_k. \mathcal{I}_{\mathcal{C}}^{X}[E_1^{\mathcal{C}}[xCy_1 \ldots y_k]] \simeq_{\varepsilon} \lambda y_1 \ldots y_k. \mathcal{I}_{\mathcal{C}}^{X}[xZ(A_1^{\mathcal{C}}(E_1^{\mathcal{C}})[y_1]) \ldots (A_k^{\mathcal{C}}(E_k^{\mathcal{C}})[y_k])]
\]

where the rightmost term is exactly \( \mathcal{A}_{\mathcal{C}}^{X,A}[x] \). \( \square \)

By permuting, in a term \( t \) typable in \( \Lambda_{2\text{poly}} \), each subterm of the form \( uB \), with \( u \) of type \( \forall X.A \), into \( \mathcal{A}_{\mathcal{C}}^{X,A}[u] \), we obtain the following:

**Theorem 5.2** (atomization). If \( \Gamma \vdash_{\Lambda_{2\text{poly}}} t : A \), then there exists a term \( t^1 \), called the atomization of \( t \), such that \( \Gamma \vdash_{\Lambda_2} t^1 : A \) and \( \Gamma \vdash_{\Lambda_{2\text{poly}}} t \simeq \varepsilon t^1 : A \).

Let an instance of the \( \varepsilon \)-rule permuting context \( \mathcal{C} \) be called *atomic* if \( \mathcal{C} : \Gamma \vdash X \Rightarrow (\Gamma' \vdash Y) \) for two variables \( X, Y \). We let \( \simeq_{\varepsilon} \) be the smallest theory generated by \( \beta, \eta \)-rules and atomic \( \varepsilon \)-rules.
Proposition 5.3. $\Gamma \vdash \Lambda_{\delta_{n}} t \simeq_{t} u : A$ iff $\Gamma \vdash \Lambda_{a} t^{s} \simeq_{s} u^{s} : A$.

Proof. The $\Leftarrow$-direction is obvious. For the $\Rightarrow$-direction, it suffices to check that the $\varepsilon$-rules commute with atomization: if $t = C[v \in \{ \ldots , w \}] \simeq_{\varepsilon} vDA_{1}(C)[w_{1}] \ldots A_{k}(C)[w_{k}] = u$, where

\begin{align*}
C : (\Delta \vdash \Lambda_{\delta_{n}} C) & \Rightarrow (\Sigma \vdash \Lambda_{\delta_{n}} D),
\end{align*}

then $t^{s} = C^{s} \circ \text{In}_{C}^{s}[v^{s} : \text{DA}_{1}(C)[w_{1}]^{s} \ldots A_{k}^{s}(C)[w_{k}]^{s}] \simeq_{\eta}
\text{In}_{D}^{s} \circ \text{El}_{D}^{s} \circ C^{s} \circ \text{In}_{C}^{s}[v^{s} : \text{DA}_{1}(C)[w_{1}]^{s} \ldots A_{k}^{s}(C)[w_{k}]^{s}]
\simeq_{\varepsilon} \text{In}_{D}^{s} \circ \text{El}_{D}^{s} \circ C^{s} \circ \text{In}_{C}^{s}[v^{s} : \text{DA}_{1}(C)[w_{1}]^{s} \ldots A_{k}^{s}(C)[w_{k}]^{s}]
\simeq_{\varepsilon} \text{In}_{D}^{s} \circ \text{El}_{D}^{s} \circ C^{s} \circ \text{In}_{C}^{s}[v^{s} : \text{DA}_{1}(C)[w_{1}]^{s} \ldots A_{k}^{s}(C)[w_{k}]^{s}]
= u^{s}.$

From Theorem 5.2 we obtain a proof of Theorem 4.3.

Proof of Theorem 4.3. The fundamental remark is that a term typable in $\Lambda_{\delta_{n}}$ is always well-fibered. From Theorem 5.2, $t$ is $\varepsilon$-equivalent to its atomization $t^{s}$, and if we let $t^{\hat{s}}$ be the $\beta\eta$-normal form of $t^{s}$, we deduce from Lemma 4.2 that $\Gamma \vdash \Lambda_{\delta_{n}} t^{\hat{s}} : A$.

Remark 5.1 (instantiation overflow). The atomization property of $\Lambda_{\delta_{n}}$ is related to the instantiation overflow property in $\Pi$. There is a variant of the RP-translation, that we call the FF-translation, is defined, whose target system is $\Lambda_{a}$. The fundamental remark leading to the FF-translation is that when a type $\forall \Pi.A$ translates a sum or a product type, one can construct, for all $C$, terms $t^{s}_{C}$ for $\forall \Pi.A \rightarrow A[\Pi/X]$ in $\Lambda_{a}$.

Our analysis shows that this property extends to all types $\forall \Pi.f(x).A(x)$. Moreover, since the terms $t^{s}_{A}$ are easily seen to be $\beta\eta$-equivalent to $x.\text{At}^{\pi}(x).A(x)$, the mapping $t \mapsto t^{s}$ of Theorem 5.2 is $\beta\eta$-equivalent to the FF-translation. From Theorem 5.2, it thus follows that the FF-translation and the RP-translation yield $\varepsilon$-equivalent terms.

6 The Yoneda fragment

We turn now to type isomorphisms and we consider the problem of finite System F types.

Definition 6.1. $A \in T(\Lambda) \text{ is a finite type if for some closed } A' \in T(\Lambda), A \equiv_{\text{fin}} A' \text{ in } \Lambda_{\text{fin}}.$

In order to investigate finite types, we generalize the isomorphism $A \equiv_{t} \forall \Pi.(A \rightarrow X) \rightarrow X$ of Lemma 3.4 to a more general Yoneda Schema:

\begin{align*}
F[\Pi/X] & \equiv \forall \Pi.(A \rightarrow X) \rightarrow F & (X \neq \text{FV}(A), F \in \text{P}X)
\end{align*}

(YS)

By orienting it from right to left, (YS) yields a type rewrite rule which eliminates a second order quantifier.

We now introduce a fragment $\Lambda_{Y\text{on}}$, larger than $\Lambda_{\text{pol}}$, and show that all types in $\Lambda_{Y\text{on}}$ can be rewritten into $\Lambda$-types by a generalization of the schema (YS). The fragment $\Lambda_{Y\text{on}}$ is obtained by restricting the types of the form $\forall \Pi.A$ to those in which $A$ is X-Yoneda:

Definition 6.2 (X-Yoneda types). For any type variable $X$, we let $\text{Yon}_{x}$ be the set of $X$-Yoneda type, inductively defined as follows:

- if $A \in \text{P}X$, then $A \in \text{Yon}_{x}$;
- if $B \in \text{Yon}_{x}$ and $A \in \text{SP}_{X}$, then $A \rightarrow B \in \text{Yon}_{x}$.
- if $A \in \text{Yon}_{x} \cap \text{Yon}_{y}$ and $X \neq Y$, then $\forall X.A \in \text{Yon}_{x}$.

