Theoretical Computer Science 435 (2012) 3–20

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Internal models of system F for decompilation

Stefano Berardi [a,*], Makoto Tatsuta [b]

[a] Department of Computer Science, University of Turin, Corso Svizzera 185, 10149 Torino, Italy

[b] National Institute of Informatics, 2-1-2 Hitotsubashi, Tokyo 101-8430, Japan

ARTICLE INFO

Keywords:
Typed λ-calculus
System F
Semantics of polymorphism
Compiler
Decompiler
de Bruijn level

ABSTRACT

This paper considers Girard's internal coding of each term of System F by some term of a code type. This coding is the type-erasing coding definable already in the simply typed lambda-calculus using only abstraction on term variables. It is shown that there does not exist any decompiler for System F in System F, where the decompiler maps a term of System F to its code. An internal model of F is given by interpreting each type of F by some type equipped with maps between the type and the code type. This paper gives a decompiler–normalizer for this internal model in F, where the decompiler–normalizer maps any term of the internal model to the code of its normal form. It is also shown that for any model of F the composition of this internal model and the model produces another model of F whose equational theory is below untyped beta–eta-equality.

1. Introduction

Let $\mathcal{F}$ denote the Church-style second order λ-calculus with $\beta\eta$-equality. For a definition of $\mathcal{F}$ and a model of $\mathcal{F}$ we refer to [15] and Section 4.

It will be nice if we may find some subsystem of $\mathcal{F}$ which simulates $\mathcal{F}$ itself and has additional useful properties that $\mathcal{F}$ does not have. We will investigate such a subsystem that has the decompilation property. If we allow some extended system for describing a compiler, the subsystem will have a compiler, which enables us meta programming. When we compose the decompiler and the compiler, we will obtain a normalizer, which is important for normalization by evaluation. If we refine the subsystem so that it faithfully represents $\mathcal{F}$, it will give a new $\beta\eta$-complete model.

In order to investigate such a subsystem, it will be better to generalize it to some subsystem of some extension $\mathcal{F}'$ of $\mathcal{F}$. The extensions we have in mind are $\mathcal{F}$ itself or $\mathcal{F}_\omega$.

This paper studies three interrelated problems, namely:

(1) *Normalization by evaluation*. To find a normalization algorithm for $\mathcal{F}$, written in some extension $\mathcal{F}'$ of $\mathcal{F}$.
(2) *Compilation and Decompilation*. To find compilation and decompilation algorithms, written in some extension $\mathcal{F}'$ of $\mathcal{F}$, for the subsystem.
(3) *$\beta\eta$-completeness*. To find some class of $\beta\eta$-complete models for $\mathcal{F}$, that is, a class of models whose equational theory is exactly $\beta\eta$-convertibility.

A normalizer, a compiler and a decompiler are defined w.r.t. some coding of the terms of $\mathcal{F}$ inside $\mathcal{F}$ itself. Abel [1,2] defined a normalizer for $\mathcal{F}$ inside ML. Pfenning and Lee [18] defined a compiler and a decompiler for $\mathcal{F}$ inside an extension $\mathcal{F}_\omega$ of $\mathcal{F}$. The first non-trivial example of $\beta\eta$-complete model of $\mathcal{F}$ is the *BB*-model [6], proved $\beta\eta$-complete in [7] and generalized in [8]. Problems (1), (2), (3) are not related *a priori*, but they may be solved together. Problem (1) requires to find a normalizer (see Section 5), i.e., some family ev_A of terms in $\mathcal{F}'$, indexed over the types A of $\mathcal{F}$, and computing the code of

* Corresponding author.
E-mail addresses: stefano@di.unito.it (S. Berardi), tatsuta@nii.ac.jp (M. Tatsuta).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.02.022

the normal form in $\mathcal{F}$, given the code of a term of type A in $\mathcal{F}$. Problem (2) requires to find two families f_A and g_A of terms of $\mathcal{F}'$, indexed over the types A of $\mathcal{F}$. The terms f_A represent a *decompiler* (see Sections 5 and 6), and compute the code u of a term t of type A in $\mathcal{F}$, from the representation of t in $\mathcal{F}'$. We may justify the use of the word "decompiler" if we consider u as a "source code" of t, and the representation of t as an "executable version" of t. In the case f_A computes the code of the normal form of t, we say that f_A is a *decompiler–normalizer*. The action of f_A is sometimes called "reification", because it transforms a program, considered as an abstract concept, into a concrete datum available for manipulation. The terms g_A represent a *compiler* (see Sections 5 and 6), and they provide the inverse transformation: they compute the representation of t from the code u of the term t. The action of g_A is sometimes called a "reflection": it is a process by which a program may define a new program. Problem (3), instead, requires to find some class of models of $\mathcal{F}$ defined as mathematical structures, whose equational theory is exactly $\beta\eta$. All these problems have independent reasons for interest.

(1) Problem (1), normalization for the image the subsystem of $\mathcal{F}'$ inside $\mathcal{F}'$, has a potential interest from a programming viewpoint: if the language $\mathcal{F}'$ can evaluate the subsystem of itself, it can also implement extensions of this subset. An example taken from real programming is the language Scheme with its primitive `eval`. The language Scheme, indeed, may define its own extensions.
(2) Problem (2), compilation and decompilation of the subsystem of $\mathcal{F}'$ inside $\mathcal{F}'$, has also a potential interest from a programming viewpoint: if a language $\mathcal{F}'$ can decompile the subsystem, then $\mathcal{F}'$ may manipulate the source code of its programs, in order to optimize them. An example of a language having this feature in the real world is again Scheme, in which there are primitives `quote`, `unquote`, for "freezing" and "unfreezing" the execution of any expression of Scheme itself, and for manipulating the syntactical tree of a Scheme expression.
(3) The interest of Problem (3), completeness, lies in the fact that a $\beta\eta$-complete model of $\mathcal{F}$ describes the equality $=_{\beta\eta}$ of the calculus $\mathcal{F}$ in the language of mathematical structures, and explains the mathematical principles which are hidden behind the syntax of $\mathcal{F}$.

After Pfenning and Lee's result, a natural additional request for the problems (1) and (2) is that $\mathcal{F}'$ should be as close to $\mathcal{F}$ as possible. In this paper we address the following version of the problems (1) and (2): whether we may define a compiler, a decompiler or a normalizer for a subsystem representing $\mathcal{F}$ in $\mathcal{F}$ itself, that is, if we require $\mathcal{F}' = \mathcal{F}$. We assume that our subsystem is the image of some map $j : \mathcal{F} \to \mathcal{F}'$ that is compatible with typing and $\beta\eta$-reductions. We call this map an embedding. The embedding maps a term to its representation. A compiler, a decompiler, or a normalizer for the image $j(\mathcal{F})$ of $\mathcal{F}$ may be defined within $\mathcal{F}$ itself. We call *internal* a compiler, a decompiler, or a normalizer which may be written in $\mathcal{F}$ itself. In this paper, we consider the existence of internal normalizers, compilers and decompilers in the case where the embedding is id. We prove that there is no normalizer, compiler nor decompiler for $\mathcal{F}$ written in $\mathcal{F}$ itself, under broad assumptions over the coding of λ-terms in $\mathcal{F}$. Instead we define a new embedding $(.)^{*^c} : \mathcal{F} \to \mathcal{F}$ of $\mathcal{F}$ into itself. Then we define a *decompiler–normalizer*, w.r.t. Girard's coding of λ-terms, and written inside $\mathcal{F}$, for the terms of the image $\mathcal{F}^{*^c} \subseteq \mathcal{F}$ of $\mathcal{F}$. This positive answer is surprising, because for real-world languages decompilation is a hard problem, and because we just showed that no internal normalizer exists for the same coding. Besides, a similar result does not hold for compilation: we prove that for taking the embedding to be $(.)^{*^c}$ (and indeed for *any* choice of embedding $\mathcal{F} \to \mathcal{F}$ and under a broad assumption over the coding) we cannot define a compiler in $\mathcal{F}$ for the terms of the subsystem. The best result for compilation is still Pfenning and Lee's compilation of $\mathcal{F}$, outside $\mathcal{F}$ and inside $\mathcal{F}_3$. We interpret these results as follows. Normalization and compilation for the subsystem require essentially stronger reduction rules than those available in $\mathcal{F}$, for instance, they require the rules of $\mathcal{F}_3$. Decompiling $\mathcal{F}^{*^c}$, instead, means deducing the structure of the normal forms from their observable behavior, and this can be done inside $\mathcal{F}$. Summing up, $\mathcal{F}$ may manipulate its own programs, at some extent, even if the last step, compiling, must be done in $\mathcal{F}_3$.

Instead Problem (3) requires to define a class of $\beta\eta$-complete models for $\mathcal{F}$, a problem which is *a priori* unrelated with the problems (1) and (2). However, we claim that within any model $\mathcal{M}$ of $\mathcal{F}$, we may use the embedding $(.)^{*^c}$ of $\mathcal{F}$ into itself, in order to define a sub-model $\mathcal{M}^{*^c}$ of $\mathcal{M}$ whose equational theory is between $\beta\eta$-equality and untyped $\beta\eta$ equality. At the end of this paper, we conjecture that by modifying the map $(.)^{*^c}$ we may define inside any model $\mathcal{M}$ some sub-model $\mathcal{M}^{*^c}$ whose equational theory is exactly $\beta\eta$ (this was our original goal, but it is not yet solved).

In Section 2 we sum up what is known about the corresponding of the problems (1), (2), (3) for the simply typed lambda calculus $\lambda\!\to$. In the rest of the paper we try to adapt the solutions we have in the case of $\lambda\!\to$ to $\mathcal{F}$. In Section 3 we introduce the definition of $\mathcal{F}$ and we define a type Tm^c of $\mathcal{F}$, coding all untyped λ-terms in $\mathcal{F}$, and we discuss the minimal property a coding of untyped λ-terms should have. In Section 4 we define models of $\mathcal{F}$ and $\beta\eta$-completeness. In Section 5 we discuss some alternative internal codings for $\mathcal{F}$, and we prove that there is no normalizer, compiler, nor decompiler for $\mathcal{F}$ in $\mathcal{F}$ for a broad choice of codings. In Section 6 we define $(.)^*$ and $(.)^{*^c}$, two variants of the same interpretation of $\mathcal{F}$ in $\mathcal{F}$, and in Section 7 we prove there is a decompiler–normalizer (but no compiler) written in $\mathcal{F}$ for the terms of $\mathcal{F}^{*^c}$. In Section 8 we discuss how to define a class of $\beta\eta$-complete models of $\mathcal{F}$ using $(.)^*$, and we prove the first result towards this goal, that we may define a class of models whose equational theory is below untyped $\beta\eta$.

2. $\beta\eta$-completeness, internal decompilation and normalization for simply typed λ-calculus

In this section we sum up what is known about the corresponding of the problems (1), (2), (3) for simply typed lambda calculus $\lambda\!\to$. In the rest of the paper, we try to adapt the solutions we have for $\lambda\!\to$ to $\mathcal{F}$.

We refer to [9] for a definition of $\lambda\!\!\rightarrow$. Friedman considered the problem of defining $\beta\eta$-complete models for $\lambda\!\!\rightarrow$. He considered all set-theoretical models of $\lambda\!\!\rightarrow$. In these models all atomic types o of $\lambda\!\!\rightarrow$ are interpreted by some set, and the type $A \to B$ is interpreted as the set of all maps from the interpretation of A to the interpretation of B. Friedman has proved the following theorem [9]: "a set-theoretical model of $\lambda\!\!\rightarrow$ is $\beta\eta$-complete if and only if all atomic types of $\lambda\!\!\rightarrow$ are interpreted by infinite sets." The proof of Friedman is highly abstract, using Classical Logic, uncountable sets and Choice Axiom. However, if we unwind Friedman's proof, we discover the following elementary and constructive argument, hidden in it. First, in any infinite set-theoretical model there is a decompiler–normalizer for $\lambda\!\!\rightarrow$. Indeed, there is some atomic type Tm representing all untyped λ-terms in a suitable context (see Section 3 for a definition of this coding). There is some family $f_A : A \to \mathrm{Tm}$ of maps of the model, for each simple type A, such that if $t : A$ is a term of $\lambda\!\!\rightarrow$, then $f_A(t) =$ the code of the normal form t' of t in Tm. To be more accurate, $f_A(t)$ returns the code of the untyped λ-term, which is the erasure of the $\beta\eta$-long normal form of the term t (again, see Section 3 for details). However, this amounts to the same, because the typing information required for a normal term of $\lambda\!\!\rightarrow$ may be recovered from the erasure of the term. The argument hidden in Friedman's proof continues as follows: if t, u are definable in $\lambda\!\!\rightarrow$ and $t = u$ in the set-theoretical model, then $f_A(t) = f_A(u) : \mathrm{Tm}$ in the same model, therefore t, u have the same ($\beta\eta$-long) normal form, and hence $t =_{\beta\eta} u$.

Surprisingly, the set-theoretical maps f_A turn out to be definable in $\lambda\!\!\rightarrow$. The definition of f_A requires two free variables $\mathrm{ap} : \mathrm{Tm} \to \mathrm{Tm} \to \mathrm{Tm}$ and $\mathrm{lam} : (\mathrm{Tm} \to \mathrm{Tm}) \to \mathrm{Tm}$ to represent application and lambda abstraction. Joly [11] explicitly defined this decompiler–normalizer $f_A : A \to \mathrm{Tm}$ for $\lambda\!\!\rightarrow$, though he only studied it as an example of a family of injections definable in $\lambda\!\!\rightarrow$, from all types to a single type. Berger [5] and Werner [12] studied this internal decompiler–normalizer for $\lambda\!\!\rightarrow$, and then defined a compiler and a normalizer for $\lambda\!\!\rightarrow$ in Gödel's system $\mathcal{T}$, an extension of $\lambda\!\!\rightarrow$. These results are solutions for the problems (1), (2), (3) for $\lambda\!\!\rightarrow$ which are probably optimal: there are a family of $\beta\eta$-complete models, an internal decompiler–normalizer for $\lambda\!\!\rightarrow$, and compilers and normalizers written in some extensions $\mathcal{T}$ of $\lambda\!\!\rightarrow$. We will try to adapt the definition of a decompiler–normalizer from $\lambda\!\!\rightarrow$ to $\mathcal{F}$. The problem is that the definition of a decompiler–normalizer in $\lambda\!\!\rightarrow$ heavily relies on two properties of $\lambda\!\!\rightarrow$: the fact that we may recover a normal form from its erasure, and the existence of a set-theoretical model. These properties do not hold for $\mathcal{F}$.

3. The system $\mathcal{F}$ and a type Tm^c coding untyped λ-terms

In this section we introduce $\mathcal{F}$, and one possible choice for a type Tm^c in which we may represent an untyped λ-term $\mathtt{t}$ (possibly not normal) by some ($\beta\eta$-long) normal form $[\mathtt{t}]^c \in \mathcal{F}$. $\beta\eta$-long normal forms in $\mathcal{F}$ are defined below. The definition of Tm^c is originally from [10]. Representation of untyped λ-terms in Tm^c is up to α-rule and it is not up to β-rule. We consider two α-convertible untyped λ-terms as two representation of the same term. If $\mathtt{t}, \mathtt{u}$ are two untyped λ-terms which are *not* α-convertible, then $\mathtt{t}, \mathtt{u}$ have two representations in Tm^c by some ($\beta\eta$-long) normal forms which are *not* $\beta\eta$-convertible. In this section and in Section 4 we discuss the minimum property that a coding of untyped λ-terms should have, and in Section 5 we describe an alternative coding for untyped λ-terms in $\mathcal{F}$, with de Bruijn's "levels" of variables [4,13].

We denote the untyped λ-calculus by Λ. We assume having variable names $\mathtt{x}, \mathtt{y}, \mathtt{a}, \mathtt{b}, \ldots$. The syntax of Λ is $\mathtt{t} ::= \mathtt{x} \mid (\mathtt{tt}) \mid \lambda\mathtt{x}.\mathtt{t}$. We use "typewriter" letters $\mathtt{t}, \mathtt{u}, \mathtt{v}, \ldots$ to denote untyped λ-terms. The length of an untyped λ-term is the number of symbols (variables, applications and λ-abstraction) in it: we denote the length of $\mathtt{t} \in \Lambda$ by $\mathtt{len}(\mathtt{t})$.

By $\mathcal{F}$ we denote the second order λ-calculus, as defined in [15]. We assume having type variables $\mathrm{Tm}, \mathrm{Var}, \mathrm{dB}, \alpha, \beta, \ldots$. The syntax of $\mathcal{F}$ is $A ::= \alpha \mid A \to A \mid \forall\alpha A$ for types. We write both $A \to B \to C$ and $A, B \to C$ for $A \to (B \to C)$. We write $\forall\alpha.A \to B$ for $\forall\alpha.(A \to B)$. A context is a set $\Gamma = \{x_1 : A_1, \ldots, x_m : A_m\}$ where $x_i \neq x_j$ for $i \neq j$. We write $x, y : A$ for $x : A, y : A$. We assume having variable names $\mathrm{ap}, \mathrm{lam}, 0, S, \mathrm{Var}, x, y, z, a_1, a_2, f_1, f_2, g_1, g_2, \ldots$. The syntax for pseudo-terms is $t ::= x \mid \lambda x : A.t \mid tt \mid \lambda\alpha.t \mid tA$. We write $t(t_1)$ and $t(t_1, t_2)$ for tt_1 and $(tt_1)t_2$ respectively. We define $\lambda\_ : A.t$ as $\lambda x : A.t$ for some fresh variable x. We will denote the set of the free variables in t by $\mathrm{FV}(t)$. There are introduction and elimination rules for $\to$ and $\forall$, assigning types to some pseudo-terms. We write $\Gamma \vdash t : A$ for "t has the type A in $\mathcal{F}$ in the context Γ". The degree $\deg(A)$ of A is inductively defined by $\deg(\alpha) = 0$, $\deg(A \to B) = \max(\deg(A) + 1, \deg(B))$ and $\deg(\forall\alpha.A) = \deg(A) + 1$. We write $=_\alpha$ for the α-convertibility relation: equality up to variable renaming. The reduction rules of $\mathcal{F}$ are β- and η-reductions. We write $=_{\beta\eta}$ for the convertibility relation up to β and η rules.