The following Lemma provides a “canonical form” for X-Yoneda types.

Lemma 6.1. For any $A \in \text{Yon}_{x}$ there is an arrow $\vec{J} \vdash \text{I}$ between finite sets, a $\text{I}$-indexed family of SP$_{X}$-types $(A_{i})_{i \in \text{I}}$ and $F \in \text{P}X$ such that $A \equiv_{t} \forall Y.\text{EXP}_{I_{i \in I}}^{F}(A_{f(j)}) = \forall Y.A_{1} \rightarrow \cdots \rightarrow A_{k} \rightarrow F$.
The main difference between a type $\Lambda_{\text{Yon}}$ and a type of the form $\Xi_{f,g}(A_1)$ is the following: while the rightmost path of the latter (see as a tree) always leads to $X$, the rightmost path of the former may lead to any type $F \in P_X$.

**Definition 6.3** (System $\Lambda_{\text{Yon}}$). The set $\text{Yon} \subseteq T(\Lambda)$ of Yoneda types is defined inductively by (1) $\forall X \in \text{Yon}$, (2) if $A, B \in \text{Yon}$, then $A \rightarrow B \in \text{Yon}$ and (3) if $A \in \text{Yon} \cap \text{Yon}_X$, then $\forall X . A \in \text{Yon}$.

We let $\Lambda_{\text{Yon}}$ be the fragment of $\Lambda$ with types $\text{Yon}$.

By Lemma 6.1 we can consider in $\Lambda_{\text{Yon}}$ a more general class of functors:

**Proposition 6.2** (extended finite polynomial functor). Let $I \xrightarrow{L} \Lambda - \beta, \eta$ be an ordered f.p.f. Then for all fresh $X \in V$ and $F \in P_X$, there exists a functor $\Xi_{f,g} : \mathbb{C}_\beta(\Lambda_{\text{Yon}})^T \rightarrow \mathbb{C}_\beta(\Lambda_{\text{Yon}})$, given by $\Xi_{f,g}(A_1) := \forall X . \text{EXP}_{k \leq \beta}^{F^X}(\text{EXP}_{j \leq \beta}(\text{EXP}^X_{i \leq \beta}(A_{f,(j)})))$ where for all $i \in B$, $f_i : g^{-1}(i) \rightarrow I$ is the restriction of $f$ to $g^{-1}(i)$.

**Remark 6.2** (existential types in $\Lambda_{\text{Yon}}$). Given a type variable $X$, let us call a finite family of $\Lambda_{\text{Yon}}$ types $A_1, \ldots, A_k$ a Yoneda family in $X$ when for some fresh variable $Y$, $A_1 \rightarrow \cdots \rightarrow A_k \rightarrow Y \in \text{Yon}_X$.

If $A_1, \ldots, A_k$ is a Yoneda family in $X$, then the existential type $\exists X . A_1 \times \cdots \times A_k$ can be expressed by a $\Lambda_{\text{Yon}}$-type through the usual System F coding as $\forall Y . (\forall X . A_1 \rightarrow \cdots \rightarrow A_k \rightarrow Y) \rightarrow Y$.

**Example of such types are** $\exists X . A \rightarrow X$, $\exists X . X \rightarrow A$ and $\exists X . (A \rightarrow X) \times (X \rightarrow B)$, when $X \notin FV(A)$.

The $\varepsilon$-theory extends in a straightforward way to $\Lambda_{\text{Yon}}$.

**Definition 6.4** ($\varepsilon$-theory). We let $\approx \varepsilon$ be the smallest theory over $\Lambda_{\text{Yon}}$ containing $\beta$ and $\eta$-rules and the $\varepsilon$-rule in Fig. 7, where $A = \forall \nu\varepsilon\text{EXP}_{j \leq \beta}^{F^X}(\text{EXP}^X_{i \leq \beta}(A_{f,(j)})) \in \text{Yon}_X$ (see Lemma 6.1) and $f(J) = \{i_1 < \cdots < i_k\}$.

The isomorphism $(YS)$ can be proved similarly to Lemma 3.4.

**Lemma 6.3.** For all $A \in \text{Yon}$, if $X \notin FV(A)$, $F \in P_X \cap \text{Yon}$, $F[A/X] \equiv \varepsilon \forall X . (A \rightarrow X) \rightarrow F$.

By Lemma 6.3, one can lift Proposition 3.5 to extended polynomial functors:

**Proposition 6.4.** For all $F \in P_X$, $\Xi_{f,g}(A_1) \equiv \varepsilon F[\Xi_{f,g}(A_1)/X]$.

The basic idea to define Yoneda reduction $\sim^*$ is to exploit the isomorphism above. In Fig. 8, we show some examples of how Yoneda types can be reduced to $\Lambda p$-type$^3$, where we recall that $\exists X . (A \rightarrow X) \times (X \rightarrow B) = \forall Y . (\forall X . (A \rightarrow X) \rightarrow (X \rightarrow B) \rightarrow Y) \rightarrow Y$.

For reasons of space, we postpone the technical details of the Yoneda reduction $\sim^*$ to Appendix D. We only state here our convergence result:

**Theorem 6.5.** For all $\Lambda_{\text{Yon}}$-type $A$, there exists a $\Lambda p$-type $A^\circ$ such that $A \sim^* A^\circ$.

**Corollary 6.1.** Any closed $\Lambda_{\text{Yon}}$-type is a finite type.

---

$^3$We indicate by $\sim^{X,F}$ a reduction eliminating quantifier $\forall X$ with positive functor $F \in P_X$. 

13
\[
\forall Y. \ (A \to Y) \to \forall X. \ (B \to X) \to (C \to X) \to Y \sim^{X,Y} \forall Y. \ (A \to Y) \to Y \sim^{Y,Y} A
\]

\[
\exists X. \ (A \to X) \times (X \to B) \sim^{X, (X \to B)} \forall Y. \ ((A \to B) \to Y) \to Y \sim^{Y,Y} A \to B
\]

Figure 8: Examples of Yoneda reduction.

7 The decidability of polynomial and Yoneda types

To establish the decidability of equivalence in \(\Lambda_{2\text{Poly}}\) and \(\Lambda_{2\text{Yon}}\), we prove that the syntactic categories generated by these fragments under the \(\varepsilon\)-theory are equivalent to the free bicartesian closed category \(\mathbb{B} = C^0_{\beta\eta}(\Lambda p)\). Decidability follows from Theorem 2.1. We develop our argument for \(\Lambda_{2\text{Yon}}\), but a similar argument works for \(\Lambda_{2\text{Poly}}\).

Theorem 6.3 yields a surjective map \([ \ ]^p : \text{Yon} \to T(\Lambda p)\) such that \(A \equiv_{\varepsilon} A^p\). The idea is to extend this map into an equivalence of categories, thanks to the lemma below:

**Lemma 7.1.** Let \(\mathbb{C}, \mathbb{D}\) be full subcategories of a category \(\mathbb{E}\). Let \(f : \text{Ob}(\mathbb{C}) \to \text{Ob}(\mathbb{D})\) be surjective and such that any object \(a\) of \(\mathbb{C}\) is isomorphic to \(f(a)\) in \(\mathbb{E}\). Then \(f\) extends to an equivalence of categories \(F : \mathbb{C} \to \mathbb{D}\).