By "$\beta\eta$-long normal form" of a term t of $\mathcal{F}$ we mean the longest η-expansion of the β-normal form t' of t which is still β-normal. Alternatively, we define the $\beta\eta$-long normal form t'' of t by repeatedly replacing each maximal subterm $u = x(t_1, \ldots, t_n)$ of t' by $\lambda\alpha.x(t_1, \ldots, t_n, \alpha)$ if u has the type $\forall\beta.B$, by $\lambda y : B.x(t_1, \ldots, t_n, y)$ if u has the type $B \to C$. We continue until we reach a term t'' in which all maximal subterms of the form $x(t_1, \ldots, t_n)$ have a variable type. For instance, if $A = \forall\alpha.\alpha \to \alpha$, the $\beta\eta$-long normal form of $\lambda x : A.x$ is $\lambda x : A.\lambda\alpha.\lambda y : \alpha.x(\alpha, y)$. The $\beta\eta$-long normal form exists by a corollary of Girard's Normalization Theorem for $\mathcal{F}$, and it is unique by the Church–Rosser confluence property for $\mathcal{F}$ [19].

For every type A of $\mathcal{F}$, we write id_A for the identity $\lambda x : A.x$ on the type A, we write Id for the type $\forall\alpha.\alpha \to \alpha$, and id for the polymorphic identity $\lambda\alpha.\lambda x : \alpha.x : \mathrm{Id}$. The term $t^n u$ is defined as $(t(\ldots(tu)\ldots))$ (n times of t) for any natural number n.

For every term of $\mathcal{F}$, we write $|t| \in \Lambda$ for the untyped λ-term obtained by stripping all type information from t, and replacing the variable x_i of $\mathcal{F}$ with the variable $\mathtt{x}_i$ of Λ. We may recursively define $|x_i| = \mathtt{x}_i$, $|\lambda x : A.t| = \lambda\mathtt{x}.|t|$, $|tu| = |t||u|$, $|\lambda\alpha.t| = |t|$, $|tA| = |t|$. An equational theory is any equivalence relation over the terms of $\mathcal{F}$, compatible with term formation and with $\beta\eta$-reduction. We call *"untyped $\beta\eta$"*, and we denote with $|\beta\eta|$, the equational theory on $\mathcal{F}$ such that, for all terms t, u of $\mathcal{F}$, we have $t =_{|\beta\eta|} u$ if and only if $|t| =_{\beta\eta} |u|$ in Λ.

We claim that $\beta\eta \subset |\beta\eta|$, that is, the equational theory $|\beta\eta|$ for $\mathcal{F}$ is larger than the equational theory $\beta\eta$, because a $\beta\eta$-long normal form of $\mathcal{F}$ cannot be uniquely recovered from its erasure. An example is given as follows. Let $\mathrm{Void} = \forall\alpha.\alpha$ and $t = \lambda x : \mathrm{Void}.x$ and $u = \lambda x : \mathrm{Void}.x(\mathrm{Void})$. Then $t, u : \mathrm{Void} \to \mathrm{Void}$ and $t \neq_{\beta\eta} u$ (t, u are different $\beta\eta$-long normal forms), while $t =_{|\beta\eta|} u$, because $|t| = \lambda \mathrm{x}.\mathrm{x} = |u|$.

We want to represent all untyped λ-terms by the elements of some type of $\mathcal{F}$. There are several choices for this coding, but we discuss them later. In this section we introduce the type Tm^c proposed by Girard [10], which we are going to adopt.

We first explain how to internalize the notion of booleans and natural numbers. We introduce a set of data types representable in $\mathcal{F}$ which is suitable for the paper: mutually recursive first-order data types.

Definition 3.1 (*Data Types*)**.** We introduce booleans, natural numbers and data types in $\mathcal{F}$, as follows.

(1) A *mutually recursive first order data type*, just a data type for short, is any closed type of the form $\forall\alpha_1, \ldots, \alpha_n.A$, with no connective $\forall$ in A, and $\deg(A) \leq 2$.
(2) $\mathrm{Bool} = \forall\alpha.\alpha \to \alpha \to \alpha$, and $\mathrm{True} = \lambda\alpha.\lambda x, y : \alpha.x$, $\mathrm{False} = \lambda\alpha.\lambda x, y : \alpha.y$.
(3) $\mathrm{Nat} = \forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha)$ and $\underline{n} = \lambda\alpha.\lambda f : (\alpha \to \alpha).\lambda a : \alpha.f^n(a)$ for any natural number $n \in N$.
(4) $S_l, S_r, S : \mathrm{Nat} \to \mathrm{Nat}$ are defined in the context $\{x : \mathrm{Nat}, f : \alpha \to \alpha, a : \alpha\}$ by: $S_l(x, \alpha, f, a) = x(\alpha, f, f(a))$ and $S_r(x, \alpha, f, a) = f(x(\alpha, f, a))$ and $S = S_r$.

In system $\mathcal{F}$, mutually recursive data types translate the idea of first order closure of a finite set of maps. Assume be given any finite set $\Gamma = \{\alpha_1, \ldots, \alpha_m\}$ of type variables, any finite set of declaration $\Delta = \{f_1 : C_1, \ldots, f_n : C_n\}$ of first order functions, such that for all $1 \leq i \leq n$ we have $C_i = (\alpha_{i_1}, \ldots, \alpha_{i_n} \to \alpha_{i_{p+1}})$ for some $\alpha_{i_1}, \ldots, \alpha_{i_{p+1}} \in \Gamma$. The first order closure is the smallest set of well-typed terms whose type is in Γ and closed under $f_1, \ldots, f_n$. Then we arbitrarily select one type $\alpha \in \Gamma$, and we call the terms of type α the elements of the first order closure. In the case there is some type $\beta \neq \alpha$, $\beta \in \Gamma$, then the elements of type β are considered to be parameters which may be used to define some element of type α: we do not consider them elements of the first order closure. The first order closure may be coded in $\mathcal{F}$ by the data type $D = \forall\alpha_1, \ldots, \alpha_m.(C_1, \ldots, C_n \to \alpha)$. We may prove that closed $\beta\eta$-long normal forms of type D in $\mathcal{F}$ are in canonical bijection with the elements of the first order closure.

Mutually recursive data types translate in system $\mathcal{F}$ the idea of first order closure of a finite set of maps. Assume be given any finite set $\Gamma = \{\alpha_1, \ldots, \alpha_m\}$ of type variables, any finite set of declaration $\Delta = \{f_1 : C_1, \ldots, f_n : C_n\}$ of first order functions, such that for all $1 \leq i \leq n$ we have $C_i = (\alpha_{i_1}, \ldots, \alpha_{i_n} \to \alpha_{i_{p+1}})$ for some $\alpha_{i_1}, \ldots, \alpha_{i_{p+1}} \in \Gamma$. We take the smallest set of well-typed terms whose type is in Γ and closed under $f_1, \ldots, f_n$. Then we arbitrarily select one type $\alpha \in \Gamma$, and we call the terms of type α the elements of the first order closure. In the case there is some type $\beta \neq \alpha$, $\beta \in \Gamma$, then the elements of type β are considered to be parameters which may be used to define some element of type α: they are not elements of the first order closure. The first order closure may be coded in $\mathcal{F}$ by the data type $D = \forall\alpha_1, \ldots, \alpha_m.(C_1, \ldots, C_n \to \alpha)$. We may prove that closed $\beta\eta$-long normal forms of type D in $\mathcal{F}$ are in canonical bijection with the elements of the first order closure.

Bool and Nat are data types of $\mathcal{F}$, whose closed $\beta\eta$-long normal forms are True, False, $\underline{0}, \underline{1}, \underline{2} \ldots$, in bijection with booleans, and with natural numbers. S_l, S_r are called the left- and right-successor. We call right-successor also "successor" and we also denote it by just S. By definition we have $S_l(\underline{n}) =_{\beta\eta} \underline{n+1} =_{\beta\eta} S_r(\underline{n}) = S(\underline{n})$ for all $n \in N$.

Using the type Bool we may define the observational equality over terms.

Definition 3.2 (*Observational Equality*)**.** Assume t, u are closed terms having a closed type A in $\mathcal{F}$.

(1) t, u are observationally equal if for all closed term $f : A \to \mathrm{Bool}$ of $\mathcal{F}$, $f(t) =_{\beta\eta} f(u)$.
(2) We write $t =_{\mathcal{O}} u$ for "t, u are observationally equal".

Observational equality is a consistent equational theory (i.e., it does not equate all terms), and it is the largest equational theory for $\mathcal{F}$ [16].

We internalize the notion of untyped λ-terms inside $\mathcal{F}$, first by an open type Tm, and then by a closed type Tm^c, using a technique called *Higher-order abstract syntax* [17], in which binders in the object-language are represented via binders in the meta-language. The particular definition of Tm^c is taken from [10]. Fix a type variable Tm. We fix the context $\Gamma_{\mathrm{Tm}} = \{\mathrm{lam} : ((\mathrm{Tm} \to \mathrm{Tm}) \to \mathrm{Tm}), \mathrm{ap} : (\mathrm{Tm} \to \mathrm{Tm} \to \mathrm{Tm})\}$. Then the elements of Tm represent the syntax trees of untyped λ-terms in $\mathcal{F}$. The variables lam and ap codify λ-abstraction and application: if $f : \mathrm{Tm} \to \mathrm{Tm}$ then $\mathrm{lam}(\lambda x : \mathrm{Tm}.f(x)) : \mathrm{Tm}$ codifies the λ-abstraction of f, and if $x, y : \mathrm{Tm}$ then $\mathrm{ap}(x, y) : \mathrm{Tm}$ codifies the application of x to y. Codes in Tm are $\beta\eta$-long normal forms: we do *not* have reduction rules for them. An example: if $x, y : \mathrm{Tm}$ are variables, then the term $\mathrm{ap}(\mathrm{lam}(\lambda x : \mathrm{Tm}.x), y) : \mathrm{Tm}$ of $\mathcal{F}$ codifies the untyped λ-term $(\lambda\mathrm{x}.\mathrm{x})(\mathrm{y}) \in \Lambda$, but $(\lambda\mathrm{x}.\mathrm{x})(\mathrm{y}) =_{\beta\eta} \mathrm{y}$, while $\mathrm{ap}(\mathrm{lam}(\lambda x : \mathrm{Tm}.x), y) \neq_{\beta\eta} y$, because the terms $\mathrm{ap}(\mathrm{lam}(\lambda x : \mathrm{Tm}.x), y) : \mathrm{Tm}$ and $y : \mathrm{Tm}$ of $\mathcal{F}$ are different $\beta\eta$-long normal forms.

According to the view expressed in [10], the free variables Tm, lam, ap of Γ_{Tm} define a *generic coding* for Λ in $\mathcal{F}$. In Definition 5.6 we will introduce a triple DB, Lam, Ap replacing these variables with an example of one *concrete* coding, de Bruijn coding "with levels" of variables [4,13]. Then we will prove that for both codings (and indeed, for any coding satisfying a minimum of requests) there is no compiler nor decompiler in $\mathcal{F}$. Indeed, it seems impossible in general to construct a

decompiler which turns two observationally equivalent functions of type Nat → Nat into the same code. We will formalize this remark for the left- and right-successor maps. However, with the interpretation $[\![.]\!]$ of $\mathcal{F}$ into $\mathcal{F}$, integers have a more informative encoding Nat*, and we will prove that a decompiler does exists, even if this is not intuitive.

We formally define the interpretations $[.]$ of Λ and $[\![.]\!]$ of $\mathcal{F}$ into the open type Tm in the context Γ_{Tm}. The map $[.]$ applied to any closed $\mathtt{t} \in \Lambda$ interprets abstractions and applications of $\mathtt{t}$ by lam and ap. When the map $[\![.]\!]$ is applied to a term of $\mathcal{F}$, it first forgets abstractions and applications over types, then, it translates abstractions and applications over terms by lam and ap.

Definition 3.3 (*Interpreting Λ in $\mathcal{F}$*)**.** Assume $\mathtt{t} \in \Lambda$ be an untyped λ-term with $\mathtt{FV(t)} \subseteq \{\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{m}\}$. Let $\Gamma = \{x_1 : A_m, \ldots, x_m : A_m\}$ and u be a term of $\mathcal{F}$ such that $\Gamma \vdash u : A$. Let σ be the map $a_1/\mathtt{x}_1, \ldots, a_m/\mathtt{x}_\mathtt{m}$ from variables of Λ to variables of type Tm, with $a_1, \ldots, a_m$ pairwise distinct. We recursively define $[\mathtt{t}]_\sigma$: Tm in the context $\Gamma_{\mathrm{Tm}} \cup \{a_1 : \mathrm{Tm}, \ldots, a_m : \mathrm{Tm}\}$ by:

(1) *(variable)* $[\mathtt{x}_i]_\sigma = a_i$.
(2) *(abstraction)* $[\lambda\mathtt{x}.\mathtt{t}]_\sigma = \mathrm{lam}(\lambda a : \mathrm{Tm}.[\mathtt{t}]_{\sigma, a/\mathtt{x}})$ (a fresh variable).
(3) *(application)* $[\mathtt{t}_1\mathtt{t}_2]_\sigma = \mathrm{ap}([\mathtt{t}_1]_\sigma, [\mathtt{t}_2]_\sigma)$.

When $m = 0$, we abbreviate $[\mathtt{t}]_\sigma$ with $[\mathtt{t}]$. We set $[\![u]\!]_\sigma = [|u|]_\sigma$ and $[\![u]\!] = [|u|]$.

We interpret a term of $\mathcal{F}$ by first stripping off its type information. If u is a term of $\mathcal{F}$, then $[\![u]\!]_\sigma$ is a term of $\mathcal{F}$ of type Tm and context $\Gamma_{\mathrm{Tm}} \cup \{a_1 : \mathrm{Tm}, \ldots, a_m : \mathrm{Tm}\}$. When u is closed then we abbreviate $[\![u]\!]_\sigma$ with $[\![u]\!]$. We may avoid the use of free variables lam, ap and define a *closed* type $\mathrm{Tm}^c = \forall\mathrm{Tm}.((\mathrm{Tm} \to \mathrm{Tm}) \to \mathrm{Tm}) \to (\mathrm{Tm} \to \mathrm{Tm} \to \mathrm{Tm}) \to \mathrm{Tm}$ of $\mathcal{F}$ representing all closed untyped λ-terms. Define $\lambda\Gamma_{\mathrm{Tm}}$ as the sequence of λ-abstractions

$$\lambda\mathrm{Tm}.\lambda\mathrm{lam} : (\mathrm{Tm} \to \mathrm{Tm}) \to \mathrm{Tm}.\lambda\mathrm{ap} : (\mathrm{Tm}, \mathrm{Tm} \to \mathrm{Tm}),$$

and define $t\Gamma_{\mathrm{Tm}}$ as the sequence of applications $t(\mathrm{Tm})(\mathrm{lam})(\mathrm{ap})$. The closed interpretation of a closed $\mathtt{t} \in \Lambda$ in Tm^c is $[\mathtt{t}]^c = \lambda\Gamma_{\mathrm{Tm}}.[\mathtt{t}] : \mathrm{Tm}^c$, and the closed interpretation of a closed $u \in \mathcal{F}$ in Tm^c is $[\![u]\!]^c = \lambda\Gamma_{\mathrm{Tm}}.[\![u]\!] : \mathrm{Tm}^c$. Closed terms of type Tm^c are, up to $\beta\eta$-rule, exactly the closures of the terms of type Tm in the context Γ_{Tm}. Tm^c is *not* a data type because $\mathrm{Tm}^c = \forall\alpha.A$ for some A such that $\deg(A) = 3$. We may define a closed term $\mathrm{ap}^c : \mathrm{Tm}^c \to \mathrm{Tm}^c \to \mathrm{Tm}^c$ such that $\mathrm{ap}^c([\![\mathtt{a}]\!]^c, [\![\mathtt{b}]\!]^c) =_{\beta\eta} [\![\mathtt{ab}]\!]^c$, by setting $\mathrm{ap}^c(x, y) = \lambda\Gamma_{\mathrm{Tm}}.\mathrm{ap}(x(\mathrm{Tm}, \mathrm{lam}, \mathrm{ap}), y(\mathrm{Tm}, \mathrm{lam}, \mathrm{ap}))$ in the context $\{x : \mathrm{Tm}^c, y : \mathrm{Tm}^c\}$. There is no way of defining a closed corresponding $\mathrm{lam}^c : ((\mathrm{Tm}^c \to \mathrm{Tm}^c) \to \mathrm{Tm}^c)$ of lam, though.

We list the properties we consider more relevant for *any* possible coding of $\mathcal{F}$ into $\mathcal{F}$. We state them only for a coding using closed types and terms, for short.

Definition 3.4 (*Properties of Coding*)**.** Assume T is any closed type of $\mathcal{F}$ equipped with a coding $[.]$ of closed untyped λ-terms into closed terms of type T. Let $[\![t]\!] = [|t|]$ be the associated code of terms of $\mathcal{F}$.

(1) $[.]$ is *normal* if $[\mathtt{t}]$ is a $\beta\eta$-long normal form, for all $\mathtt{u} \in \Lambda$.
(2) $[.]$ is *surjective* if for all closed $c : T$ we have $c =_{\beta\eta} [\mathtt{u}]$ for some closed $\mathtt{u} \in \Lambda$.
(3) $[.]$ is *injective* if $[\mathtt{t}] =_{\beta\eta} [\mathtt{u}]$ implies $\mathtt{t} =_\alpha \mathtt{u}$, for all $\mathtt{t}, \mathtt{u} \in \Lambda$.
(4) $[.]$ is *extending* if $\mathtt{len(t)} < \mathtt{len}(|[\mathtt{t}]|)$, for all $\mathtt{t} \in \Lambda$.
(5) $K : T \to T$ *internalizes* the coding $[.]$ in $\mathcal{F}$ if $K([\![t]\!]) = [\![[\![t]\!]]\!]$.

"Normal" means that a coding produces a $\beta\eta$-long normal form. This is a natural request for any type of data, besides, we may always replace a coding by a normal coding. "Surjective" means that every closed term of type T is the interpretation of some untyped λ-term. "Injective" means that two interpretations of untyped λ-terms are $\beta\eta$-equal in $\mathcal{F}$ if and only if the two original untyped λ-terms are α-convertible (are the "same" term). Both surjective and injective are minimal requests over a coding. A coding is "extending" if, after erasing all types, the coding of an untyped λ-term is longer than the original term. This is the case of all known codings which interpret λ-abstractions and applications by some operators. For instance, any application $\mathtt{tu}$ is represented by some notation of the form $\mathrm{ap}([\![t]\!], [\![u]\!])$, which is at least one symbol more. An "internalization" is a map representing the coding of terms of type T inside $\mathcal{F}$. All known codings may be internalized by some map K. Informally, the reason is that $\mathcal{F}$ may represent all maps provably total in Second Order Arithmetic [10].

The coding $[.]^c$ is surjective and injective.

Lemma 3.5 (*Injectivity of $[.]^c$*)**.** (1) *(Surjective) If $\Gamma_{Tm} \vdash c : Tm$, then $c =_{\beta\eta} [\mathtt{u}]$ for some closed $\mathtt{u} \in \Lambda$.*
(2) *(Injective) If $\mathtt{u}_1, \mathtt{u}_2 \in \Lambda$, then $\mathtt{u}_1 =_\alpha \mathtt{u}_2$ if and only if $[\mathtt{u}_1] =_{\beta\eta} [\mathtt{u}_2]$.*

The same properties hold if we replace $[.]$ by $[.]^c$ and we consider closed $c : Tm^c$.

Proof. By induction on c, we can prove the following statement which implies both (1) and (2). We assume $\Gamma_{\mathrm{Tm}}, a_1 : \mathrm{Tm}, \ldots, a_n : \mathrm{Tm} \vdash c : \mathrm{Tm}$, and c is $\beta\eta$-long normal, and we prove that $c = [\mathtt{u}]_\sigma$ for some $\mathtt{u} \in \Lambda$, unique up to α-rule, and for some $\sigma = a_1/\mathtt{x}_1, \ldots, a_n/\mathtt{x}_\mathtt{n}$. Indeed, by analysis of the normal form, either $c = a_i = [\mathtt{x}_i]_\sigma$ or $c = \mathrm{ap}(c_1, c_2)$ or $c = \mathrm{lam}(\lambda a.d)$ for some $c_1, c_2 : \mathrm{Tm}$ in the same context, and some d in the context extended with $a : \mathrm{Tm}$. c_1, c_2, d are

$\beta\eta$-long normal, and by induction hypothesis on c_1, c_2, d we deduce $c_i = [\mathtt{u}_i]_\sigma$ and $d = [\mathtt{v}]_{\sigma,a/\mathtt{x}}$ for some (unique, up to α-rule) untyped λ-terms $\mathtt{u}_1, \mathtt{u}_2, \mathtt{v}$. We conclude that $c = [\mathtt{u}]_\sigma$ for some (unique, up to α-rule) untyped λ-term $\mathtt{u}$. □

The coding in Tm^c expands the term and the map $[\![.]\!]^c$ may be internalized in $\mathcal{F}$.

Lemma 3.6 (*Internalization of Coding*)**.** (1) *(Normal) For all closed* $\mathtt{t} \in \Lambda$, $[\![\mathtt{t}]\!]^c$ *is a* $\beta\eta$*-long normal form.*
(2) *(Expansion) For all closed* $\mathtt{t} \in \Lambda$ *we have* $\mathtt{len}(\mathtt{t}) < \mathtt{len}(|[\mathtt{t}]^c|)$.
(3) *(K) There is some closed term* $K : Tm^c \to Tm^c$ *of* $\mathcal{F}$ *such that* $K([\![t]\!]^c) =_{\beta\eta} [\![[\![t]\!]^c]\!]^c$ *for all closed terms* t *of* $\mathcal{F}$.

Proof. (1) Immediate by definition of $[\![\mathtt{t}]\!]^c$.
(2) By definition we may check that $\mathtt{len}(|[\mathtt{x}]|) = 1 = \mathtt{len}(\mathtt{x})$ and $\mathtt{len}(|[\lambda\mathtt{x}.\mathtt{t}]|) = \mathtt{len}(\mathtt{lam}(\lambda\mathtt{a}.|[\mathtt{t}]|)) = 3 + \mathtt{len}(|[\mathtt{t}]|)$ and $\mathtt{len}(|[\mathtt{tu}]|) = \mathtt{len}(\mathtt{ap}(|[\mathtt{t}]|, |[\mathtt{u}]|)) =$ (there are two applications hidden in $\mathtt{ap}(.,.)$) $3 + \mathtt{len}(|[\mathtt{t}]|) + \mathtt{len}(|[\mathtt{u}]|)$. By induction over $\mathtt{t} \in \Lambda$ we conclude that for all closed $\mathtt{t} \in \Lambda$ we have $\mathtt{len}(\mathtt{t}) < \mathtt{len}(|[\mathtt{t}]|) < \mathtt{len}(|[\mathtt{t}]^c|)$.
(3) If we unfold the definition of $[\![[\![t]\!]]\!]$, we notice that the free variables lam, ap in $[\![t]\!]$ are replaced by some fresh variables $\mathrm{lam}' : \mathrm{Tm}, \mathrm{ap}' : \mathrm{Tm}$ in $[\![[\![t]\!]]\!]$. Let $\Gamma = \Gamma_{\mathrm{Tm}} \cup \{\mathrm{lam}' : \mathrm{Tm}, \mathrm{ap}' : \mathrm{Tm}\}$. In the context Γ, we define $\mathrm{lam}_0(f) = \mathrm{ap}(\mathrm{lam}', \mathrm{lam}(f)) : \mathrm{Tm}$, where $f : (\mathrm{Tm} \to \mathrm{Tm})$. In the same context we define $\mathrm{ap}_0(a, b) = \mathrm{ap}(\mathrm{ap}(\mathrm{ap}', a), b) : \mathrm{Tm}$, where $a, b : \mathrm{Tm}$. By induction on the term t of $\mathcal{F}$ we prove $[\![t]\!][\mathrm{lam}_0/\mathrm{lam}, \mathrm{ap}_0/\mathrm{ap}] =_{\beta\eta} [\![[\![t]\!]]\!]$. We define a term $K_0 : \mathrm{Tm}^c \to \mathrm{Tm}^c$ with free variables $\{\mathrm{lam}' : \mathrm{Tm}, \mathrm{ap}' : \mathrm{Tm}\}$ by the following equation in Γ: $K_0(x, \mathrm{Tm}, \mathrm{lam}, \mathrm{ap}) = x(\mathrm{Tm}, \mathrm{lam}_0, \mathrm{ap}_0) : \mathrm{Tm}$. By definition, $K_0([\![t]\!]^c, \mathrm{Tm}, \mathrm{lam}, \mathrm{ap}) =_{\beta\eta} [\![t]\!][\mathrm{lam}_0/\mathrm{lam}, \mathrm{ap}_0/\mathrm{ap}] =_{\beta\eta} [\![[\![t]\!]]\!]$. Now we define a closed term $K : \mathrm{Tm}^c \to \mathrm{Tm}^c$ by the following equation in Γ_{Tm}: $K(x, \mathrm{Tm}, \mathrm{lam}, \mathrm{ap}) = \mathrm{lam}(\lambda\mathrm{lam}' : \mathrm{Tm}.\mathrm{lam}(\lambda\mathrm{ap}' : \mathrm{Tm}.K_0(x, \mathrm{Tm}, \mathrm{lam}, \mathrm{ap})))$. Thus, $K([\![t]\!]^c, \mathrm{Tm}, \mathrm{lam}, \mathrm{ap}) =_{\beta\eta} \mathrm{lam}(\lambda\mathrm{lam}' : \mathrm{Tm}.\mathrm{lam}(\lambda\mathrm{ap}' : \mathrm{Tm}.[\![[\![t]\!]]\!])) =_{\beta\eta} [\![\lambda\mathrm{Tm}.\lambda\mathrm{lam}' : \mathrm{Tm}.\lambda\mathrm{ap}' : \mathrm{Tm}.[\![t]\!]]\!] =_{\beta\eta} [\![[\![t]\!]^c]\!]$. By a closure in the context Γ_{Tm} we conclude $K([\![t]\!]^c) =_{\beta\eta} \lambda\Gamma_{\mathrm{Tm}}.[\![[\![t]\!]^c]\!] = [\![[\![t]\!]^c]\!]^c$ in the empty context. □

4. Models of $\mathcal{F}$ and $\beta\eta$-completeness

In this section we define the models of $\mathcal{F}$, and what are, in our opinion, all minimum requests for a coding of untyped λ-terms in $\mathcal{F}$. Then we introduce the notion of $\beta\eta$-completeness for types and for models of $\mathcal{F}$.

The definition of models is taken from the definition of β-model in Mitchell [15]. We add the requirement that Mitchell's maps $\Phi_{a,b}$ and Φ_F, interpreting term and type application, are *injective*: this is equivalent to ask that our models satisfy η-rule. There is another (this time purely stylistic) difference with the original Mitchell's definition: in the signature of the model we explicitly indicate the list of kinds Tp, Pred interpreting the set of types and predicates, and the kind constants $(.\Rightarrow.)$, $\Pi(.)$, interpreting arrow and quantifier operators. The first step in the definition of a model for $\mathcal{F}$ is the definition of a frame. A frame for $\mathcal{F}$ is a structure in which we may interpret types and terms of $\mathcal{F}$, but is not yet an interpretation.

Definition 4.1 ($\mathcal{F}$*-Frames*)**.** A *frame for* $\mathcal{F}$ is a tuple $\mathcal{M} = \langle \mathrm{Tp}, \mathtt{E}(.), \mathrm{Pred}, \Rightarrow, \Pi, \{\Phi_{a,b}\}_{a,b\in\mathrm{Tp}}, \{\Phi_F\}_{F\in\mathrm{Pred}}\rangle$:

(1) Tp is a set of elements called *"the types"* of $\mathcal{M}$.
(2) For all $a \in \mathrm{Tp}$, $\mathtt{E}(a)$ is a set of elements called *"the terms of type a"* of $\mathcal{M}$.
(3) $\mathrm{Pred} \subseteq (\mathrm{Tp} \to \mathrm{Tp})$ is a set of maps over types of $\mathcal{M}$ called *"the predicates"* of $\mathcal{M}$.
(4) $\Rightarrow : \mathrm{Tp} \to \mathrm{Tp} \to \mathrm{Tp}$ is a map on types of $\mathcal{M}$ called *"arrow"*, which we write in infix notation. For all $a, b \in \mathrm{Tp}$, $\Phi_{a,b}$ is an *injection* from $\mathtt{E}(a \Rightarrow b)$ to $(\mathtt{E}(a) \to \mathtt{E}(b))$ interpreting *application to a term*.
(5) $\Pi : \mathrm{Pred} \to \mathrm{Tp}$ is a map from the predicates of $\mathcal{M}$ to the types of $\mathcal{M}$ called *"quantifier"*. For all $F \in \mathrm{Pred}$, Φ_F is an *injection* from $\mathtt{E}(\Pi(F))$ to $\Pi_{a\in\mathrm{Tp}}\mathtt{E}(F(a))$ interpreting *application to a type*.

Elements of type Pred are also called "type constructors" in the literature, especially in the papers describing models of the higher order systems, like $\mathcal{F}_3$, *CC*. The next step in the definition of a model is introducing the notion of an interpretation map and a type structure. A type structure is a triple of some set Tp, some $\mathtt{E}(.)$, associating to each $a \in \mathrm{Tp}$ some set $\mathtt{E}(a)$, and some interpretation map $[\![\![.]\!]\!]$, sending any type of $\mathcal{F}$ into Tp, and any term of $\mathcal{F}$ into $\mathtt{E}(a)$ for some $a \in \mathrm{Tp}$.

Definition 4.2 (*Interpretations and Type Structures*)**.** Let Tp be any set and $\mathtt{E}(.)$ be any map associating to each $a \in \mathrm{Tp}$ some set $\mathtt{E}(a)$. Assume that $\Gamma = \{\alpha_1, \ldots, \alpha_n\}$ is a set of type variables of $\mathcal{F}$, $\sigma : \Gamma \to \mathrm{Tp}$ is a map from the variables in Γ to the set Tp, and A is any type of $\mathcal{F}$ with $\mathtt{FV}(A) \subseteq \Gamma$.

(1) $[\![\![.]\!]\!]_{(.)}$ is an *interpretation for the types* in $\mathcal{M}$ if: $[\![\![.]\!]\!]_{(.)}$ maps all pairs A, σ as above in Tp: $[\![\![A]\!]\!]_\sigma \in \mathrm{Tp}$, and: $[\![\![.]\!]\!]_{(.)}$ acts as variable interpretation, that is, $[\![\![\alpha_i]\!]\!]_\sigma = \sigma(\alpha_i)$ for $i = 1, \ldots, n$.
(2) $[\![\![.]\!]\!]$ is an *interpretation for the types and terms* in $\mathcal{M}$ if it is the union of two maps $[\![\![.]\!]\!]_{(.)}$, $[\![\![.]\!]\!]_{(.,.)}$ such that:
  (a) $[\![\![.]\!]\!]_{(.)}$ is an interpretation for the types of $\mathcal{F}$,
  (b) whenever $\Delta = \{x_1 : A_1, \ldots, x_m : A_m\}$ is a set of term variables of $\mathcal{F}$, $\tau(x_j) \in \mathtt{E}([\![\![A_j]\!]\!]_\sigma)$ for $j = 1, \ldots, m$, $\mathtt{FV}(A) \subseteq \Gamma$, and $\Delta \vdash t : A$ in $\mathcal{F}$, then $[\![\![t]\!]\!]_{\sigma,\tau} \in \mathtt{E}([\![\![A]\!]\!]_\sigma)$, and $[\![\![x_j]\!]\!]_{\sigma,\tau} = \tau(x_j)$ for $j = 1, \ldots, m$.

  We call σ, τ a type and a term substitution (on Γ, Δ and in Tp, $\mathtt{E}(.)$, $[\![\![.]\!]\!]$).
(3) A *type structure* is any list $\langle \mathrm{Tp}, \mathtt{E}(.), [\![\![.]\!]\!]\rangle$ with the properties above.

From frames and interpretation maps we may define models of $\mathcal{F}$.

Definition 4.3 ($\mathcal{F}$*-Model*)**.** A *model of* $\mathcal{F}$ is a frame for $\mathcal{F}$ equipped with an interpretation map $[\![.]\!]$ for types and terms of $\mathcal{F}$, such that the interpretation of types of $\mathcal{F}$ satisfies the following two conditions for $\to, \forall$:

(1) $[\![A \to B]\!]_\sigma = ([\![A]\!]_\sigma \Rightarrow [\![B]\!]_\sigma)$,
(2) $[\![\forall\alpha.A]\!]_\sigma = \Pi(F)$ for some $F \in \mathrm{Pred}$ such that $F(a) = [\![A]\!]_{\sigma,a/\alpha}$ for all $a \in \mathrm{Tp}$,

and the interpretation of terms of $\mathcal{F}$ satisfies the following four conditions for abstraction and application:

(1) If $t : A \to B$, $u : A$, $a = [\![A]\!]_\sigma$, and $b = [\![B]\!]_\sigma$, then $[\![t(u)]\!]_{\sigma,\tau} = \Phi_{a,b}([\![t]\!]_{\sigma,\tau}, [\![u]\!]_{\sigma,\tau})$,
(2) If $\lambda x : A.t : A \to B$, $a = [\![A]\!]_\sigma$, $b = [\![B]\!]_\sigma$, and $\phi(c) = [\![t]\!]_{\sigma,(\tau,c/x)}$ for all $c \in [\![A]\!]_\sigma$, then $[\![\lambda x : A.t]\!]_{\sigma,\tau} = \Phi^{-1}_{a,b}(\phi)$,
(3) if $t : \forall\alpha.A$ and $F(a) = [\![A]\!]_{\sigma,a/\alpha}$ for all $a \in \mathrm{Tp}$, then $[\![t(B)]\!]_{\sigma,\tau} = \Phi_F([\![t]\!]_{\sigma,\tau}, [\![B]\!]_\sigma)$,
(4) If $\lambda\alpha.t : \forall\alpha.A$, $F(a) = [\![A]\!]_{\sigma,a/\alpha}$ for all $a \in \mathrm{Tp}$, and $\psi(a) = [\![t]\!]_{(\sigma,a/\alpha),\tau}$ for all $a \in \mathrm{Tp}$, then $[\![\lambda\alpha.t]\!]_{\sigma,\tau} = \Phi^{-1}_F(\psi)$.