**Proof.** Let \(u_a : a \to f(a)\) be the isomorphism between \(a\) and \(f(a)\). We let \(F(a) = f(a)\) and \(F(g : a \to b) = u_a \circ g \circ u_b^{-1}\). \(F\) is clearly faithful and surjective. It is also full since any \(h : F(a) \to F(b)\) is equal to \(F(u_b^{-1} \circ h \circ u_a)\).

We let \(p\text{Yon}\) be the set of types obtained by extending \(\text{Yon}\) with finite products and coproducts, and \(\Lambda_{2p\text{Yon}}\) be the fragment of \(\Lambda_{2p}\) generated by \(p\text{Yon}\). We wish to show that \(C^0_{\beta\eta}(\Lambda p)\) and \(C^0_{\beta\eta}(\Lambda_{2p\text{Yon}})\) are full subcategories of \(C^0_{\beta\eta}(\Lambda_{2p})\). In the case of \(C^0_{\beta\eta}(\Lambda p)\), fullness can be deduced from the existence of normal forms with respect to permutative conversions. In fact a term with a type in \(\Lambda p\) and in permutative normal form enjoys the subformula property, hence it is typable in \(\Lambda p\). The existence of permutative normal forms is well-known in the case of \(\Lambda p\) since 30, and was extended to \(\Lambda_{2p}\) in 31. This implies the following:

**Proposition 7.2.** For all \(\Gamma, A \in T(\Lambda p)\), if \(\Gamma \vdash_{\Lambda_{2p}} t : A\), then there exists \(t^{\text{perm}} \equiv_{\beta\eta} t\) such that \(\Gamma \vdash_{\Lambda p} t^{\text{perm}} : A\).

In the case of \(C^0_{\beta\eta}(\Lambda_{2\text{Yon}})\), fullness is deduced from the remark that if \(\Gamma, A \in \text{Yon}\) and \(\Gamma \vdash_{\Lambda_{2\text{poly}}} t : A\), then \(\Gamma \vdash_{\Lambda_{2}} t^* : A\), and from the following lemma, proved in Appendix D.

**Lemma 7.3.** For all \(\Gamma, A \in \text{Yon}\), if \(\Gamma \vdash_{\Lambda_{2\text{poly}}} t : A\), then \(\Gamma \vdash_{\Lambda_{2\text{Yon}}} t \equiv_{\varepsilon} t^* : A\).

We can now apply Lemma 7.1 to obtain:

**Theorem 7.4.** \(C^0_{\beta\eta}(\Lambda p)\) and \(C^0_{\beta\eta}(\Lambda_{2\text{Yon}})\) are equivalent categories.

**Remark 7.1.** In more concrete terms, Theorem 7.4 yields an algorithm to translate terms \(t, u\) such that \(\Gamma \vdash_{\Lambda_{2\text{Yon}}} t, u : A\) into terms \(\tilde{t}^p, \tilde{u}^p\) such that \(\Gamma \vdash_{\Lambda p} \tilde{t}^p, \tilde{u}^p : A^p\) and \(t \equiv_{\varepsilon} u\) holds iff \(\tilde{t}^p \equiv_{\beta\eta} \tilde{u}^p\) (apply the isomorphisms \(B \equiv_{\varepsilon} B^p\) and compute the normal forms of \(\tilde{t}^p\) and \(\tilde{u}^p\) for permutative conversions).

From Theorem 2.1 we can finally conclude:

**Theorem 7.5.**

i. Type inhabitation in \(\Lambda_{2\text{Yon}}\) (resp. \(\Lambda_{2\text{Poly}}\)) is decidable.

ii. The \(\varepsilon\)-theory for \(\Lambda_{2\text{Yon}}\) (resp. \(\Lambda_{2\text{Poly}}\)) is decidable.

iii. The \(\varepsilon\)-theory for \(\Lambda_{2\text{Yon}}\) (resp. \(\Lambda_{2\text{Poly}}\)) coincides with \(\simeq_{\text{ctx}}\).
8 Conclusion and future work

We described two fragments of System F which correspond, up to contextual equivalence, to the free bicartesian closed category. These fragments arise from the second order translation of finite polynomial functors and its connection with the Yoneda embedding. We equipped both fragments with a syntactic equivalence, the \( \varepsilon \)-theory, and we used it to establish two properties of the polymorphic terms in \( \Lambda_2 \text{poly} \) (naturality and atomization). Moreover, we introduced a type-rewriting relation and we used it to show that the types of \( \Lambda_2 \text{Yon} \) are isomorphic, modulo contextual equivalence, to propositional types.

Future work

We indicate some directions for further research.

Syntactic analysis of \( \varepsilon \)-equivalence Our rely on the close connection between the \( \varepsilon \)-theory and the \( \beta\eta \)-theory of \( \Lambda_\rho \), and of some extensively investigated properties of the latter (see [13, 1, 21, 33]). The decidability of \( \varepsilon \)-equivalence is obtained here by a translation into \( \Lambda_\rho \), i.e. without directly providing a decision algorithm for \( \varepsilon \)-equivalence within \( \Lambda_2 \text{poly} \) or \( \Lambda_2 \text{Yon} \). The rewriting techniques of [21] as well as the focusing techniques of [33] might provide interesting tools to investigate the \( \varepsilon \)-theory in a less indirect manner.

Finite types through recursive types Our approach to type isomorphisms can be naturally extended by considering a generalized Yoneda schema [38] involving recursive types:

\[
F[\mu X.T/X] \equiv \forall X.(T \to X) \to F \quad (T, F \in \mathcal{P}_X)
\]

As \( \mu X.T \) can be replaced by its System F coding \( \forall X.(T \to X) \to X \), any type which reduces to a closed \( \Lambda_\rho \)-type by (GYS) is a finite type. This shows in particular that the rewriting defined for \( \Lambda_2 \text{Yon} \) does not capture all finite types of System F: the type \( \exists X.(X \to X) \) (in its System F coding) is not a \( \Lambda_2 \text{Yon} \)-type and does not converge onto a closed \( \Lambda_\rho \)-type using our Yoneda reduction, but it can be reduced to 1 by (GYS). We could not find so far any finite type which does not reduce to a \( \Lambda_\rho \)-type under this stronger schema.

Since program equivalence in presence of recursive types is undecidable in general (see [2]), a more viable option could be to restrict the attention to isomorphisms involving recursive polynomial types, which have a decidable type isomorphism [11].