A basic property we will prove for all models is commutation of the interpretation map with substitution and with $\beta\eta$. Commutation is stated as follows:

Definition 4.4 (*Commutation with Substitution and* $\beta\eta$*-Convertibility*)**.** Assume A is a type with free type variables in Γ, α, and T is a type with free type variables in Γ. Assume t is a term in the context $\Delta, x : A$ with free type variables in Γ, α, and u, v are terms in the context Δ with free type variables in Γ. Let σ, τ be a type and a term substitution on Γ, Δ in $\mathrm{Tp}, \mathrm{E}(.), [\![.]\!]$. We say that

(1) $[\![.]\!]$ *commutes with substitution* if for all A, T, t, u, σ, τ we have $[\![A[T/\alpha]]\!]_\sigma = [\![A]\!]_{\sigma,[\![T]\!]_\sigma/\alpha}$ and $[\![t[T/\alpha, u/x]]\!]_{\sigma,\tau} = [\![t]\!]_{(\sigma,[\![T]\!]_\sigma/\alpha),(\tau,[\![u]\!]_{\sigma,\tau}/x)}$.
(2) $[\![.]\!]$ is *commutes with* $\beta\eta$ if for all u, v, σ, τ, $u =_{\beta\eta} v$ implies $[\![u]\!]_{\sigma,\tau} = [\![v]\!]_{\sigma,\tau}$.

In Section 8 we will prove that a type structure $\langle \mathrm{Tp}, \mathrm{E}(.), [\![.]\!]\rangle$ can be extended to a model if and only if: $[\![.]\!]$ commutes with substitutions and $\beta\eta$-convertibility, and it satisfies a property called "Weak Extensionality" by H. Barendregt. In fact, we introduced type structures in order to have an alternative characterization of models in some proofs. For any frame $\mathcal{M}$, the interpretation map $[\![.]\!]$ on terms and types of $\mathcal{F}$ is uniquely determined by the equations in Definition 4.3. This is proved by induction over the types and terms of $\mathcal{F}$. In the case of the interpretation of $\lambda x : A.t$, $\lambda\alpha.t$ we use the fact that $\Phi_{a,b}$, Φ_F are assumed to be injective, and therefore $\Phi^{-1}_{a,b}$, Φ^{-1}_F are uniquely determined when they are defined. All other cases are immediate. We write $[\![t]\!]_{\mathcal{M},\sigma}$ for the interpretation of a term t in the model $\mathcal{M}$, w.r.t. the variable assignment σ. We write $[\![t]\!]_{\mathcal{M}} = [\![t]\!]_{\mathcal{M},\emptyset}$ for the interpretation of a closed term t having a closed type.

If A is a closed type, $t, u : A$ are closed terms of $\mathcal{F}$, and $\mathcal{M}$ is a model, we write $t =_{\mathcal{M}} u$ if $[\![t]\!]_{\mathcal{M}} = [\![u]\!]_{\mathcal{M}}$. It is straightforward to show that the equivalence relation $=_{\mathcal{M}}$ defines an equational theory for $\mathcal{F}$ including $\beta\eta$. A model $\mathcal{M}$ of $\mathcal{F}$ is inconsistent if any two elements of the same type of $\mathcal{M}$ are equal. A model is consistent if it is not inconsistent. Among the consistent models of $\mathcal{F}$ we quote: the term model, consisting of all *open* types and terms of $\mathcal{F}$, the observational model, consisting of all *closed* types and terms of $\mathcal{F}$ modulo observational equality (Definition 3.2), and Longo-Moggi PER models [14]. We may now formally define the notion of $\beta\eta$-completeness.

Definition 4.5 ($\beta\eta$*-Completeness for Types and Models of* $\mathcal{F}$)**.** Assume A, B are closed types of $\mathcal{F}$.

(1) eq_A is an *internal equality* for A if $\mathrm{eq}_A : A \to A \to \mathrm{Bool}$ is a closed term of $\mathcal{F}$ and for any two closed terms $t, u : A$ of $\mathcal{F}$ we have $t =_{\beta\eta} u$ if and only if $\mathrm{eq}_A(t, u) =_{\beta\eta} \mathrm{True}$.
(2) $f : A \to B$ is an *internal injection* if f is a closed term of $\mathcal{F}$ and for all closed terms $t, u : A$ of $\mathcal{F}$: if $f(t) =_{\beta\eta} f(u)$ then $t =_{\beta\eta} u$.
(3) A closed type A of $\mathcal{F}$ is $\beta\eta$*-complete* in a model $\mathcal{M}$ for $\mathcal{F}$ if for all closed terms $t, u : A$ of $\mathcal{F}$: $t =_{\mathcal{M}} u$ if and only if $t =_{\beta\eta} u$.
(4) A closed type A of $\mathcal{F}$ is $\beta\eta$*-complete* if A is $\beta\eta$-complete in all consistent models of $\mathcal{F}$.
(5) A model $\mathcal{M}$ is $\beta\eta$*-complete* if all closed types of $\mathcal{F}$ are $\beta\eta$-complete in $\mathcal{M}$.

All data types (like Bool, Nat) of $\mathcal{F}$ have an internal equality and are $\beta\eta$-complete (that is, $\beta\eta$-complete in all consistent models of $\mathcal{F}$). In this section we only prove this result for the data types Bool, Nat. In the next section, we prove $\beta\eta$-completeness for a data type internalizing de Bruijn coding with levels of variables.

Lemma 4.6. *Assume* A, B *are closed types of* $\mathcal{F}$.

(1) *Nat has some internal equality* eq_{Nat} *in* $\mathcal{F}$.
(2) *True* $\neq_{\mathcal{M}}$ *False, for any consistent model* $\mathcal{M}$ *of* $\mathcal{F}$.
(3) (Statman's Lemma) *Bool is* $\beta\eta$*-complete.*
(4) *If* A *has some internal equality* eq_A*, then* A *is* $\beta\eta$*-complete.*
(5) *Nat is* $\beta\eta$*-complete.*
(6) *If* B *is* $\beta\eta$*-complete and* $f : A \to B$ *is an internal injection, then* A *is* $\beta\eta$*-complete.*

Proof. (1) We define $\mathrm{eq}_{\mathrm{Nat}}$ using a double iterator on Nat w.r.t. the constructors $\underline{0} : \mathrm{Nat}, S : \mathrm{Nat} \to \mathrm{Nat}$. This iterator is definable in $\mathcal{F}$. We set
(a) $\mathrm{eq}_{\mathrm{Nat}}(\underline{0}, \underline{0}) = \mathrm{True}$
(b) $\mathrm{eq}_{\mathrm{Nat}}(S(n), S(n')) = \mathrm{eq}_{\mathrm{Nat}}(n, n')$
(c) In all remaining cases we set $\mathrm{eq}_{\mathrm{Nat}}(n, n') = \mathrm{False}$
By induction on the length of the $\beta\eta$-long normal forms of t, u, we can prove that $\mathrm{eq}_{\mathrm{Nat}}(t, u) =_{\beta\eta} \mathrm{True}$ if and only if $t =_{\beta\eta} u$.
(2) Assume $\mathrm{True} =_{\mathcal{M}} \mathrm{False}$, that is, $[\![\mathrm{True}]\!]_{\mathcal{M}} = [\![\mathrm{False}]\!]_{\mathcal{M}}$. Then for all types θ of $\mathcal{M}$ and all $a, b \in \mathrm{E}(\theta)$ we have $[\![\mathrm{True}]\!]_{\mathcal{M}}(\theta, a, b) = [\![\mathrm{False}]\!]_{\mathcal{M}}(\theta, a, b)$, and by β-rule, $a = b$, against the consistency of $\mathcal{M}$.
(3) Assume $\mathcal{M}$ is any consistent model, $t, u : \mathrm{Bool}$ are closed terms of $\mathcal{F}$, and $t =_{\mathcal{M}} u$. If either the $\beta\eta$-long normal forms of t, u are both True, or are both False, then $t =_{\beta\eta} u$. If one is True while the other is False, then $\mathrm{True} =_{\mathcal{M}} \mathrm{False}$, against (2).
(4) Assume $t =_{\mathcal{M}} u$. Then $\mathrm{eq}_A(t, u) =_{\mathcal{M}} \mathrm{eq}_A(t, t) =_{\mathcal{M}} \mathrm{True}$. By (3) above we deduce $\mathrm{eq}_A(t, u) =_{\beta\eta} \mathrm{True}$. By definition of internal equality we conclude $t =_{\beta\eta} u$.
(5) By (1), Nat has some internal equality $\mathrm{eq}_{\mathrm{Nat}}$. By (4), Nat is $\beta\eta$-complete.
(6) Assume $\mathcal{M}$ is any consistent model, $t, u : A$ are two closed terms of $\mathcal{F}$, and $t =_{\mathcal{M}} u$. Then $f(t) =_{\mathcal{M}} f(u)$ and $f(t), f(u) : B$. By $\beta\eta$-completeness of B we deduce $f(t) =_{\beta\eta} f(u)$, and by f internal injection we conclude $t =_{\beta\eta} u$, as wished. □

There are consistent models which are $\beta\eta$-complete (with respect to all types, and not only with respect to data types). The term model is trivially $\beta\eta$-complete, and there are also non-trivial examples [8]. The observational model and most PER-models are consistent but not $\beta\eta$-complete: in these models, the type Nat → Nat is *not* $\beta\eta$-complete.

Lemma 4.7. *Assume $\mathcal{O}$ is the observational model of $\mathcal{F}$ and $S_l, S_r : \mathrm{Nat} \to \mathrm{Nat}$ are the left- and right-successor (Definition 3.1).*

(1) *$S_l \neq_{|\beta\eta|} S_r$ and $S_l =_{\mathcal{O}} S_r$.*
(2) *$\mathcal{O}$ is not $\beta\eta$-complete for the type Nat → Nat.*

Proof. (1) $|S_l| = \lambda\mathtt{x}.\lambda\mathtt{f}.\lambda\mathtt{a}.\mathtt{x}(\mathtt{f}, \mathtt{f}(\mathtt{a}))$ and $|S_r| = \lambda\mathtt{x}.\lambda\mathtt{f}.\lambda\mathtt{a}.\mathtt{f}(\mathtt{x}(\mathtt{f}, \mathtt{a}))$ are not $\beta\eta$-convertible, therefore $S_l \neq_{|\beta\eta|} S_r$. We have still to prove $[\![S_l]\!]_{\mathcal{O}} = [\![S_r]\!]_{\mathcal{O}}$. In all models $\mathcal{M}$, if A is a closed type and $f, g : (A \to A)$ are closed terms of $\mathcal{F}$, then $[\![f]\!]_{\mathcal{M}}, [\![g]\!]_{\mathcal{M}}$ are identified with maps over $[\![A]\!]_{\mathcal{M}}$. If $[\![f]\!]_{\mathcal{M}}(a) = [\![g]\!]_{\mathcal{M}}(a)$ for all elements a of $[\![A]\!]_{\mathcal{M}}$, then $[\![f]\!]_{\mathcal{M}} = [\![g]\!]_{\mathcal{M}}$. $\mathcal{O}$ is a model of $\mathcal{F}$ by [16]: therefore, in order to prove $[\![S_l]\!]_{\mathcal{O}} = [\![S_r]\!]_{\mathcal{O}}$ we assume that t is an element of $[\![\mathrm{Nat}]\!]_{\mathcal{O}}$ and we prove $[\![S_l]\!]_{\mathcal{O}}(t) = [\![S_r]\!]_{\mathcal{O}}(t)$. By definition of observational model, t is a *closed* term of $\mathcal{F}$ of type Nat. Therefore the $\beta\eta$-long normal form of t is $\underline{n} = \lambda\alpha.\lambda f : \alpha \to \alpha.\lambda a : \alpha.f^n(a)$ for some $n \in N$. We deduce $S_l(t) =_{\beta\eta} S_l(\underline{n}) =_{\beta\eta} \underline{n+1} =_{\beta\eta} S_r(\underline{n}) =_{\beta\eta} S_r(t)$. Hence $S_l(t), S_r(t)$ are observationally equal, i.e., $[\![S_l]\!]_{\mathcal{O}}(t) = [\![S_r]\!]_{\mathcal{O}}(t)$.
(2) By (1) and $\beta\eta \subset |\beta\eta|$. □

We may now discuss what are the minimal requests we should have over any type T coding untyped λ-terms in $\mathcal{F}$. First of all, the coding should be *injective*, as we said in the previous section: any two codings should be $\beta\eta$-convertible if and only if they represent the same syntactic tree, that is, if and only if the two untyped λ-terms are equal under α-conversion. Besides, T should be $\beta\eta$-complete, like any other type of data. That is, equality over the codings of two terms should be independent of the equational theory we choose for $\mathcal{F}$, as it is the case for the equality between elements of any data type in $\mathcal{F}$. This leads to the following definition:

Definition 4.8 (α-*Completeness*)**.** Assume T is any type of $\mathcal{F}$ equipped with a coding $[.]$ of untyped λ-terms. The coding $[.]$ is α-*complete* if $[.]$ is an injective coding and T is a $\beta\eta$-complete type.

If a coding is α-complete and the codings of two untyped λ-terms are equal in some model of $\mathcal{F}$, then the two untyped terms are α-convertible.

5. There is no normalizer, compiler, nor decompiler for $\mathcal{F}$ inside $\mathcal{F}$

In this section, by adapting Turing's diagonalization argument, we prove that for any α-complete coding there is no normalizer nor compiler for all terms in $\mathcal{F}$ written in $\mathcal{F}$ itself, and that the coding in Tm^c and de Bruijn coding are, indeed, α-complete. By defining an internal equality for Tm^c we also prove that there is no decompiler for all terms in $\mathcal{F}$ written in $\mathcal{F}$ itself. The result for normalizers generalizes to any normal coding having an "internalization" K of the coding map and extending the terms.

We first formally define normalizers, compilers, and decompilers for $\mathcal{F}$ inside $\mathcal{F}$, w.r.t. any coding $[\![.]\!]$.

Definition 5.1. Let $[\![.]\!]^c$ be any closed coding of $\mathcal{F}$ in some closed type T.

(1) A normalizer for $\mathcal{F}$ in $\mathcal{F}$ is a family of closed terms $\mathrm{ev}_A : T \to T$ of $\mathcal{F}$, for each closed type A of $\mathcal{F}$, such that for all closed terms $t : A$ of $\mathcal{F}$, with its $\beta\eta$-long normal form $t' : A$, we have $\mathrm{ev}_A([\![t]\!]^c) =_{\beta\eta} [\![t']\!]^c$.
(2) A compiler for $\mathcal{F}$ in $\mathcal{F}$ is a family $g_A : T \to A$ of closed terms of $\mathcal{F}$, for each closed type A of $\mathcal{F}$, such that for all closed terms $t : A$ of $\mathcal{F}$ we have $g_A([\![t]\!]^c) =_{\beta\eta} t$.

(3) A decompiler for $\mathcal{F}$ in $\mathcal{F}$ is a family $f_A : A \to T$ of closed terms of $\mathcal{F}$, for each closed type A of $\mathcal{F}$, such that for all closed terms $t : A$ of $\mathcal{F}$ we have $f_A(t) =_{\beta\eta} [\![t']\!]^c$ for some $t' =_{\beta\eta} t$.

The non-existence of a normalizer is provable for any normal coding having a map K internalizing the coding map in $\mathcal{F}$ and extending the terms (Definition 3.4). Under these assumptions, we prove that the existence of a normalizer implies that the coding map has some fixed point t, contradicting the fact that the coding map should extend t.

The non-existence of a compiler for $\mathcal{F}$ in $\mathcal{F}$ is immediate if we consider that the range T of the compiler must include some closed term, while the target type A may be the type Void, having no closed term. However, we can prove the non-existence of a compiler for $\mathcal{F}$ in $\mathcal{F}$ even if we restrict the family g_A to any inhabited type A of $\mathcal{F}$. In this case we use Turing's diagonalization argument in order to prove that the successor map has a fixed point, which leads to contradiction.

Theorem 5.2 (*Normalizer and Compiler for $\mathcal{F}$ in $\mathcal{F}$*)**.** *Assume $[\![.]\!]^c$ is any closed coding in T.*

(1) *If $[\![.]\!]^c$ is normal, has an internalization map K and extends the terms, then there is no normalizer for $\mathcal{F}$ in $\mathcal{F}$ w.r.t. $[\![.]\!]^c$.*
(2) *There is no normalizer for $\mathcal{F}$ in $\mathcal{F}$ w.r.t. the closed coding in Tm^c.*
(3) *There is no compiler for $\mathcal{F}$ in $\mathcal{F}$, w.r.t. for any coding, even if we restrict the compiler g_A to the inhabited types A of $\mathcal{F}$.*

Proof. (1) Assume ev_A is a normalizer for $\mathcal{F}$ in $\mathcal{F}$. Set $A = T$. By assumption, there is a map $K : T \to T$ internalizing $[\![.]\!]^c$. We define a closed term $f : T \to T$ of $\mathcal{F}$ by $f(x) = \mathrm{ev}_A(\mathrm{ap}^c(x, K(x))) : T$, where $x : T$. Assume $g : T$ is the $\beta\eta$-long normal form of $f([\![f]\!]^c)$. Then by definition we have: $g =_{\beta\eta} f([\![f]\!]^c) =_{\beta\eta} \mathrm{ev}_A(\mathrm{ap}^c([\![f]\!]^c, K([\![f]\!]^c))) =_{\beta\eta}$ (since K internalizes $[\![.]\!]^c$) $\mathrm{ev}_A(\mathrm{ap}^c([\![f]\!]^c, [\![[\![f]\!]^c]\!]^c)) =_{\beta\eta} \mathrm{ev}_A([\![f([\![f]\!]^c)]\!]^c) =_{\beta\eta}$ (since ev_A is a normalizer and g is the $\beta\eta$-long normal form of $f([\![f]\!]^c)$) $[\![g]\!]^c$. The terms g, $[\![g]\!]^c$ are $\beta\eta$-long normal by assumption, therefore $g =_{\beta\eta} [\![g]\!]^c$ implies $g =_\alpha [\![g]\!]^c$, and hence $|g| =_\alpha |[\![g]\!]^c| = |[|g|]^c|$, contradicting the assumption that $[.]^c$ extends the length of the term $|g|$.
(2) By Lemma 3.6 and (1) above.
(3) Assume $g_A : T \to A$ is a compiler for $\mathcal{F}$, written inside $\mathcal{F}$, for any inhabited type A. Let $A = (T \to \mathrm{Nat})$, an inhabited type, and S_r be the right-successor (Definition 3.1). Define a closed term $h : A$ of $\mathcal{F}$ by $h(x) = S_r(g_A(x)(x)) : \mathrm{Nat}$, where $x : T$. By definition we have $h([\![h]\!]^c) =_{\beta\eta} S_r(g_A([\![h]\!]^c)([\![h]\!]^c)) =_{\beta\eta} S_r(h([\![h]\!]^c))$. The $\beta\eta$-long normal form of $h([\![h]\!]^c)$ is $\underline{n}$ in Nat for some $n \in N$, while the $\beta\eta$-long normal form of $S_r(h([\![h]\!]^c))$ is $\underline{n+1}$, which is a contradiction. In this proof we used no specific property of $[\![.]\!]^c$. □

We will prove that there is no decompiler for $\mathcal{F}$ for any α-complete coding in $\mathcal{F}$, and that Tm^c is, indeed, an α-complete coding. We first define an internal equality $\mathrm{eq}_{\mathrm{Tm}^c}$, deciding $\beta\eta$-equality for Tm^c (hence α-equality for the coding of untyped λ-terms). We cannot define the internal equality as we do for data types, because Tm^c is not a data type. As a preliminary step, we translate the elements of Tm^c into some suitable data type dB^c by some internal injection of $\mathcal{F}$, and then we use the internal equality of dB^c to prove that the coding in Tm^c is α-complete. The type dB^c internalizes de Bruijn coding with level of variables for untyped λ-terms.