Generalized connectives From a proof-theoretic perspective, finite polynomial functors correspond to generalized connectives in the sense of [31, 34, 3]. In particular, the functors \( \Phi_{f,g} \) and \( \Upsilon_{f,g} \) can be seen as encoding introduction and elimination rules for generalized connectives. For instance, let \( \triangledown \) be the ternary connective governed by the rules below:

\[
\begin{array}{c}
\frac{A_1}{(A_1, A_2, A_3)} \triangledown \frac{A_2}{(A_1, A_2, A_3)} \triangledown \frac{A_3}{(A_1, A_2, A_3)} \triangledown \frac{A_1, A_2}{(A_1, A_2, A_3)} \triangledown \frac{A_1, A_3}{(A_1, A_2, A_3)} \triangledown X \\
\end{array}
\]

The introduction and elimination rules can then be seen as encoded, respectively, by the functors \( \Phi_{f,g} : (A_i)_{i \in \mathbf{3}} \to (A_1 \times A_2) + (A_1 \times A_3) \) and \( \Upsilon_{f,g} : (A_i)_{i \in \mathbf{3}} \to \forall X.(A_1 \to A_2 \to X) \to (A_1 \to A_3 \to X) \to X \). We would like to investigate whether the isomorphism \( \Phi_{f,g} \equiv \Upsilon_{f,g} \) can be used to provide a formal account of the \textit{inversion principles} discussed in the proof-theoretic literature (see [23]).
References

[1] Thorsten Altenkirch, Peter Dybjer, Martin Hofmann, and Philip J. Scott. Normalization by evaluation for typed lambda calculus with coproducts. In 16th Symposium on Logic in Computer Science (LICS), 2001.

[2] Steve Awodey, Jonas Frey, and Sam Speight. Impredicative encodings of (higher) inductive types. https://arxiv.org/abs/1802.02820, 2018.

[3] Roland C. Backhouse. On the meaning and construction of the rules in Martin-Löf’s theory of types. In Computing Science Notes CS 8606. Department of Mathematics and Computing Science, University of Groningen, 1986.

[4] E.S. Bainbridge, Peter J. Freyd, Andre Scedrov, and Philip J. Scott. Functorial polymorphism. Theoretical Computer Science, 70:35–64, 1990.

[5] Henning Basold and Helle Hvid Hansen. Well-definedness and observational equivalence for inductive-coinductive programs. Journal of Logic and Computation, exw091, 2016.

[6] Jean-Philippe Bernardy, Patrik Jansson, and Koen Claessen. Testing Polymorphic Properties. In ESOP 2010: Programming Languages and Systems, volume 6012 of Lecture Notes in Computer Science, pages 125–144. Springer Berlin Heidelberg, 2010.

[7] Kim B. Bruce, Roberto Di Cosmo, and Giuseppe Longo. Provable isomorphisms of types. Mathematical Structures in Computer Science, 1:1–20, 1991.

[8] Joachim de Lataillade. Dinatural terms in System F. In Proceedings of the Twenty-Fourth Annual IEEE Symposium on Logic in Computer Science (LICS 2009), pages 267–276, Los Angeles, California, USA, 2009. IEEE Computer Society Press.

[9] Derek Dreyer, Amal Ahmed, and Lars Birkedal. Logical step-indexed logical relations. Logical Methods in Computer Science, 7:2–16, 2011.

[10] Fernando Ferreira and Gilda Ferreira. Atomic polymorphism. Journal of Symbolic Logic, 78(1):260–274, 2013.

[11] Marcelo Fiore. Isomorphisms of generic recursive polynomial types. POPL 2004, Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 77–88, 2004.

[12] Nicola Gambino and Joachim Kock. Polynomial functors and polynomial monads. Mathematical Structures in Computer Science, 154(1):153–192, 2013.

[13] Neil Ghani. $\beta\eta$-equality for coproducts. In TLCA ’95, International Conference on Typed Lambda Calculi and Applications, volume 902 of Lecture Notes in Computer Science, pages 171–185. Springer-Verlag, 1995.

[14] Jean-Yves Girard. The System $F$ of variable types, fifteen years later. Theoretical Computer Science, 45(2):159–192, 1986.

[15] Jean-Yves Girard, Yves Lafont, and Paul Taylor. Proofs and Types, volume 7 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1989.

[16] A. D. Gordon. Operational equivalences for untyped and polymorphic object calculi. In A.D. Gordon and A.M. Pitts, editors, Higher Order Operational Techniques in Semantics. Cambridge University Press, 1998.
[17] Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, 2016.

[18] Rye Hasegawa. Categorical data types in parametric polymorphism. *Mathematical Structures in Computer Science*, 4(1):71–109, 2009.

[19] Martin Hyland, Edmund P. Robinson, and Giuseppe Rosolini. Algebraic types in PER models. In M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Proceedings of the 5th International Conference on Mathematical Foundations of Programming Semantics*, volume 442 of *Lecture Notes in Computer Science*, pages 333–350, New York, 1989. Springer.

[20] Bart Jacobs. *Categorical logic and type theory*, volume 141 of *Studies in Logic and Foundations of Mathematics*. Elsevier, 1999.

[21] Sam Lindley. Extensional rewriting with sums. In *Typed Lambda Calculi and Applications, TLCA 2007*, volume 4583 of *Lecture Notes in Computer Science*, pages 255–271. Springer Berlin Heidelberg, 2007.

[22] Saunders MacLane. *Categories for the working mathematicians*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1978.

[23] Enrico Moriconi and Laura Tesconi. On inversion principles. *History and Philosophy of Logic*, 29(2):103–113, 2008.

[24] Paolo Pistone. On dinaturality, typability and $\beta\eta$-stable models. In Schloos Dagstul-Leibniz-Zentrum fuer Informatik, editor, *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, volume 84 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:17, Dagstuhl, Germany, 2017.

[25] Paolo Pistone. Proof nets and the instantiation overflow property. https://arxiv.org/abs/1803.09297, 2018.

[26] Paolo Pistone. Proof nets, coends and the Yoneda isomorphism. In Thomas Ehrhard, Maribel Fernández, Valeria de Paiva, and Lorenzo Tortora de Falco, editors, *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications (Linearity-TLLA 2018)*, volume 292 of *EPTCS*, pages 148–167, 2019.

[27] A.M. Pitts. Typed operational reasoning. In B. C. Pierce, editor, *Advanced Topics in Types and Programming Languages*. MIT Press, 2005.

[28] Gordon Plotkin and Martin Abadi. A logic for parametric polymorphism. In *TLCA ’93, International Conference on Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 361–375. Springer Berlin Heidelberg, 1993.

[29] Dag Prawitz. *Natural deduction, a proof-theoretical study*. Almqvist & Wiskell, 1965.

[30] Dag Prawitz. Ideas and results in proof theory. In J.E. Fenstad, editor, *Proceedings of the 2nd Scandinavian Logic Symposium (Oslo)*, volume 63 of *Studies in logic and foundations of mathematics*. North-Holland, 1971.

[31] Dag Prawitz. Proofs and the meaning and completeness of the logical constants. In Jaakko Hintikka, Ilkka Niiniluoto, and Esa Saarinen, editors, *Essays on Mathematical and Philosophical Logic: Proceedings of the Fourth Scandinavian Logic Symposium and the First Soviet-Finnish Logic Conference, Jyväskylä, Finland, June 29–July 6, 1976*, pages 25–40. Kluwer, Dordrecht, 1979.
A The isomorphism between $A$ and $A^*$

We define for all $\Lambda 2p$-type $A$, two contexts $C_A : A \vdash_0 \Lambda 2p A^*$ and $\overline{C}_A : A^* \vdash_0 \Lambda 2p A$ as follows:

- for $A = X$, $C_A = \overline{C}_A = [ ]$;
- for $A = B \rightarrow C$, $C_A = \lambda y. C_C \circ ([ ][C_B[y]])$, $\overline{C}_A = \lambda y. \overline{C}_C \circ ([ ][C_B[y]])$,
- for $A = \forall Y.B$, $C_A = \Lambda Y.C_B[[ ]Y]$ and $\overline{C}_A = \Lambda Y.\overline{C}_B[[ ]Y]$;
- for $A = B \times C$, $C_A = \Lambda Y.\lambda y.y(C_B[\pi_1[ ]]) (C_C[\pi_2[ ]])$, $\overline{C}_A = x(B \times C) \lambda y.z. (C_B[y], C_C[z])$;
- for $A = 1$, $C_A = \Lambda Y.\lambda y.y$, $\overline{C}_A = *$;
- for $A = B + C$, $C_A = \Lambda Y.\lambda ab.\delta_X([ ], y.a(C_B[y]), y.b(C_C[y]))$, and $\overline{C}_A = [ (B + C) \lambda y.t_1(\overline{C}_B[y]) \lambda y.t_2(\overline{C}_C[y]) ]$;
- for $A = 0$, $C_A = \xi_{Y,X}[ ]$ and $\overline{C}_A = [ ]0$.

To show that the pair $(C_A, \overline{C}_A)$ is an $\varepsilon$-isomorphism we can argue by induction on $A$. We here only consider the most significative case, namely the one of the sum: if $A = B + C$, then we can compute

$$C_A \circ \overline{C}_A = \Lambda Y.\lambda ab.D \circ \overline{C}_A = \Lambda Y.\lambda y.D[[ ][(B + C) \lambda y.t_1(\overline{C}_B[y]) \lambda y.t_2(\overline{C}_C[y])]]$$

where $D = \delta_Y([ ], y.a(C_B[y]), y.b(C_C[y]))$, and

$$\overline{C}_A \circ C_A = C_A(B + C) \lambda y.t_1(\overline{C}_B[y]) \lambda y.t_2(\overline{C}_C[y])$$

$$\approx_\beta \delta_{B+C}([ ], y.t_1(\overline{C}_B[c] \circ C_B[y]), y.t_2(\overline{C}_C[c] \circ C_C[y]))$$

$$\approx_\varepsilon^{[I,H]} \delta_{B+C}([ ], y.t_1 y, y.t_2 y) \approx_\eta [ ]$$
B Proof of Lemma 4.2

Proof. Claim i, can be easily established by induction on a typing derivation of $\Lambda_{2p}^\alpha$. To show ii, we prove the following stronger claim: let $t$ be $\beta$-normal, $\eta$-long, well-fibered and such that for all extraction $uB$ occurring in $t$, $B \in SP_\alpha$. Then, if $\Gamma, A \in SP_\alpha$, then for all renaming $\theta$, $\Gamma \vdash t : A$ implies $\Gamma \vdash t^\theta : A$. From this claim, ii follows by taking $\alpha = \emptyset$.

We argue by induction on $t$. We will use the fact that $A \theta = B \theta$ implies $A = B$ (as $\theta$ is injective), that $(A \rightarrow B) \theta = A \theta \rightarrow B \theta$, $(A[B/X]) \theta = A \theta[B/X]$, and that $(\forall X.A) \theta = \forall X.A \theta_X$, where $\theta_X$ differs from $\theta$ in that $\theta(X) = X$.

- if $t = x$: from $\Gamma \vdash x : A \vdash_{\Lambda_{2p}^\alpha} x : A$ and $\Gamma, A \in SP_\alpha$ we deduce $\Gamma, x : A \vdash_{\Lambda_{2p}^\alpha} x : A$ (as $x \theta = x$);
- if $t = \lambda y.t'$, then $A = B \rightarrow C$, where $C \in SP_\alpha$ and $\alpha \notin B$, and we have $\Gamma, y : B \vdash_{\Lambda_{2p}^\alpha} t^\theta : C \theta$. Then $\Gamma, y : B \in SP_\alpha$ hence by the induction hypothesis $\Gamma \vdash_{\Lambda_{2p}^\alpha} t : C$, so we can conclude $\Gamma \vdash_{\Lambda_{2p}^\alpha} t : A$ (as $t \theta = \lambda y.t' \theta$).
- if $t = x_1 \cdots x_n$, then we must have $\Gamma \vdash t_i \vdash_{\Lambda_{2p}^\alpha} C_i \theta$. Moreover, as $\Gamma$ contains $x : C_1 \rightarrow \cdots \rightarrow C_n \rightarrow A \in SP_\alpha$, $\alpha \notin C_i$, for $i = 1, \ldots, n$, whence $C_i \in SP_\alpha$. Hence by the induction hypothesis we have $\Gamma \vdash t_i : C_i$ and we can conclude $\Gamma \vdash_{\Lambda_{2p}^\alpha} t : A$ (as $t \theta = x(t_1 \theta) \cdots (t_n \theta)$).
- if $t = \Lambda Y.t'$, then $A = \forall Y.A_1 \rightarrow \cdots \rightarrow A_k \rightarrow Y$ and, since $t$ is $\eta$-long, $t' = \lambda x_1 \ldots x_k t''$, hence we have $\Gamma, x_1 : A_1 \theta, \ldots, x_k : A_k \theta \vdash_{\Lambda_{2p}^\alpha} t''_\theta : Y$. Now, since $A \in SP_\alpha$, we deduce $\alpha \notin A_i$; for $i = 1, \ldots, k$, and as $A_i \in SP_\alpha$, we deduce $A_i \in SP_\alpha + Y$. Since moreover $Y \notin FV(\Gamma)$, from $\Gamma \in SP_\alpha$ we deduce $\Gamma \in SP_\alpha + Y$. In definitive we have that $\Gamma, x_1 : A_1, \ldots, x_k : A_k \vdash_{\Lambda_{2p}^\alpha} t'' : Y$ is a $SP_\alpha + Y$-judgement, and since $t''$ is well-fibered, for all extraction $x B$ occurring in $t''$, $B \in SP_\alpha + Y$. We can thus apply the induction hypothesis, and we have $\Gamma, x_1 : A_1, \ldots, x_k : A_k \vdash_{\Lambda_{2p}^\alpha} t'' : Y$, and we can conclude $\Gamma \vdash_{\Lambda_{2p}^\alpha} t : A$ (as $t \theta = \Lambda Y.t' \theta$).
- if $t = x B t_1 \cdots t_n$, then $\Gamma$ contains $x : \forall X.A_1 \rightarrow \cdots \rightarrow A_k \rightarrow X$, where $\alpha \notin A_i$, and since $t$ is $\eta$-long there is $k \leq n$ such that $B = B_1 \rightarrow \cdots \rightarrow B_{n-k} \rightarrow Z$, $\Gamma \vdash_{\Lambda_{2p}^\alpha} t_j \vdash : A_j[B/X]$, for $i \leq k$ and $1 \leq j \leq n - k$. Since $B \in SP_\alpha$ and $A \in SP_\alpha$, we have then $A_j[B/X] \in SP_\alpha$ and $B_j \in SP_\alpha$, hence $\Gamma \vdash t_i : A_j[B/X]$ is a $SP_\alpha$-judgement, so by the induction hypothesis we deduce $\Gamma \vdash_{\Lambda_{2p}^\alpha} t_i : A_j[B/X]$, and $\Gamma \vdash t_j : B_j$ is a $SP_\alpha$-judgement, so by the induction hypothesis we deduce $\Gamma \vdash_{\Lambda_{2p}^\alpha} t_j : B_j$. We can thus conclude $\Gamma \vdash_{\Lambda_{2p}^\alpha} t : A$ by the rules of $\Lambda_{2p}^\alpha$.