In this coding, a λ-abstraction nested within m other λ-abstractions always binds the variable of name $\mathtt{m}$ (it binds the variable of name $n + m$ if the λ-term lives in the context $\{\mathtt{x}_0, \ldots, \mathtt{x}_{n-1}\}$). For instance, the λ-term $\mathtt{t} = \lambda\mathtt{x}.(\mathtt{x}(\lambda\mathtt{y}.(\mathtt{xy})))$ is coded by $\lambda 0.(0(\lambda 1.(01)))$. Remark that the same term $\mathtt{t}$, in the coding with deBruijn indexes [4,13], would be coded $\lambda 0.(0(\lambda 0.(10)))$instead.

DeBruijn level coding is an example of α-complete coding. In dB^c we explicitly represent *variable names* with a type Var isomorphic to Nat, something we do *not* have in Tm. We fix two type variables Var, dB and the context Γ_{dB} with the variables representing de Bruijn coding: $0 : \mathrm{Var}$, $S : \mathrm{Var} \to \mathrm{Var}$, $\mathrm{var} : \mathrm{Var} \to \mathrm{dB}$, $\mathrm{ap} : \mathrm{dB} \to \mathrm{dB} \to \mathrm{dB}$, $\mathrm{lam} : \mathrm{Var} \to \mathrm{dB} \to \mathrm{dB}$.

Definition 5.3. We write $\lambda\Gamma_{\mathrm{dB}}$ as an abbreviation for the λ-abstractions: $\lambda\mathrm{Var}.\lambda\mathrm{dB}.\lambda 0 : \mathrm{Var}.\lambda S : \mathrm{Var} \to \mathrm{Var}.\lambda\mathrm{var} : (\mathrm{Var} \to \mathrm{dB}).\lambda\mathrm{ap} : (\mathrm{dB} \to \mathrm{dB} \to \mathrm{dB}).\lambda\mathrm{lam} : (\mathrm{Var} \to \mathrm{dB} \to \mathrm{dB})$ and $x(\Gamma_{\mathrm{dB}})$ for $x(\mathrm{Var}, \mathrm{dB}, 0, S, \mathrm{var}, \mathrm{ap}, \mathrm{lam})$. Then we set:

(1) $\mathrm{dB}^c = \forall\mathrm{Var}.\forall\mathrm{dB}.\mathrm{Var} \to (\mathrm{Var} \to \mathrm{Var}) \to (\mathrm{Var} \to \mathrm{dB}) \to (\mathrm{dB} \to \mathrm{dB} \to \mathrm{dB}) \to (\mathrm{Var} \to \mathrm{dB} \to \mathrm{dB}) \to \mathrm{dB}$.
(2) We define the closed terms of $\mathcal{F}$ representing the constructors of dB^c:
  (a) $\mathrm{var}^c : \mathrm{Nat} \to \mathrm{dB}^c$ by $\mathrm{var}^c(x) = \lambda\Gamma_{\mathrm{dB}}.\mathrm{var}(x(\mathit{Var}, 0, S))$.
  (b) $\mathrm{ap}^c : \mathrm{dB}^c \to \mathrm{dB}^c \to \mathrm{dB}^c$ by $\mathrm{ap}^c(x, y) = \lambda\Gamma_{\mathrm{dB}}.\mathrm{ap}(x(\Gamma_{\mathrm{dB}}), y(\Gamma_{\mathrm{dB}}))$.
  (c) $\mathrm{lam}^c : (\mathrm{Nat} \to \mathrm{dB}^c \to \mathrm{dB}^c)$ by $\mathrm{lam}^c(n, y) = \lambda\Gamma_{\mathrm{dB}}.\mathrm{lam}(n(\mathit{Var}, 0, S), y(\Gamma_{\mathrm{dB}}))$.

If we take A such that $\mathrm{dB}^c = \forall\mathrm{Var}.\forall\mathrm{dB}.A$, then there is no $\forall$ in A and $\deg(A) = 2$, and therefore dB^c is a data type. We can prove that dB^c is $\beta\eta$-complete (in fact, all data types are, but we only need $\beta\eta$-completeness of Bool, Nat and dB^c).

Lemma 5.4. *dB^c is $\beta\eta$-complete*

Proof. We define an internal equality eq_D for $D = \mathrm{dB}^c$ using a double iterator on dB^c w.r.t. the constructors $\mathrm{var}^c : \mathrm{Nat} \to \mathrm{dB}^c$, $\mathrm{lam}^c : \mathrm{Nat}, \mathrm{dB}^c \to \mathrm{dB}^c$ and $\mathrm{ap}^c : \mathrm{dB}^c, \mathrm{dB}^c \to \mathrm{dB}^c$. This iterator is definable in $\mathcal{F}$. Assume $\wedge : \mathrm{Bool}, \mathrm{Bool} \to \mathrm{Bool}$ is a closed term of $\mathcal{F}$ representing boolean conjunction. Then for $D = \mathrm{dB}^c$ we set:

(1) $\mathrm{eq}_D(\mathrm{var}^c(n), \mathrm{var}^c(n')) = \mathrm{eq}_{\mathrm{Nat}}(n, n')$
(2) $\mathrm{eq}_D(\mathrm{lam}^c(v, t), \mathrm{lam}^c(v', t')) = \mathrm{eq}_{\mathrm{Nat}}(v, v') \wedge \mathrm{eq}_D(t, t')$

(3) $\mathrm{eq}_D(\mathrm{ap}^c(t,t'),\mathrm{ap}^c(u,u')) = \mathrm{eq}_D(t,t') \wedge \mathrm{eq}_D(u,u')$
(4) In all remaining cases we set $\mathrm{eq}_D(t,t') = \mathrm{False}$

By induction on the $\beta\eta$-long normal forms of t, u we can prove that if $D = \mathrm{dB}^c$, then $\mathrm{eq}_D(t,u) =_{\beta\eta} \mathrm{True}$ if and only if $t =_{\beta\eta} u$. □

The code of a untyped λ-term in dB (or in dB^c) is uniquely given for any untyped context (list of untyped variables) for the λ-term. Assume that u is the de Bruijn code of an untyped λ-term with free variables in $\mathtt{x}_0, \ldots, \mathtt{x}_{n-1}$, and $\mathrm{lam}(x,t)$ is a subterm of u, within nested m abstractions of u. We call m the "level" of the variable x. We require that $x = x_{n+m}$, that is, that the variable bound by any lam nested within $m-1$ more lambda's is the variable number $n+m$.

Definition 5.5. Let $n \in N$, and $v(n) = S^n(0)$ in the context Γ_{dB}. The coding $\ulcorner \mathtt{t} \urcorner_n$ of a term $\mathtt{t} \in \Lambda$ in Γ_{dB} is recursively defined as follows:

(1) $\ulcorner \mathtt{x}_i \urcorner_n = \mathrm{var}(v(i))$.
(2) $\ulcorner \lambda \mathtt{x}.\mathtt{t} \urcorner_n = \mathrm{lam}(\mathrm{var}(v(n)), \ulcorner \mathtt{t}[\mathtt{x}_n/\mathtt{x}] \urcorner_{n+1})$.
(3) $\ulcorner \mathtt{tu} \urcorner_n = \mathrm{ap}(\ulcorner \mathtt{t} \urcorner_n, \ulcorner \mathtt{u} \urcorner_n)$.

We define $\ulcorner \mathtt{t} \urcorner^c_n = \lambda \Gamma_{\mathrm{dB}}.\ulcorner \mathtt{t} \urcorner_n : \mathrm{dB}^c$. If $\mathtt{t}$ is closed then $\ulcorner \mathtt{t} \urcorner^c_0$ is closed. We first prove that the de Bruijn level coding of $\mathtt{t} \in \Lambda$ in $\mathcal{F}$ is injective, that is, an injection up to α-conversion, and that there is an internal injection from Tm^c to dB^c, sending a representation $[\mathtt{t}]^c$ of any *closed* $\mathtt{t} \in \Lambda$ to a representation $\ulcorner \mathtt{t} \urcorner^c_0$ of the same $\mathtt{t}$ in dB^c.

An untyped λ-term $\mathtt{t}$ is in fact represented by a family of de Bruijn codes, depending on a parameter $n \in \mathrm{Nat}$, and representing the code of $\mathtt{t}$ with free variables in $\mathtt{x}_0, \ldots, \mathtt{x}_{n-1}$. Implicitly, the type of the de Bruijn coding of $\mathtt{t}$ should be $\mathrm{Nat} \to \mathrm{dB}^c$. We define a canonical substitution over the context Γ_{Tm}, replacing a generic coding of Λ by a concrete coding, de Bruijn level coding.

Definition 5.6 (*The de Bruijn Substitution*)**.** We set $\mathrm{DB} = \mathrm{Nat} \to \mathrm{dB}^c$. We define $\mathrm{Lam} : (\mathrm{DB} \to \mathrm{DB}) \to \mathrm{DB}$ and $\mathrm{Ap} : \mathrm{DB} \to \mathrm{DB} \to \mathrm{DB}$ with arguments $a, b : \mathrm{DB}$, $n : \mathrm{Nat}$, $f : \mathrm{DB} \to \mathrm{DB}$ by:

(1) $\mathrm{Lam}(f)(n) = \mathrm{lam}^c(n, f(\lambda\_ : \mathrm{Nat}.\mathrm{var}^c(n))(n+1)) : \mathrm{dB}^c$.
(2) $\mathrm{Ap}(a,b)(n) = \mathrm{ap}^c(a(n), b(n)) : \mathrm{dB}^c$.

We call $\delta = [\mathrm{DB}/\mathrm{Tm}, \mathrm{Lam}/\mathrm{lam}, \mathrm{Ap}/\mathrm{ap}]$ the de Bruijn substitution.

We prove that the de Bruijn substitution δ sends the coding of untyped λ-terms in Tm to their de Bruijn coding in DB. A program recovering the deBruijn syntax from the abstract syntax was already known for deBruijn indexes [3]. As far as we know, this is the first program working for deBruijn levels.

Lemma 5.7 (*de Bruijn Level Coding and Tm^c are α-Complete Codings*)**.** (1) *de Bruijn coding is injective: if* $\mathtt{t}, \mathtt{u} \in \Lambda$ *have free variables in* $\mathtt{x}_0, \ldots, \mathtt{x}_{n-1}$*, then* $\mathtt{t} =_\alpha \mathtt{u}$ *if and only if* $\ulcorner \mathtt{t} \urcorner_n =_{\beta\eta} \ulcorner \mathtt{u} \urcorner_n$.
(2) *de Bruijn level coding is an α-complete coding.*
(3) *There is a term $db : Tm^c \to dB^c$ of $\mathcal{F}$, such that for all closed terms t of Λ we have $db([\mathtt{t}]^c) =_{\beta\eta} \ulcorner \mathtt{t} \urcorner^c_0$.*
(4) *db is an internal injection from Tm^c to dB^c.*
(5) *The type Tm^c is an α-complete coding.*

Proof. (1) By induction on the pair $\mathtt{t}, \mathtt{u}$.
(2) By (1) and the fact that dB^c is $\beta\eta$-complete by 5.4.
(3) We first define a term $d : \mathrm{Tm}^c \to \mathrm{Nat} \to \mathrm{dB}^c$ of $\mathcal{F}$, such that for all closed terms $\mathtt{t} \in \Lambda$ we have $d([\mathtt{t}]^c, n) = \ulcorner \mathtt{t} \urcorner^c_n$. We will use continuation-passing-style programming. Assume $\mathtt{t} \in \Lambda$, $\mathrm{FV}(\mathtt{t}) \subseteq \{\mathtt{x}_0, \ldots, \mathtt{x}_{n-1}\}$ and $\sigma(\mathtt{x}_i) = a_i$ for all $i < n$. If $x : \mathrm{Tm}^c$, then by definition of Tm^c we have

$$x(\mathrm{DB}) : ((\mathrm{DB} \to \mathrm{DB}) \to \mathrm{DB}) \to (\mathrm{DB} \to \mathrm{DB} \to \mathrm{DB}) \to \mathrm{DB}.$$

We set $d(x) = x(\mathrm{DB})(\mathrm{Lam})(\mathrm{Ap}) : \mathrm{DB}$. Let $\rho(a_i) = \lambda\_ : \mathrm{Nat}.\ulcorner \mathtt{x}_i \urcorner^c_0$ for all $i < n$. Then we can show $d(\rho([t]^c_\sigma), n) =_{\beta\eta} \ulcorner t \urcorner^c_n$ as follows. By unfolding the definition of d we have $d(\rho([\mathtt{t}]^c_\sigma))(n) = (\rho([\mathtt{t}]^c_\sigma)(\mathrm{DB})(\mathrm{Lam})(\mathrm{Ap}))(n) = (\rho([\mathtt{t}]^c_\sigma(\mathrm{DB})(\mathrm{Lam})(\mathrm{Ap})))(n)$. By unfolding the definition of $[.]^c_{(.)}$, the latter is equal to $(\rho((\lambda\Gamma_{\mathrm{dB}}.[\mathtt{t}]_\sigma)(\mathrm{DB})(\mathrm{Lam})(\mathrm{Ap})))(n) = (\rho(\delta([\mathtt{t}]_\sigma)))(n)$ for any $n : \mathrm{Nat}$. By induction on all $\mathtt{t} \in \Lambda$ with free variables in $\mathtt{x}_0, \ldots, \mathtt{x}_{n-1}$, we can prove that $\rho(\delta([\mathtt{t}]_\sigma))(n) = \ulcorner \mathtt{t} \urcorner^c_n : \mathrm{dB}^c$. Eventually we set $\mathrm{db}(x) = d(x,0) : \mathrm{dB}^c$. We conclude $\mathrm{db}([\mathtt{t}]^c) = \ulcorner \mathtt{t} \urcorner^c_0 : \mathrm{dB}^c$ for all closed $\mathtt{t} \in \Lambda$.
(4) All closed terms $t, u : \mathrm{Tm}^c$ in $\mathcal{F}$ have normal forms $[\mathtt{f}]^c, [\mathtt{g}]^c$ for some closed $\mathtt{f}, \mathtt{g} \in \Lambda$ such that $[\mathtt{f}]^c =_{\beta\eta} t$ and $[\mathtt{g}]^c =_{\beta\eta} u$. Assume $\mathrm{db}(t) =_{\beta\eta} \mathrm{db}(u)$. Then $\ulcorner \mathtt{f} \urcorner^c_0 =_{\beta\eta} \mathrm{db}([\mathtt{f}]^c) =_{\beta\eta} \mathrm{db}(t) =_{\beta\eta} \mathrm{db}(u) =_{\beta\eta} \mathrm{db}([\mathtt{g}]^c) =_{\beta\eta} \ulcorner \mathtt{g} \urcorner^c_0$, and hence $\mathtt{f} =_\alpha \mathtt{g}$ in Λ. We conclude $t =_{\beta\eta} [\mathtt{f}]^c =_\alpha [\mathtt{g}]^c =_{\beta\eta} u$.
(5) By (4), there is an internal injection $\mathrm{db} : \mathrm{Tm}^c \to \mathrm{dB}^c$. Since dB^c is $\beta\eta$-complete by 5.4, by Lemma 4.6(6), Tm^c is $\beta\eta$-complete. Tm^c is injective because if $[\mathtt{t}]^c =_{\beta\eta} [\mathtt{u}]^c$, then by (3) we have $\ulcorner \mathtt{t} \urcorner^c_0 =_{\beta\eta} \mathrm{db}([\mathtt{t}]^c) =_{\beta\eta} \mathrm{db}([\mathtt{u}]^c) =_{\beta\eta} \ulcorner \mathtt{u} \urcorner^c_0$, hence $\mathtt{t} =_\alpha \mathtt{u}$ by (1) (injectivity of dB^c). □

We may now prove that $\mathcal{F}$ cannot decompile itself.