\[ \square \]

C The Yoneda reduction

We wish to introduce a rewrite relation over $\Lambda_{2p}$-types such that whenever $A \rightsquigarrow B$, $A$ is $\varepsilon$-isomorphic to $B$, and moreover, when $A$ is a Yoneda type, the rewriting converges onto a propositional type. A natural idea is to exploit the isomorphism of Lemma 6.3 to define a rewrite rule of the form
\[
\Upsilon_{f,g}^F(A_i) \rightsquigarrow F[\Psi_{f,g}(A_i)/X]
\]
(\text{red})

However, this rule is not strong enough to prove convergence onto a propositional type (see Remark C.2 below). To describe a stronger rule we need to introduce some new technical notions. For all natural number $n$, we let $\lfloor n \rfloor = \{1, \ldots, n\}$.
We need two lemmas: C.2 and Lemma 6.4, it can be seen that Yoneda types. We need two lemmas: C.2 and Lemma 6.4, it can be seen that Yoneda types.

**Definition C.2**

- a polynomial type is a quasi-polynomial type as in Def. C.2, then by exploiting the β-isomorphism to the type A'i → ··· → Ai→ P(C, i) can be simulated by a finite number of β-isomorphisms plus the rule (red). However, the isomorphism A → B → C implies A ∼ B → A → C does not preserve the Yoneda restriction: for instance, while (Y → X) → (X → Y) → Y ∈ YonX, (X → Y) → (Y → X) → Y ∉ YonX. For this reason, for ∼ to be a relation over pYon, we must reduce quasi-polynomial types directly through (red*).

By inspecting the type reduction rules, and by arguing as in the case of (red) through Remark C.2 and Lemma 6.4, it can be seen that A ∼ B implies A ∼ B. We need two lemmas:

**Lemma C.1**. For all A2p-types A, B such that A ∼ B, and for all type variable X,

i. $FV(B) ⊆ FV(A)$;
ii. if \( A \in \text{SP}_X \), \( B \in \text{SP}_X \);

iii. if \( A \in \text{P}_X \) (resp. \( A \in \text{N}_X \)), \( B \in \text{P}_X \) (resp. \( B \in \text{N}_X \)).

**Lemma C.2.** Let \( Y \neq X \in V \). If \( \forall X.A \in \text{Yon}_Y \) and \( \forall X.A \rightsquigarrow B \) by an instance of \( \text{red}^* \), then \( B \in \text{Yon}_Y \).

**Proof.** We can suppose \( \forall X.A \) to be quasi-polynomial with index-set \( J \subseteq \{k\} \) and associated polynomial \( \Psi_{f,g}(C_i) \). For any type \( E \), we let \( E^\dagger = E[\Psi_{f,g}(C_i)/X] \). We have then that

\[
A = A_1 \to \cdots \to A_k \to A_{k+1} \to \cdots \to A_{k+n} \to D
\]

where for \( i \in J \), \( A_i \in \text{SSP}_X \), for \( i \in \{k\} - J \), \( X \notin FV(A_i) \), for \( j \in \{n\} \), \( A_{k+j} \in \text{N}_X \), and \( D \) is a base type and

\[
B = A_{i_1} \to \cdots \to A_{i_p} \to A_{j_{k+1}} \to \cdots \to A_{j_{k+n}} \to D^\dagger
\]

where \( \{i_1, \ldots, i_p\} = \{k\} - J \). Since \( A \in \text{Yon}_Y \), it must be \( D \in \text{P}_Y \) and there exists \( b \leq k + n \) such that for \( 1 \leq i \leq b \), \( A_i \in \text{SP}_Y \) and for \( b < i \leq k + n \), \( A_i \in \text{N}_Y \). We must consider then two cases:

1. If \( b > k \) Let \( b = k + b' \), for some \( b' \in \{n\} \). Then for \( i \in J \), \( X \notin FV(A_i) \), whence \( Y \notin FV(\Psi_{f,g}(C_i)) \), for \( i \in \{k\} - J \), \( A_i \in \text{SP}_Y \), for \( 1 \leq j \leq b', A_{k+j} \in \text{SP}_Y \) and for \( b' < j \leq n \), \( A_{k+j} \in \text{N}_Y \). Then \( B \) can be decomposed as follows:

\[
\begin{array}{c}
A_{i_1} \to \cdots \to A_{i_p} \\
\text{sp}_Y \\
A_{j_{k+1}} \to \cdots \to A_{j_{k+b}} \\
\text{sp}_Y \\
A_{j_{k+b'+1}} \to \cdots \to A_{j_{k+n}} \\
\text{sp}_Y \\
\end{array}
\]

and we can conclude \( B \in \text{Yon}_Y \).

2. If \( b < k \) Then for \( i \in J \), \( i \leq b \), \( X \notin FV(A_i) \), for \( i \notin J \), \( b < i \), \( A_i \in \text{N}_Y \), and since \( A_i = X.A \subseteq \exp^{X-j-1} (C_{f(i)}) \), we have \( C_{f(i)} \in \text{P}_Y \), whence \( \Psi_{f,g}(C_i) \in \text{P}_Y \), for all \( j \in g^{-1}(i) \); moreover, for \( i \in \{k\} - J \), \( i \leq b \), \( A_i \in \text{SP}_Y \), for \( i \in \{k\} - J \), \( b < i \), \( A_i \in \text{N}_Y \) and for \( j \in \{n\} \), \( A_{k+j} \in \text{N}_Y \). Then \( B \) can be decomposed as follows:

\[
\begin{array}{c}
A_{i_1} \to \cdots \to A_{i_p} \\
\text{sp}_Y \\
A_{j_{k+1}} \to \cdots \to A_{j_{k+b}} \\
\text{sp}_Y \\
A_{j_{k+b'+1}} \to \cdots \to A_{j_{k+n}} \\
\text{sp}_Y \\
\end{array}
\]

and we can conclude \( B \in \text{Yon}_Y \).

\[\square\]

**Proposition C.3.** If \( A \in \text{pYon} \) and \( A \rightsquigarrow B \), then \( B \in \text{pYon} \).

**Proof.** We claim that if \( A \in \text{Yon}_Y \) and \( A \rightsquigarrow B \), then \( B \in \text{Yon}_Y \). If \( A = \forall X.A \) and \( A \rightsquigarrow B \) by \( \text{red}^* \), then the claim follows from Lemma [C.2]. If \( A \rightsquigarrow B \) by any of the rules in Fig. 9, then the claim can be easily checked.

Now suppose \( A \in \text{pYon} \) and \( A \rightsquigarrow B \). If \( A = \forall X.A' \) is a quasi-polynomial type with associated polynomial \( \Psi_{f,g}(C_i) \) and \( A \rightsquigarrow B \) by \( \text{red}^* \), then for any subtype of \( A' \) of the form \( \forall Y.D \), from \( D \in \text{Yon}_Y \) by Lemma [C.1] we deduce \( D[\Psi_{f,g}(C_i)/X] \in \text{Yon}_Y \), hence \( \forall Y.D[\Psi_{f,g}(C_i)/X] \in \text{pYon} \). We can then conclude that \( B \in \text{pYon} \). For the reduction rules in Fig. 9 the only non trivial case to check is the rule \( \forall X.A \rightsquigarrow \forall X.A' \). In this case, from \( \forall X.A \in \text{pYon} \), we deduce \( A \in \text{Yon}_X \), and by our claim this implies \( A' \in \text{Yon}_X \), hence finally \( \forall X.A \in \text{pYon} \).

We can finally prove Theorem [0.2] that, that any Yoneda type rewrites into a propositional type:
Proof of Theorem 6.5. We argue by induction on $A$. If $A = X$, the claim is obvious. If $A = B \to C$, then by the induction hypothesis $B \sim^* B'$ and $C \sim^* C'$, where $B'$, $C'$ are $\Lambda p$-types, hence $A \sim^* B' \to C'$, which is a $\Lambda p$-type.

If $A = \forall X. B$, then it must be $A = \forall \tilde{X}_1. A_1 \to \forall \tilde{X}_2. A_2 \to \cdots \to \forall \tilde{X}_n. A_n$ where $A_n$ is a quasi-base type. By the induction hypothesis $A$ reduces to $A' = \forall \tilde{X}_1. A'_1 \to \forall \tilde{X}_2. A'_2 \to \cdots \to \forall \tilde{X}_n. A'_n$ where $A'_i \in T(\Lambda p)$. We can then eliminate by a finite number of applications of $\text{red}^*$ all quantifiers $\forall \tilde{X}_1, \ldots, \forall \tilde{X}_n$ starting from the rightmost one (by exploiting Remark C.1), obtaining a propositional type.

---

D Commutation property of the second order translation

In order to show Lemma D.3, we will prove a stronger property expressing the commutation of the second order translation and the isomorphisms $(C_A, \overline{C}_A)$ introduced in Appendix A. We exploit the abbreviations introduced in Section 4.

Proposition D.1. If $\Gamma \vdash_{\Lambda 2\text{prim}} t : A$, then $\Gamma \vdash_{\Lambda 2\text{prim}} C_A[t] \simeq \epsilon t^* \circ_m C\Gamma : A^*$.

Since, when $A \in T(\Lambda 2)$, $C_A[\cdot] \simeq_{\eta} \overline{C}_A[\cdot] \simeq_{\eta} [\cdot]$, it is clear that Lemma D.3 follows from Proposition D.1. To prove Proposition D.1, we first establish two technical lemmas.

Lemma D.2. Let $J \overset{f}{\to} I$ be a diagram in $\text{FinLin}$, $(A_i)_{i \in I}$ be a $I$-indexed set of $SPX$ types and $A = \forall \tilde{Y}. \text{EXP} \overset{f}{\in} J(A_{f(i)}) \in \mathsf{Yn}_X$. Then for all $\Lambda 2p$-type $C$ the equation below holds:

$$x : \forall X. A \vdash_{\Lambda 2\text{prim}} \tilde{Y}. \lambda \tilde{z}. f^x(\overline{C}_C)[x CY (A'_{f(i_1)}(\overline{C}_C)[z_1]) \cdots (A'_{f(i_k)}(\overline{C}_C)[z_k])] \simeq_{\epsilon} x C^* : A[C^*/X]$$

(2)

where $f(J) = \{i_1 < \cdots < i_k\}$.

Proof. From $x CY (A'_{f(i_1)}(\overline{C}_C)[z_1]) \cdots (A'_{f(i_k)}(\overline{C}_C)[z_k]) \simeq_{\epsilon} f^x \overline{C}_C[x C^* z_1 \cdots z_k]$ we deduce $T \simeq_{\epsilon} \tilde{Y}. \lambda \tilde{z}. x C^* z_1 \cdots z_k \simeq_{\eta} x C^*$, where $T$ is the left-hand term in Equation 2.

Lemma D.3. Let $A$ be as in Lemma D.2. Then for all $\Lambda 2p$-type $C$, $C_{\forall X. A} C^* \simeq_{\epsilon} C_A[C/X]$.

Proof. By a simple calculation we can deduce $C_{\forall X. A} C^* \simeq_{\beta} C_A[[ ][C^*]]$ and

$$C_A[C/X] \simeq_{\beta} C_A[\tilde{Y}. \lambda \tilde{z}. f^x(\overline{C}_C)[[ ] CY (A'_{f(i_1)}(\overline{C}_C)[z_1]) \cdots (A'_{f(i_k)}(\overline{C}_C)[z_k])]$$

so we can conclude by Lemma D.2.

Proof of Proposition D.1. By induction on $t$:

- if $t = x$, then $C_A t = t^* \circ_m C_A = C_A$;
- if $t = \lambda y. t'$, then $A = B \to C$ and $t^* = \lambda y. (t')^*$, and by the induction hypothesis $C_C[t'] \simeq_{\epsilon} (t')^* \circ_m C\Gamma \cdot y \cdot B$. So we have $C_A t \simeq_{\beta} \lambda y. C_C[(t')^*] \circ_m C_B[y]/y \simeq_{\epsilon} \lambda y. (t')^* \circ_m C\Gamma \cdot C_B \circ \overline{C}_B[y]/y \simeq_{\epsilon} \lambda y. (t')^* \circ_m C\Gamma$;
- if $t = uv$, where $\Gamma \vdash_{\Lambda 2\text{prim}} u : C \to A$ and $\Gamma \vdash_{\Lambda 2\text{prim}} v : C$, then $t^* = u^* v^*$ and by the induction hypothesis we have $C_{\forall C \to A} [u] \simeq_{\beta} \lambda y. C_A[u \circ \overline{C}_C[y]] \simeq_{\epsilon} u^* \circ_m C\Gamma$ and $C_{\forall C} [v] \simeq_{\epsilon} v^* \circ_m C\Gamma$. We deduce then $C_A t \simeq_{\epsilon} C_A[u \circ \overline{C}_C[v]] \simeq_{\beta} C_{\forall C \to A} [u] (v^* \circ_m C\Gamma) \simeq_{\epsilon} (u^* \circ C\Gamma)(v^* \circ C\Gamma) = t^* \circ C\Gamma$. 