Theorem 5.8. (1) *There is no decompiler for $\mathcal{F}$ in $\mathcal{F}$ w.r.t. any α-complete coding $[\![.]\!]^c$.*
(2) *There is no decompiler for $\mathcal{F}$ in $\mathcal{F}$ w.r.t. Tm^c nor dB^c.*

Proof. (1) We assume having some decompiler f_A w.r.t. some α-complete coding $[\![.]\!]^c$. We will show a contradiction. By Lemma 4.7(1), we have $S_l =_{\mathcal{O}} S_r$. By definition of decompiler, we have some S_l' and S_r' such that $S_l =_{\beta\eta} S_l'$, $f_{\mathrm{Nat}\to\mathrm{Nat}}(S_l) =_{\beta\eta} [\![S_l']\!]^c$, $S_r =_{\beta\eta} S_r'$, and $f_{\mathrm{Nat}\to\mathrm{Nat}}(S_r) =_{\beta\eta} [\![S_r']\!]^c$. Since $f_{\mathrm{Nat}\to\mathrm{Nat}}(S_l) =_{\mathcal{O}} f_{\mathrm{Nat}\to\mathrm{Nat}}(S_r)$, we have $[\![S_l']\!]^c =_{\mathcal{O}} [\![S_r']\!]^c$. By α-completeness of the coding, we deduce first $[\![S_l']\!]^c =_{\beta\eta} [\![S_r']\!]^c$ (by $\beta\eta$-completeness), then $|S_l'| =_\alpha |S_r'|$ (by injectivity). Therefore we have $|S_l| =_{\beta\eta} |S_l'| =_\alpha |S_r'| =_{\beta\eta} |S_r|$, which contradicts with $S_l \neq_{|\beta\eta|} S_r$ in Lemma 4.7(1).
(2) Because Tm^c and dB^c are α-complete codings by Lemma 5.7. □

In the next section we will weaken the definition of decompiler. In Definition 6.1 we consider the possibility of decompiling not $\mathcal{F}$ itself, but some interpretation of $\mathcal{F}$ in $\mathcal{F}$. In this case the argument of Theorem 5.8 no longer applies, because what we recover by decompilation is not the original term of $\mathcal{F}$ but some interpretation of it, again in $\mathcal{F}$. We will consider the interpretation of $\mathcal{F}$ in $\mathcal{F}$ obtained by restricting all type quantifiers to the set of types which we call "connected" to the type Tm coding untyped λ-terms. Our main, Theorem 6.5, is that this interpretation of $\mathcal{F}$ in $\mathcal{F}$ may be decompiled by $\mathcal{F}$.

6. An interpretation $(.)^{*^c}$ of $\mathcal{F}$ into itself whose image is decompilable

In this section we define a particular interpretation $(.)^{*^c}$ of $\mathcal{F}$ inside $\mathcal{F}$ and a decompiler–normalizer, written in $\mathcal{F}$, for the terms of $\mathcal{F}^{*^c}$. In addition we prove that there is no compiler written in $\mathcal{F}$ for the terms of $\mathcal{F}^{*^c}$. In Theorem 5.2 we have already proved that there is no normalizer in $\mathcal{F}$ for the codes of terms of $\mathcal{F}$.

We formally define the notion of a decompiler–normalizer and a compiler for $\mathcal{F}^\circ$ in $\mathcal{F}$, for any interpretation $(.)^\circ$ of $\mathcal{F}$ in $\mathcal{F}$.

Definition 6.1. Assume $(.)^\circ$ is *any* interpretation of $\mathcal{F}$ in $\mathcal{F}$.

(1) A compiler for $\mathcal{F}^\circ$ in $\mathcal{F}$ is any family $g_A : \mathrm{Tm}^c \to A^\circ$ of closed terms of $\mathcal{F}$, for each closed type A of $\mathcal{F}$, such that for all closed terms $t : A$ of $\mathcal{F}$ we have $g_A([\![t]\!]^c) =_{\beta\eta} t^\circ$.
(2) A decompiler–normalizer for $\mathcal{F}^\circ$ in $\mathcal{F}$ is any family $f_A : A^\circ \to \mathrm{Tm}^c$ of closed terms of $\mathcal{F}$, for each closed type A of $\mathcal{F}$, such that for all closed terms $t : A$ of $\mathcal{F}$ we have $f_A(t^\circ) =_{\beta\eta} [\![t']\!]^c$ for the $\beta\eta$-long normal form t' of t.

The goal of this section is to define some interpretation $(.)^{*^c}$ of $\mathcal{F}$ into $\mathcal{F}$, and a decompiler–normalizer in $\mathcal{F}$ for the terms of $\mathcal{F}^{*^c}$. For some interpretation of $\mathcal{F}$ into $\mathcal{F}$ we could easily adapt the proof of Theorem 5.8, and show that the interpretation has no decompiler. This is the case, for instance, of any interpretation compatible with observational equality. Indeed, if $t =_{\mathcal{O}} u$ implies $t^{*^c} =_{\mathcal{O}} u^{*^c}$, for any term t, u of $\mathcal{F}$, then $S_l^{*^c} =_{\mathcal{O}} S_r^{*^c}$, and if we repeat the proof of 5.8 we derive a contradiction from the existence of a decompiler for $(.)^{*^c}$.

In this section, we will instead select an interpretation $(.)^{*^c}$ which does not fall in this class, and which may interpret two observationally equal terms like S_l, S_r by two terms which are *not* observationally equal.

Our interpretation is defined by restricting the domain of every quantifier over types. We use the map $(.)^*$: we first define A^* by induction over the types of $\mathcal{F}$, an induction external to $\mathcal{F}$.

Let A, B be two types of $\mathcal{F}$. A connection of A, B is a pair (f, g) of terms $f : A \to B$ and $g : B \to A$ of $\mathcal{F}$ in some context Γ. Two types are connected if they have a connection. For instance, $(\mathrm{id}_{\mathrm{Tm}}, \mathrm{id}_{\mathrm{Tm}})$ is a connection between Tm and Tm in the context Γ_{Tm}, while (lam, ap) is a connection between $\mathrm{Tm}\to\mathrm{Tm}$ and Tm, again in the context Γ_{Tm}. In $\mathcal{F}^*$, all type quantifiers $(\forall\alpha.A)^*$ will be bounded over the types α which are connected with Tm, i.e., for which two maps $f_\alpha : \alpha \to \mathrm{Tm}$ and $g_\alpha : \mathrm{Tm} \to \alpha$ are given in the context in which A^* lives. The restriction over the quantifiers is reminiscent of what happens in the normalization proof for $\mathcal{F}$: a quantifier has to be restricted to the set of candidates in order to interpret $\forall\alpha.A$ as a candidate.[1] We define a family of connections (f_A, g_A) between A^* and Tm, and then we prove that if all type variables in $\mathrm{FV}(A)$ are connected with Tm, then f_A is a decompiler. The idea of the pair (f, g) is taken from Friedman's proof for $\lambda\to$, but in the case of $\lambda\to$ there is the additional requirement that (f, g) is an embedding-retraction pair, i.e., $g\circ f = \mathrm{id}_\alpha$, which we do not ask for $\mathcal{F}$.

Definition 6.2. Assume A is a type with free type variables in $\{\alpha_1, \ldots, \alpha_n, \mathrm{Tm}\}$ of $\mathcal{F}$. We define A^* by induction on A. When $A = \forall\alpha.B$ by possibly renaming α we assume $\alpha \neq \mathrm{Tm}$.

(1) $\alpha^* = \alpha$, if α is a type variable.
(2) $(B \to C)^* = B^* \to C^*$.
(3) $(\forall\alpha.B)^* = \forall\alpha.(\alpha \to \mathrm{Tm}) \to (\mathrm{Tm} \to \alpha) \to B^*$ with $\alpha \neq \mathrm{Tm}$.

[1] We owe this remark to an anonymous referee.

If $\Gamma = x_1 : A_1, \ldots, x_m : A_m$ is any context of $\mathcal{F}$ then $\Gamma^* = x_1 : A_1^*, \ldots, x_m : A_m^*$.
We define A^{*c} as $\forall \mathrm{Tm}.((\mathrm{Tm} \to \mathrm{Tm}) \to \mathrm{Tm}) \to (\mathrm{Tm} \to \mathrm{Tm} \to \mathrm{Tm}) \to A^*$.

For all types A^*, we now define some connection between A^* and Tm in a suitable context. We explain the idea first.

(1) For any type variable $\alpha \neq \mathrm{Tm}$ we assume some connection (f_α, g_α) between α and Tm. If $\alpha = \mathrm{Tm}$ the connection is the identity.
(2) For arrow types we follow Joly [11]. We lift one connection between B^*, Tm and another connection between C^*, Tm to a connection between $B^* \to C^*$ and $\mathrm{Tm} \to \mathrm{Tm}$, and eventually we compose it with the connection (lam, ap) between $\mathrm{Tm} \to \mathrm{Tm}$ and Tm, obtaining a connection between $A^* = (B^* \to C^*)$ and Tm.
(3) The case of $(\forall \alpha.A)^*$ is the main original idea of this paper. Assume some connection between A^* and Tm is given. We may lift the map : $\mathrm{Tm} \to A^*$ to a map : $\mathrm{Tm} \to (\forall \alpha.A)^*$ by λ-abstraction. Conversely, we take any term : $(\forall \alpha.A)^*$, and we take α to be Tm, and the connection between α and Tm to be the connection $(\mathrm{id}_{\mathrm{Tm}}, \mathrm{id}_{\mathrm{Tm}})$ between Tm and Tm. We obtain a term in $A[\mathrm{Tm}/\alpha]^*$ and we apply some suitable instance of the map : $A^* \to \mathrm{Tm}$.

Definition 6.3 (*The Connection f_A, g_A Between A^* and Tm*)**.** For each type A of $\mathcal{F}$ with $\mathrm{FV}(A) \subseteq \{\alpha_1, \ldots, \alpha_n, \mathrm{Tm}\}$, by induction on the number of connectives in A, we define two terms $f_A : A^* \to \mathrm{Tm}$ and $g_A : \mathrm{Tm} \to A^*$ of $\mathcal{F}$, in the context $\{f_1 : \alpha_1 \to \mathrm{Tm}, \ldots, f_n : \alpha_n \to \mathrm{Tm}, g_1 : \mathrm{Tm} \to \alpha_1, \ldots, g_n : \mathrm{Tm} \to \alpha_n\} \cup \Gamma_{\mathrm{Tm}}$. Assume y is a fresh variable. When $A = \forall \alpha.B$ by possibly renaming α we assume $\alpha \neq \mathrm{Tm}$.

(1) $f_{\alpha_i} = f_i$ and $f_{\mathrm{Tm}} = \mathrm{id}_{\mathrm{Tm}}$.
$g_{\alpha_i} = g_i$ and $g_{\mathrm{Tm}} = \mathrm{id}_{\mathrm{Tm}}$.
(2) $f_{B \to C} = \lambda y : (B \to C)^*.\mathrm{lam}(f_C \circ y \circ g_B)$.
$g_{B \to C} = \lambda y : \mathrm{Tm}.g_C \circ \mathrm{ap}(y) \circ f_B$.

(3) $f_{\forall \alpha.B} = \lambda y : (\forall \alpha.B)^*.f_{B[\mathrm{Tm}/\alpha]}(y(\mathrm{Tm}, \mathrm{id}_{\mathrm{Tm}}, \mathrm{id}_{\mathrm{Tm}}))$. $g_{\forall \alpha.B} = \lambda y : \mathrm{Tm}.\lambda \alpha.\lambda f_\alpha : \alpha \to \mathrm{Tm}.\lambda g_\alpha : \mathrm{Tm} \to \alpha.g_B(y)$.

For any closed type A, we define $f_A^c = \lambda x : A^{*c}.\lambda \Gamma_{\mathrm{Tm}}.f_A(x\Gamma_{\mathrm{Tm}}) : A^{*c} \to \mathrm{Tm}^c$.

Implicitly, the interpretation of A in $\mathcal{F}^*$ is the triple (A^*, f_A, g_A), of the type A^* and some connection between A^* and Tm. In the definition of $g_{\forall \alpha.B}$, the term g_B has three more free variables: $\alpha, f_\alpha : \alpha \to \mathrm{Tm}$, and $g_\alpha : \mathrm{Tm} \to \alpha$.

We define the interpretation t^* of a term t, living in a context extended with variables of the form f_β, g_β, which should be fresh. This is not a serious restriction: by possibly renaming the term variables in t and the context of t, we may always assume that f_β, g_β are fresh variables, not occurring in t. Besides, our main result concerns closed terms t, which trivially satisfy the condition "f_β, g_β not free in t".

Definition 6.4. Assume $\Gamma \vdash t : A$ in $\mathcal{F}$, with $\Gamma = \{x_1 : A_1, \ldots, x_m : A_m\}$, and any variable in Γ is distinct from any f_α, g_α. We define $t^* : A^*$ in the context $\Gamma^* \cup \{f_1 : \alpha_1 \to \mathrm{Tm}, \ldots, f_n : \alpha_n \to \mathrm{Tm}, g_1 : \mathrm{Tm} \to \alpha_1, \ldots, g_n : \mathrm{Tm} \to \alpha_n\} \cup \Gamma_{\mathrm{Tm}}$. If $t = \lambda x : A. \ldots$ or $t = \lambda \alpha. \ldots$, by possibly renaming x and α we assume $x \neq f_\beta, g_\beta$ for any β and $\alpha \neq \mathrm{Tm}$.

(1) $x^* = x$.
(2) $(tu)^* = t^* u^*$.
(3) $(\lambda x : A.t)^* = \lambda x : A^*.t^*$, with $x \neq f_\beta, g_\beta$ for any β.
(4) $(\lambda \alpha.t)^* = \lambda \alpha.\lambda f_\alpha : \alpha \to \mathrm{Tm}.\lambda g_\alpha : \mathrm{Tm} \to \alpha.t^*$, with $\alpha \neq \mathrm{Tm}$.
(5) $(t(T))^* = t^*(T^*, f_T, g_T)$.

We define t^{*c} as $\lambda \Gamma_{\mathrm{Tm}}.t^*$.

In the definition of $(\lambda \alpha.t)^*$, the term t^* has two fresh free variables: $f_\alpha : (\alpha \to \mathrm{Tm})$, $g_\alpha : (\mathrm{Tm} \to \alpha)$.

For example, we have $f_A(\mathrm{id}^*) =_{\beta\eta} [\![\mathrm{id}]\!]$ for $A = \mathrm{Id}$ (the type of polymorphic identity $\mathrm{id} = \lambda \alpha.\lambda x : \alpha.x$). We unfold the definition of f_A. f_A first applies the clause for $\forall \alpha.\alpha \to \alpha$, and maps id^* to $\mathrm{id}^*(\mathrm{Tm}, \mathrm{id}_{\mathrm{Tm}}, \mathrm{id}_{\mathrm{Tm}}) =_{\beta\eta} \lambda x : \mathrm{Tm}.x$. Then f_A applies the clause for $\mathrm{Tm} \to \mathrm{Tm}$, sending $\lambda x : \mathrm{Tm}.x$ to $\mathrm{lam}(\lambda x : \mathrm{Tm}.x)$. The latter is the coding of the untyped λ-term λx.x, and therefore it is equal to $[\lambda \mathrm{x.x}]$, that is, to $[|\mathrm{id}|]$, or $[\![\mathrm{id}]\!]$. We conclude $f_A(\mathrm{id}^*) =_{\beta\eta} [\![\mathrm{id}]\!]$, as expected: f_A is a decompiler in the context Γ_{Tm} at least when applied to id^*.

The main theorem of the paper is:

Theorem 6.5 (*There is a Decompiler–Normalizer for $\mathcal{F}^{*c}$ in $\mathcal{F}$*)**.** *Assume u, v are terms of $\mathcal{F}$ and no variable of the form f_β, g_β is free in u, v. Assume $\emptyset \vdash t : A$ in $\mathcal{F}$, and that t' is the $\beta\eta$-long normal form of t.*

(1) $(.)^{*c}$ is an interpretation*: if $u =_{\beta\eta} v$, then $u^{*c} =_{\beta\eta} v^{*c}$.*
(2) f_A^c is a decompiler–normalizer*: $f_A^c(t^{*c}) =_{\beta\eta} [\![t']\!]^c : Tm^c$.*
(3) *There is no compiler for $\mathcal{F}^{*c}$ in $\mathcal{F}$ w.r.t. the coding in Tm^c.*

We devote the next section to the proof of Theorem 6.5.

7. Proof of main theorem

In this section we prove Theorem 6.5. During the proof, we extend the notion of a compilable (decompilable) term t in the context Γ_{Tm} to terms with free type and term variables, by asking that the "canonical substitution" of t is compilable (decompilable). The "canonical substitution" replaces any type variable α with Tm, any connection (f_α, g_α) with the pair of identities $(\mathrm{id}_{\mathrm{Tm}}, \mathrm{id}_{\mathrm{Tm}})$.

Definition 7.1 (*Canonical Substitution*)**.** Assume $t : A$ has the context $\Gamma = \{x_1 : A_1, \ldots, x_m : A_m\}$ with free type variables $\alpha_1, \ldots, \alpha_n$, and no variable of the form f_β, g_β is in Γ. Let $a_1, \ldots, a_m$ be fresh variables and assume the context $a_1 : \mathrm{Tm}, \ldots, a_m : \mathrm{Tm}$.

(1) σ, τ are the canonical substitutions for t if $\sigma(\alpha_i) = \mathrm{Tm}$, $\sigma(f_{\alpha_i}) = \sigma(g_{\alpha_i}) = \mathrm{id}_{\mathrm{Tm}}$ for all $i = 1, \ldots, n$, $\sigma(x_j) = \sigma(g_{A_j}(a_j))$ and $\tau(x_j) = a_j$ for all $j = 1, \ldots, m$.
(2) t is decompilable if $\sigma(f_A(t^*)) =_{\beta\eta} [\![t]\!]_\tau$ for some σ, τ canonical for t.
(3) t is compilable if $\sigma(t^*) =_{\beta\eta} \sigma(g_A([\![t]\!]_\tau))$ for some σ, τ canonical for t.

Recall that f_A, g_A are terms in the context Γ_{Tm} extended with $f_{\alpha_1}, g_{\alpha_1}, \ldots, f_{\alpha_n}, g_{\alpha_n}$. In the definition of "compilable" and "decompilable" we do not ask that we recover the open untyped λ-term underlying the term, but only that we may compile (decompile) the canonical instance of the term. If $A = \alpha_i$ for some i, then $\sigma(f_A) = \sigma(f_\alpha) = \mathrm{id}_{\mathrm{Tm}} = \sigma(g_\alpha) = \sigma(g_A)$, and therefore both "decompilable" and "compilable" unfold to $\sigma(t^*) = [\![t]\!]_\tau$. "compilable" and "decompilable" are equivalent in this case, and they both say that the canonical substitution σ maps t^* into the code $[\![t]\!]_\tau$ for t in the context Γ_{Tm}.

We now prove that $(.)^*, f_A, g_A$ commute with substitutions of term and type variables (the proof is long definition unfolding), and then a crucial statement, that formation of the map $g_{\forall\alpha.A}$ commutes with $\forall$-elimination. In order to state the commutative properties, let us recall that any substitution T/α in $\mathcal{F}$ corresponds to a substitution $\sigma = [T^*/\alpha, f_T/f_\alpha, g_T/g_\alpha]$ in $\mathcal{F}^*$.

Lemma 7.2 (*Substitution Lemma*)**.** *Assume $\alpha \neq Tm$ and $x \neq f_\beta, g_\beta$ for any β. Let $\sigma = [T^*/\alpha, f_T/f_\alpha, g_T/g_\alpha]$. Assume no f_β, g_β is free in the term t. The following commutative properties hold:*

(1) $A[T/\alpha]^* = A^*[T^*/\alpha]$,
(2) $t[u/x]^* = t^*[u^*/x]$,
(3) $f_{A[T/\alpha]} = \sigma(f_A)$,
$g_{A[T/\alpha]} = \sigma(g_A)$,
(4) $t[T/\alpha]^* = \sigma(t^*)$,
(5) $g_{\forall\alpha.A}$ *commutes with* $\forall$*-elimination:* $g_{\forall\alpha.A}(y)(T^*, f_T, g_T) =_{\beta\eta} g_{A[T/\alpha]}(y)$.

Proof. The proof is long but routine, by definition unfolding. We postpone it to the Appendix. □

As a consequence of Lemma 7.2(2), we can prove that $(.)^*$ is compatible with β. The compatibility of $(.)^*$ with η, instead, may be proved directly, by using η itself.

Lemma 7.3 (*Commutation of* $(.)^*$ *with* $\beta\eta$)**.** *Assume that no variable of the form f_β, g_β is free in t, u.*

(1) $((\lambda x : A.t)(u))^* =_{\beta\eta} t[u/x]^*$.
(2) $((\lambda\alpha.t)(T))^* =_{\beta\eta} t[T/\alpha]^*$.
(3) $(\lambda x : A.t(x))^* =_{\beta\eta} t^*$, *if* $x \notin \mathrm{FV}(t)$.
(4) $(\lambda\alpha.t(\alpha))^* =_{\beta\eta} t^*$, *if* $\alpha \notin \mathrm{FV}(t)$.
(5) *If* $t =_{\beta\eta} u$ *then* $t^* =_{\beta\eta} u^*$.

Proof. By possibly renaming x, α we may assume $x \neq f_\beta, g_\beta$ for any β and $\alpha \neq \mathrm{Tm}$, so that Lemma 7.2 applies.

(1) By definition, $((\lambda x : A.t)(u))^* =_{\beta\eta} t^*[u^*/x]$. Now we apply Lemma 7.2(2) to obtain $t[u/x]^*$.
(2) By definition, $((\lambda\alpha.t)(T))^* =_{\beta\eta} t^*[T^*/\alpha, f_T/f_\alpha, g_T/g_\alpha]$. Now we apply Lemma 7.2(4) to obtain $t[T/\alpha]^*$.
(3) If $x \notin \mathrm{FV}(t)$ then $x \notin \mathrm{FV}(t^*)$, because $x \neq f_\beta, g_\beta$ for any β and the extra term variables in t^* are different from x. By definition, $(\lambda x : A.t(x))^* =_{\beta\eta} \lambda x : A^*.t^*(x) =_{\beta\eta} t^*$ (by η-rule and $x \notin \mathrm{FV}(t^*)$).
(4) If $\alpha \notin \mathrm{FV}(t)$ then $\alpha \notin \mathrm{FV}(t^*)$, because $\alpha \neq \mathrm{Tm}$, which is the only extra type variable in t^*. From $\alpha \notin \mathrm{FV}(t)$ we deduce $f_\alpha, g_\alpha \notin \mathrm{FV}(t^*)$. By definition, $(\lambda\alpha.t(\alpha))^* =_{\beta\eta} \lambda\alpha.\lambda f_\alpha : \alpha \to \mathrm{Tm}.\lambda g_\alpha : \mathrm{Tm} \to \alpha.t^*(\alpha, f_\alpha, g_\alpha) =_{\beta\eta} t^*$ (by η-rule and $\alpha, f_\alpha, g_\alpha \notin \mathrm{FV}(t^*)$).
(5) By (1)–(4), compatibility of $(.)^*$, and term formation. □

In the next lemma we prove that $\beta\eta$-long normal terms in $\mathcal{F}$ are decompilable.

Lemma 7.4 (*Decompilation of* $\beta\eta$*-Long Normal Forms*)**.** *Assume that $t_1, \ldots, t_n, t, u$ are any terms with no variable f_β, g_β free, A, B, T are any types, and $\alpha \neq Tm$ is any type variable of $\mathcal{F}$.*

(1) *All term variables x are compilable.*
(2) *If $t : A \to B$ is compilable and $u : A$ is decompilable, then $tu : B$ is compilable.*

(3) *If* $t : \forall\alpha.A$ *is compilable then* $t(T) : A[T/\alpha]$ *is compilable.*
(4) *If* $t : \alpha$ *is compilable, then* t *is decompilable.*
(5) *If* $t = x(t_1)\dots(t_n) : A$ *and each* t_i *is either decompilable or a type, then* t *is compilable. If* $A = \alpha$*, then* t *is also decompilable.*
(6) *If* t *is decompilable then* $\lambda x : A.t$ *is decompilable.*
(7) *If* t *is decompilable then* $\lambda\alpha.t$ *is decompilable.*
(8) *If* t *is* $\beta\eta$*-long normal then* t *is decompilable.*

Proof. (1) By definition.
(2) By definition.
(3) By definition and Lemma 7.2(5).
(4) If $t : \alpha$, then both "decompilable" and "compilable" unfold to $\sigma(t^*) =_{\beta\eta} [\![t]\!]_\tau$.
(5) Assume $t = x(t_1)\dots(t_n) : A$, with each t_i a term or a type. Then by induction on n we can show that t is compilable: if $n = 0$ we use (1). If t_n is a term, this case is shown by (2). If t_n is a type, this case is shown by (3). Now in the case $t : \alpha$ by (4) we conclude that t is decompilable.
(6) By definition.
(7) By definition and Lemma 7.2(4).
(8) Every $\beta\eta$-long normal term t in $\mathcal{F}$ has the form $\lambda x_1.\dots\lambda x_n.x(t_1,\dots,t_m)$, where each x_i is a term variable or a type variable, each t_j is a $\beta\eta$-long normal term or a type, and $x(t_1,\dots,t_m) : \alpha$ for some type variable α. We argue by induction on the size of the $\beta\eta$-long normal form. If $n = 0$ we apply (5). If $n \geq 1$, let $t' = \lambda x_2.\dots\lambda x_n.x(t_1,\dots,t_m)$. If x_1 is a term variable, the thesis is shown by (6) applied to t'. If x_n is a type variable, the thesis is shown by (7) applied to t'. □

Theorem 6.5(2) (there is a decompiler–normalizer for $\mathcal{F}^*$ in $\mathcal{F}$) is an immediate consequence of Lemma 7.4(8). There is no compiler for $\mathcal{F}^*$ in $\mathcal{F}$ because the composition of a decompiler–normalizer and a compiler defines a normalizer, which is the key idea in normalization by evaluation.

Proof of Theorem 6.5. Assume that no variable of the form f_β, g_β is free in t. Therefore Lemmas 7.3 and 7.4 apply to t.

(1) $(.)^*$ commutes with $\beta\eta$ by Lemma 7.3(5).
(2) Assume t is a closed term, A is a closed type, and $t : A$. Let $t' =_{\beta\eta} t$ be the $\beta\eta$-long normal form of t. Then by Lemma 7.4(8) we have $f_A(t'^*) =_{\beta\eta} [\![t']\!]$ (the substitutions σ, τ are empty because t, A are closed). By Lemma 7.3(5) we have $t^* =_{\beta\eta} t'^*$. We conclude $f_A(t^*) =_{\beta\eta} f_A(t'^*) =_{\beta\eta} [\![t']\!]$. Thus, $f^c_A(t^{*c}) =_{\beta\eta} \lambda\Gamma_{\mathrm{Tm}}.f_A(t^*) =_{\beta\eta} \lambda\Gamma_{\mathrm{Tm}}.[\![t']\!] = [\![t']\!]^c$. The family f^c_A of closed terms of $\mathcal{F}$ is a decompiler.
(3) Assume that there is some family $G_A : \mathrm{Tm}^c \to A^{*c}$ of closed terms of $\mathcal{F}$, for A closed type of $\mathcal{F}$, which is a compiler. Define $\mathrm{ev}_A = f^c_A \circ G_A : \mathrm{Tm}^c \to \mathrm{Tm}^c$. Let $t : A$ be a closed term of $\mathcal{F}$ and $t' =_{\beta\eta} t$ be the $\beta\eta$-long normal form of t. Then we have $\mathrm{ev}_A([\![t]\!]^c) = f^c_A \circ G_A([\![t]\!]^c) =_{\beta\eta} f^c_A(t^{*c}) =_{\beta\eta} [\![t']\!]^c$. Thus, the family ev_A is a normalizer, which contradicts Theorem 5.2(1). □

8. What we can prove about $\mathcal{F}$ and $\beta\eta$-completeness

In this section we conjecture that we may use the map $(.)^*$ or $(.)^{*c}$ in order to define some class of $\beta\eta$-complete models. We cannot prove this goal yet, but we can prove a result which is close to this goal: inside every *consistent* model

$$\mathcal{M} = \langle \mathrm{Tp}, \mathrm{E}(.), \mathrm{Pred}, \Rightarrow, \Pi, \{\Phi_{a,b}\}_{a,b\in\mathrm{Tp}}, \{\Phi_F\}_{F\in\mathrm{Pred}}, [\![.]\!]\rangle$$

of $\mathcal{F}$ (Definition 4.3) we may internally define some model $\mathcal{M}^*$ whose equational theory includes $\beta\eta$ and is included in $|\beta\eta|$. This is not yet our goal because $\beta\eta \subset |\beta\eta|$.

Let $\delta = [\mathrm{DB}/\mathrm{Tm}, \mathrm{Lam}/\mathrm{lam}, \mathrm{Ap}/\mathrm{ap}]$ be the de Bruijn substitution (Definition 5.6). The substitution δ replaces the free variables Tm, lam, ap, denoting a *generic coding* for Λ in $\mathcal{F}$, with the closed type DB and the closed terms Lam, Ap, denoting *de Bruijn level coding* for Λ in $\mathcal{F}$. We define a type structure for $\mathcal{M}^*$ (Definition 4.3), and then we prove that it may be extended in a canonical way to a model. A type in Tp^* is a triple of a type of $\mathcal{M}$ and a connection pair between the type and the interpretation of DB in $\mathcal{M}$. The interpretation $[\![.]\!]^*$ of a type or a term in $\mathcal{M}^*$ is obtained by composing the interpretation $(.)^*$ of $\mathcal{F}$ into $\mathcal{F}, \delta$, and the interpretation $[\![.]\!]$ of $\mathcal{F}$ in $\mathcal{M}$. Note that we could also define $\mathcal{M}^*$ by using $(.)^{*c}$ instead of $(.)^*$, but we will use $(.)^*$ in this section for notational simplicity. We could also replace the type DB we used in the definition of $\mathcal{M}^*$ by any type of an α-complete coding: we chose de Bruijn coding because it is commonly considered the canonical coding of untyped λ-terms.

Definition 8.1 (*An Internal Type Structure for* $\mathcal{M}$)**.** Assume $\mathcal{M}$ is any model of $\mathcal{F}$. Then we define a type structure $\langle \mathrm{Tp}^*, \mathrm{E}(.)^*, [\![.]\!]^*\rangle$ for $\mathcal{F}$ as follows. Let $\mathrm{DB} = [\![\mathrm{DB}]\!]$ be the interpretation of de Bruijn's type in $\mathcal{M}$.

(1) The set Tp^* of types of $\mathcal{M}^*$ consists of all triples (a, ϕ, ψ), with $a \in \mathrm{Tp}$ and $\phi \in \mathrm{E}(a \Rightarrow \mathrm{DB})$, $\psi \in \mathrm{E}(\mathrm{DB} \Rightarrow a)$, two maps connecting a and DB.
(2) The set of elements of the type $(a, \phi, \psi) \in \mathrm{Tp}^*$ is the set $\mathrm{E}((a, \phi, \psi))^* = \mathrm{E}(a)$.

(3) Let $\Delta = \{\alpha_1, \ldots, \alpha_n\}$ and $\sigma : \Delta \to \mathrm{Tp}^*$, with $\sigma(\alpha_i) = (a_i, \phi_i, \psi_i)$, for $i = 1, \ldots, n$. Let σ^* be the type and term assignment of $\mathcal{M}$ defined by $\sigma^*(\alpha_i) = a_i$, $\sigma^*(f_{\alpha_i}) = \phi_i$, and $\sigma^*(g_{\alpha_i}) = \psi_i$.
   (a) For any type A of $\mathcal{F}$ with free type variables in $\Delta = \alpha_1, \ldots, \alpha_n$, and any assignment σ, σ^* as above, we set
   $$[\![A]\!]^*_\sigma = ([\![\delta(A^*)]\!]_{\sigma^*}, [\![\delta(f_A)]\!]_{\sigma^*}, [\![\delta(g_A)]\!]_{\sigma^*}).$$
   (b) For any term t of $\mathcal{F}$ in the context $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ with Δ, any σ as above, and any substitution τ over Γ such that $\tau(x_i) \in \mathrm{E}([\![A_i]\!]^*_\sigma)$ for $i = 1, \ldots, n$ we set:
   $$[\![t]\!]^*_{\sigma,\tau} = [\![\delta(t^*)]\!]_{\sigma^*,\tau}.$$

Note that $t^{*c}(\Gamma_{DB}) =_{\beta\eta} \delta(t^*)$.
We will prove that there is a unique model

$$\mathcal{M}^* = \langle \mathrm{Tp}^*, \mathrm{E}(.)^*, \mathrm{Pred}^*, \Rightarrow^*, \Pi^*, \{\Phi^*_{a,b}\}_{a,b\in\mathrm{Tp}^*}, \{\Phi^*_F\}_{F\in\mathrm{Pred}^*}, [\![.]\!]^*\rangle$$

of $\mathcal{F}$, included in $\mathcal{M}$, and obtained by extending the type structure $\langle \mathrm{Tp}^*, \mathrm{E}(.)^*, [\![.]\!]^*\rangle$ with the minimum set Pred* of predicates. We have to isolate a necessary and sufficient condition for the existence and uniqueness of such an extension. Following Barendregt [4], we say that an interpretation $[\![.]\!]$ is "Weakly Extensional" if the interpretations of the type $\forall\alpha.A$ and the terms $\lambda x : B.t$, $\lambda\alpha.u$ are uniquely determined by the interpretations of all instances of A, t, u.

Definition 8.2 (*Weak Extensionality*)**.** Assume $\langle \mathrm{Tp}, \mathrm{E}(.), [\![.]\!]\rangle$ is a type structure. Let A, B be types and t, u, v, w be terms. Let σ and τ be substitutions on the free variables of A, B, t, u, v, w different from α, x.
$[\![.]\!]$ is weakly extensional if the following holds.

(1) If $[\![A]\!]_{\sigma,a/\alpha} = [\![B]\!]_{\sigma,a/\alpha}$ for all $a \in \mathrm{Tp}$, then $[\![\forall\alpha.A]\!]_\sigma = [\![\forall\alpha.B]\!]_\sigma$.
(2) If $[\![t]\!]_{\sigma,(\tau,c/x)} = [\![u]\!]_{\sigma,(\tau,c/x)}$ for all $c \in [\![A]\!]_\sigma$, then $[\![\lambda x : A.t]\!]_{\sigma,\tau} = [\![\lambda x : A.u]\!]_{\sigma,\tau}$.
(3) If $[\![v]\!]_{(\sigma,a/\alpha),\tau} = [\![w]\!]_{(\sigma,a/\alpha),\tau}$ for all $a \in \mathrm{Tp}$, then $[\![\lambda\alpha.v]\!]_{\sigma,\tau} = [\![\lambda\alpha.w]\!]_{\sigma,\tau}$.

We may now state and prove a necessary and sufficient condition for extending a type structure to a model.

Lemma 8.3 (*Extension Lemma*)**.** *Let $\langle Tp, \mathrm{E}(.), [\![.]\!]\rangle$ by any type structure. Then $\langle Tp, \mathrm{E}(.), [\![.]\!]\rangle$ can be extended to a model $\mathcal{M}$ of $\mathcal{F}$ if and only if:*

- $[\![.]\!]$ *commutes with substitutions*
- $[\![.]\!]$ *commutes with $\beta\eta$ (Definition 4.3(4)(a) and (b))*
- $[\![.]\!]$ *satisfies Weak Extensionality.*

If the extension exists, it is unique if we choose the smallest possible Pred.