22
• if \( t = \Lambda Y.t' \), then \( A = \forall Y.B \) and \( t^* = \Lambda Y.(t')^* \), and by the induction hypothesis \( c_B[t'] \simeq_\varepsilon (t')^* \circ_m c_{\Gamma} \), whence \( c_A[t] \simeq_\beta \Lambda Y.c_B[t'] \simeq_\varepsilon \Lambda Y.(t')^* \circ_m c_{\Gamma} = t^* \circ_m c_{\Gamma} \).

• if \( t = uC \), then \( A = B[C/Y] \), \( t^* = u^*C^* \) and by the induction hypothesis \( c_{\forall Y.B}[u] \simeq_\varepsilon u^* \circ_m c_{\Gamma} \). By Lemma [D.3] and the induction hypothesis we have \( c_{B[C/Y]}[u] \simeq_\varepsilon (c_{\forall Y.B}[u])C^* \simeq_\varepsilon (u^* \circ_m c_{\Gamma})C^* = t^* \circ_m c_{\Gamma} \) (as \( Y \notin FV(\Gamma) \)).

• if \( t = * \), then \( A = 1 \) and \( t^* = \Gamma \times x.x \) and we have \( c_A[t] \simeq_\beta t^* = t^* \circ_m c_{\Gamma} \).

• if \( t = \langle u, v \rangle \), then \( A = B \times C \), \( t^* = \Lambda X.\lambda y.x^* \circ_m c_{\Gamma} \) and by the induction hypothesis \( c_B[u] \simeq_\varepsilon u^* \circ_m c_{\Gamma} \) and \( c_C[v] \simeq_\varepsilon v^* \circ_m c_{\Gamma} \). We have then \( c_A[t] \simeq_\beta \Lambda Y.\lambda y.(c_B[\pi_1t])(c_C[\pi_2t]) \simeq_\varepsilon \Lambda Y.\lambda y.\pi_1(y \circ_m c_{\Gamma})(\pi_2(y \circ_m c_{\Gamma})) = t^* \circ_m c_{\Gamma} \).

• if \( t = \pi_1^Au \), then \( A = A_1 \times A_2 \), \( t^* = \pi_1^A \lambda a_1 \lambda a_2 . a_1 \) and by the induction hypothesis \( c_{A_1 \times A_2}[u] \simeq_\beta (\Lambda X.\lambda y.\pi_1^A(y \circ_m c_{\Gamma}))(\pi_2^A(y \circ_m c_{\Gamma})) \simeq_\varepsilon u^* \circ_m c_{\Gamma} \). We have then \( t^* \circ_m c_{\Gamma} = (u^* \circ_m c_{\Gamma})\lambda a_1 \lambda a_2 . a_1 \simeq_\varepsilon (\Lambda X.\lambda y.(c_{A_1}[\pi_1^A u])(c_{A_2}[\pi_2^A u]))\lambda a_1 \lambda a_2 . a_1 = t^* \circ_m c_{\Gamma} \).

• if \( t = \iota u \), then \( A = A_1 + A_2 \), \( t^* = \Lambda X.\lambda a_1 \lambda a_2 . a_1 \) and by the induction hypothesis \( c_{A_1 + A_2}[u] \simeq_\varepsilon u^* \circ_m c_{\Gamma} \). We have then \( c_A[t] \simeq_\beta \Lambda X.\lambda a_1 \lambda a_2 . \delta_X(t, u, a_1)(c_{A_1}[y])(\iota a_2)(c_{A_2}[y]) \simeq_\beta \Lambda X.\lambda a_1 \lambda a_2 . a_1(u^* \circ_m c_{\Gamma}) = t^* \circ_m c_{\Gamma} \).

• if \( t = \delta_C(u, x, v_1, x, v_2) \), then \( t^* = \pi_1^C \lambda x.x^* \circ_m c_{\Gamma} \) and by the induction hypothesis we have \( c_{B_1 + B_2}[u] \simeq_\varepsilon u^* \circ_m c_{\Gamma} \) and \( c_C[v_1] \simeq_\varepsilon v_1^* \circ_m c_{\Gamma, x:B_1} \). We can compute then

\[
t^* \circ_m c_{\Gamma} = (u^* \circ_m c_{\Gamma})C^* \lambda x.(v_1^* \circ_m c_{\Gamma, x:B_1}) \lambda x.(v_2^* \circ_m c_{\Gamma, x:B_2})
\]

\[
\simeq_\varepsilon (c_{B_1 + B_2} \circ u)C^* \lambda x_1.(v_2^* \circ_m c_{\Gamma}) \lambda x_2.(v_2^* \circ_m c_{\Gamma})
\]

\[
\simeq_\beta (\Lambda X.\lambda x_1 \lambda x_2 . \delta_X(u, x, a_1)(c_{B_1}[y])(\iota a_2)(c_{B_2}[y]))C^* \lambda x_1.(v_2^* \circ_m c_{\Gamma, x:B_1}) \lambda x_2.(v_2^* \circ_m c_{\Gamma, x:B_2})
\]

\[
\simeq_\beta \delta_{C^*}(u, x, v_1^*[c_{B_1}[x]/x] \circ_m c_{\Gamma, x:A_1}, x, v_2^*[c_{B_2}[x]/x] \circ_m c_{\Gamma})
\]

\[
= \delta_{C^*}(u, x, v_1^* \circ_m c_{\Gamma, x:B_1}, x, v_2^* \circ_m c_{\Gamma, x:B_2}) \simeq_\varepsilon \delta_{C^*}(u, x, C_{C}[v_1^*], x, C_{C}[v_2^*])
\]

\[
\simeq_\eta C_{C}(\delta_C(u, x, v_1, x, v_2)) = c_{\Gamma}[t]
\]

• if \( t = \xi_C u \), then \( t^* = u^*C \) and by the induction hypothesis \( c_0[u] \simeq_\varepsilon u^* \circ_m c_{\Gamma} \) so we have \( t^* \circ_m c_{\Gamma} = (u^* \circ_m c_{\Gamma})C \simeq_\varepsilon (c_C[u])C \simeq_\varepsilon C_{C}[uC] \) where the last step is an application of Lemma [D.3].

\( \square \)