Proof. First, we prove that every extension of a type structure to a model satisfies all conditions above. Then, we prove that every type structure satisfying all conditions above may be extended to a model with the smallest possible Pred. We will sometimes write $[\![t]\!]_{a/\alpha,c/x,b/\beta,d/y}$ for $[\![t]\!]_{(a/\alpha,b/\beta),(c/x,d/y)}$.

(1) Assume that $\langle \mathrm{Tp}, \mathrm{E}(.), [\![.]\!]\rangle$ may be extended to a model $\mathcal{M}$. Then the following conditions are satisfied:
   (a) *Commutation with type substitution:* $[\![A[T/\alpha]]\!]_\sigma = [\![A]\!]_{\sigma,[\![T]\!]_\sigma/\alpha}$. By induction over A, using the conditions over $[\![.]\!]_\sigma$ for a model.
   (b) *Commutation with term substitution:* $[\![t[T/\alpha, u/x]]\!]_{\sigma,\tau} = [\![t]\!]_{(\sigma,[\![T]\!]_\sigma/\alpha),(\tau,[\![u]\!]_{\sigma,\tau}/x)}$. By induction over t, using the conditions over $[\![.]\!]_{\sigma,\tau}$ for a model, and commutation with type substitution.
   (c) *Commutation with β.* By unfolding the interpretation of $(\lambda x : A.t)(u)$, $(\lambda\alpha.v)(T)$ and by commutation with term substitution.
   (d) *Commutation with η.* Assume that $t : A \to B$, $x \notin \mathrm{FV}(t)$, and $a = [\![A]\!]_\sigma$, $b = [\![B]\!]_\sigma$. For all $c \in \mathrm{E}([\![A]\!]_\sigma)$ define $\phi(c) = [\![tx]\!]_{\sigma,(\tau,c/x)}$. By interpretation of λ-abstraction we have $[\![\lambda x.tx]\!]_{\sigma,\tau} = \Phi^{-1}_{a,b}(\phi)$. By the interpretation of application we have $\Phi_{a,b}([\![t]\!]_{\sigma,\tau})(c) = \Phi_{a,b}([\![t]\!]_{\sigma,\tau}, [\![x]\!]_{c/x}) = [\![tx]\!]_{\sigma,(\tau,c/x)} = \phi(c)$. Hence $\Phi_{a,b}([\![t]\!]_{\sigma,\tau}) = \phi$. Thus, $[\![t]\!]_{\sigma,\tau} = \Phi^{-1}_{a,b}(\phi) = [\![\lambda x.tx]\!]_{\sigma,\tau}$. The case of $\lambda\alpha.t\alpha$ with $\alpha \notin \mathrm{FV}(t)$ is proved in a similar way.
   (e) *Weak Extensionality.* Immediate: the interpretations of $\forall\alpha.A$, $\lambda x : B.t$, $\lambda\alpha.u$ are defined in term of the set of interpretations of all instances of A, t, u.
(2) Assume we have a type structure $\langle \mathrm{Tp}, \mathrm{E}(.), [\![.]\!]\rangle$ which satisfies the following conditions: $[\![.]\!]$ commutes with substitution and $\beta\eta$, and it satisfies Weak Extensionality. We prove that we may extend the type structure to a model $\mathcal{M}$ having the smallest possible set Pred, and that this extension is unique.
   The smallest set Pred of predicates of $\mathcal{M}$ is the set of all maps $F : \mathrm{Tp} \to \mathrm{Tp}$ such that there are a type A of $\mathcal{F}$, a type variable α, and a substitution σ of $\mathcal{M}$, and for all $a \in \mathrm{Tp}$, $F(a) = [\![A]\!]_{\sigma,a/\alpha}$.
   Clearly, this choice for the set Pred is the smallest possible. We have to prove that there is a unique model $\mathcal{M}$ extending Tp, E(.), Pred, $[\![.]\!]$. If there is such a model, the only possible choices for $\Rightarrow$, Π, $\Phi_{a,b}$, Φ_F are the following:
   (a) $a \Rightarrow b = [\![\alpha \to \beta]\!]_{a/\alpha,b/\beta}$,
   (b) $\Pi(F) = [\![\forall\alpha.A]\!]_\sigma$ if $F(a) = [\![A]\!]_{\sigma,a/\alpha}$,

(c) $\Phi_{a,b}(c,d) = [\![x(y)]\!]_{c/x,d/y}$,

(d) $\Phi_F(c,a) = [\![x(\alpha)]\!]_{c/x,a/\alpha}$ if $F(a') = [\![A]\!]_{\sigma,a'/\alpha}$ and $x : \forall\alpha.A$.

Conversely, for a given $[\![.]\!]$ the equations above define $\Rightarrow$, Π, $\Phi_{a,b}$, Φ_F. This is immediate for $\Rightarrow$, $\Phi_{a,b}$, Φ_F. In the case of Π, the uniqueness of the value of $\Pi(F)$ follows from Weak Extensionality for $[\![.]\!]$ and $\forall$. Assume $F(a) = [\![A]\!]_{\sigma,a/\alpha} = [\![A']\!]_{\sigma',a/\alpha}$ for all $a \in \mathrm{Tp}$. By possibly renaming we have $\mathrm{dom}(\sigma) \cap \mathrm{dom}(\sigma') = \emptyset$. We deduce $F(a) = [\![A]\!]_{\sigma,\sigma',a/\alpha} = [\![A']\!]_{\sigma,\sigma',a/\alpha}$ for all $a \in \mathrm{Tp}$, and we conclude $\Pi(F) = [\![\forall\alpha.A]\!]_{\sigma} = [\![\forall\alpha.A]\!]_{\sigma,\sigma'} = [\![\forall\alpha.A']\!]_{\sigma,\sigma'} = [\![\forall\alpha.A']\!]_{\sigma'}$.

We have still to check the injectivity conditions:

(a) $\Phi_{a,b}$ *is an injection*. Indeed, assume $c, c' \in \mathrm{E}(a \Rightarrow b)$ and $\Phi_{a,b}(c) = \Phi_{a,b}(c')$. Then for all $d \in \mathrm{E}(a)$ we have $\Phi_{a,b}(c)(d) = \Phi_{a,b}(c')(d)$, and by definition of $\Phi_{a,b}$ we have $[\![xy]\!]_{c/x,d/y} = [\![xy]\!]_{c'/x,d/y}$. This equation is equivalent to $[\![xy]\!]_{a/\alpha,(c/x,c'/x',d/y)} = [\![x'y]\!]_{a/\alpha,(c/x,c'/x',d/y)}$. By Weak Extensionality of $[\![.]\!]$ we deduce $[\![\lambda y : \alpha.xy]\!]_{c/x,c'/x'} = [\![\lambda y : \alpha.x'y]\!]_{c/x,c'/x}$. By commutation with the η-rule we conclude $[\![x]\!]_{c/x,c'/x'} = [\![x']\!]_{c/x,c'/x'}$, that is, $c = c'$.

(b) Φ_F *is an injection*. This is proved in a way similar to the previous point.

We can prove that $\Rightarrow$ and Π satisfy the two conditions for the interpretation of types in a model.

(a) By commutativity with substitution of $[\![.]\!]$ we have $[\![A]\!]_{\sigma} \Rightarrow [\![B]\!]_{\sigma} = [\![\alpha \to \beta]\!]_{[\![A]\!]_{\sigma}/\alpha,[\![B]\!]_{\sigma}/\beta} = [\![A \to B]\!]_{\sigma}$.

(b) The condition: " if $F(a) = [\![A]\!]_{\sigma,a/\alpha}$ then $\Pi(F) = [\![\forall\alpha.A]\!]_{\sigma}$" holds by definition of Π.

We prove now that the interpretation of $[\![.]\!]_{\sigma,\tau}$ satisfies the four conditions for the interpretation of terms in a model. We use commutativity with substitution and β of $[\![.]\!]$, and injectivity of $\Phi_{a,b}$, Φ_F.

(a) $\Phi_{a,b}([\![t]\!]_{\sigma,\tau}, [\![u]\!]_{\sigma,\tau}) = [\![x(y)]\!]_{[\![t]\!]_{\sigma,\tau}/x,[\![u]\!]_{\sigma,\tau}/y} = [\![t(u)]\!]_{\sigma,\tau}$, if $a = [\![A]\!]_{\sigma}$ and $b = [\![B]\!]_{\sigma}$.

(b) Assume $a = [\![A]\!]_{\sigma}$ and $b = [\![B]\!]_{\sigma}$ and $\phi(d) = [\![t]\!]_{\sigma,(\tau,d/y)}$ for all $d \in \mathrm{E}(a)$. Then $\Phi_{a,b}([\![\lambda y : A.t]\!]_{\sigma,\tau})(d) = [\![x(y)]\!]_{([\![\lambda x:A.t]\!]_{\sigma,\tau})/x,d/y} = [\![(\lambda y : A.t)(y)]\!]_{\sigma,(\tau,d/y)} = [\![t]\!]_{\sigma,(\tau,d/y)} = \phi(d)$. Thus, $[\![\lambda y : A.t]\!]_{\sigma,\tau} = \Phi^{-1}_{a,b}(\phi)$.

(c) For any type B with free variables in the domain of σ we have: $\Phi_F([\![t]\!]_{\sigma,\tau}, [\![B]\!]_{\sigma}) = [\![x(\alpha)]\!]_{[\![t]\!]_{\sigma,\tau}/x,[\![B]\!]_{\sigma}/\alpha} = [\![t(B)]\!]_{\sigma,\tau}$ if $F(a) = [\![A]\!]_{\sigma,a/\alpha}$ and $x : \forall\alpha.A$.

(d) Assume $F(a) = [\![A]\!]_{\sigma,a/\alpha}$, $x : \forall\alpha.A$, and $\psi(a) = [\![t]\!]_{(\sigma,a/\alpha),\tau}$ for all $a \in \mathrm{Tp}$. Then $\Phi_F([\![\lambda\alpha.t]\!]_{\sigma,\tau})(a) = [\![x(\alpha)]\!]_{a/\alpha,[\![\lambda\alpha.t]\!]_{\sigma,\tau}/x} = [\![(\lambda\alpha.t)(\alpha)]\!]_{(\sigma,a/\alpha),\tau} = [\![t]\!]_{(\sigma,a/\alpha),\tau} = \psi(a)$. Thus, $[\![\lambda\alpha.t]\!]_{\sigma,\tau} = \Phi_F^{-1}(\psi)$. □

This finishes the definition of $\mathcal{M}$ and the proof that $\mathcal{M}$ is unique if Pred is the smallest possible. □

The type structure $\langle \mathrm{Tp}^*, \mathrm{E}(.)^*, [\![.]\!]^* \rangle$ satisfies all conditions required to be extended to a model $\mathcal{M}^*$.

Lemma 8.4 (*Extension to a Model*)**.** *Assume $\mathcal{M}$ is any model and $\langle Tp^*, \mathrm{E}(.), [\![.]\!]^* \rangle$ is the type structure defined from $\mathcal{M}$.*

(1) $[\![.]\!]^*$ *commutes with substitution and $\beta\eta$, and it is weakly extensional.*

(2) *There is a unique extension of $\langle Tp^*, \mathrm{E}(.), [\![.]\!]^* \rangle$ to a model $\mathcal{M}^*$ with the minimum set Pred*.*

Proof. (1) The commutation of $[\![.]\!]^*$ with substitution is a consequence of Lemma 7.2(1)(2)(4) (the commutation of $(.)^*$ w.r.t. substitution) and commutation of $[\![.]\!]$ and $\delta(.)$ with substitution. Commutation of $[\![.]\!]^*$ w.r.t. $\beta\eta$ is a consequence of Lemma 7.3(5) (commutation of $(.)^*$ w.r.t. $\beta\eta$) and commutation of $[\![.]\!]$ and $\delta(.)$ w.r.t. $\beta\eta$. Weak Extensionality of $[\![.]\!]^*$ is a consequence of Weak Extensionality for $[\![.]\!]$ and the fact that $(.)^*$ is uniquely defined in term of $\to$, $\forall$, λ.

(2) By (1) and Lemma 8.3. □

This concludes the definition of $\mathcal{M}^*$ and the proof that it is a model. The decompiler for $\mathcal{F}^*$ is interpreted as a family of maps in $\mathcal{M}^*$. By applying them we can prove:

Theorem 8.5. *If $\mathcal{M}$ is consistent, then its equational theory $=_{\mathcal{M}^*}$ is between $\beta\eta$ and $|\beta\eta|$.*

Proof. Since $\mathcal{M}^*$ is a model of $\mathcal{F}$, its equational theory includes $\beta\eta$. Assume t, u are closed terms of $\mathcal{F}$ of a closed type A, and $t =_{\mathcal{M}^*} u$. Let t' and u' be the $\beta\eta$-long normal forms of t and u respectively. We have $t' =_{\mathcal{M}^*} u'$, therefore $\delta(t'^*) =_{\mathcal{M}} \delta(u'^*)$ by definition of $\mathcal{M}^*$. By Theorem 6.5 for the coding in Tm^c, we deduce that $\delta([\![t']\!]) =_{\beta\eta} \delta(f_A(t'^*)) =_{\mathcal{M}} \delta(f_A(u'^*)) =_{\beta\eta} \delta([\![u']\!])$, and therefore $\delta([\![t']\!]) =_{\mathcal{M}} \delta([\![u']\!])$ in the type $\delta(\mathrm{Tm}) = \mathrm{DB} = (\mathrm{Nat} \to \mathrm{dB}^c)$. We claim that this implies $t =_{|\beta\eta|} u$ in $\mathcal{F}$.

Indeed, if we apply both sides to $\underline{0}$ we deduce $\delta([\![t']\!])(\underline{0}) =_{\mathcal{M}} \delta([\![u']\!])(\underline{0})$ in the type dB^c, so $\delta([\![t]\!])(\underline{0}) =_{\beta\eta} \delta([\![u]\!])(\underline{0})$ in $\mathcal{F}$ by $\beta\eta$-completeness of dB^c in the consistent model $\mathcal{M}$. By $\delta([\![t']\!])(\underline{0}) =_{\beta\eta} \ulcorner |t'| \urcorner^c_0$ in the proof of Lemma 5.7(2), we conclude $\ulcorner |t'| \urcorner^c_0 =_{\beta\eta} \ulcorner |u'| \urcorner^c_0$, so $|t'| =_{\alpha} |u'|$. We conclude $|t| =_{\beta\eta} |t'| =_{\alpha} |u'| =_{\beta\eta} |u|$ and hence $t =_{|\beta\eta|} u$ in $\mathcal{F}$. □

Remark that the equational theory of $\mathcal{M}$ could be, say, observational equality: yet, even in this case, we may define inside $\mathcal{M}$, and using only the language of $\mathcal{M}$, some model $\mathcal{M}^*$ whose equational theory is weaker than $|\beta\eta|$.

We would like to prove a stronger result: *"in every model $\mathcal{M}$ of $\mathcal{F}$ we can define some model $\mathcal{M}^*$ whose equational theory is* exactly *$\beta\eta$."* We conjecture that this can be done if in the proof of the previous theorem we change the type Tm representing all untyped λ-terms of Λ in $\mathcal{F}$ to some suitable type of $\mathcal{F}$ representing all pseudo-terms of $\mathcal{F}$. The reason why we cannot prove this result for the moment is that our decompiler produces codes of untyped λ-terms, erasing all types from the term, and therefore it identifies any two terms which are equal in $|\beta\eta|$.

Appendix

Proof of Lemma 7.2. (1) By induction on A. We distinguish three cases, according to the first symbol of A.

(a) Assume $A = \beta$. Then $A[T/\alpha]^* = \beta[T/\alpha]^*$ and $A^*[T^*/\alpha^*] = \beta^*[T^*/\alpha^*] = \beta[T^*/\alpha]$. If $\beta = \alpha$ then both expressions are T^*. If $\beta \neq \alpha$ then both expressions are β.

(b) Assume $A = B \to C$. Then $A[T/\alpha]^* = (B[T/\alpha] \to C[T/\alpha])^* = B[T/\alpha]^* \to C[T/\alpha]^* =$ (by induction hypothesis on B, C) $B^*[T^*/\alpha] \to C^*[T^*/\alpha] = (B^* \to C^*)[T^*/\alpha] = A^*[T^*/\alpha]$.

(c) Assume $A = \forall\beta.B$: by possibly renaming β we have $\alpha \neq \beta$, Tm and $\beta \notin \mathrm{FV}(T)$. Then $A[T/\alpha]^* =$ (by $\alpha \neq \beta$) $(\forall\beta.B[T/\alpha])^* = \forall\beta.(\beta \to \mathrm{Tm}) \to (\mathrm{Tm} \to \beta) \to (B[T/\alpha])^* =$ (by induction hypothesis on B) $= \forall\beta.(\beta \to \mathrm{Tm}) \to (\mathrm{Tm} \to \beta) \to B^*[T^*/\alpha] =$ (by $\alpha \neq \beta$, Tm) $(\forall\beta.(\beta \to \mathrm{Tm}) \to (\mathrm{Tm} \to \beta) \to B^*)[T^*/\alpha] = (\forall\beta.B)^*[T^*/\alpha] = A^*[T^*/\alpha]$.

(2) By induction on t. We distinguish five cases, according to the first symbol of t.

(a) Assume $t = y$. Then $t[u/x]^* = y[u/x]^*$ and $t^*[u^*/x] = y[u^*/x]$. If $y = x$ then both expressions are u^*. If $y \neq x$ then both expressions are y.

(b) Assume $t = vw$, an application to a term. Then $t[u/x]^* = (v[u/x]w[u/x])^* = v[u/x]^*w[u/x]^* =$ (by induction hypothesis on v, w) $v^*[u^*/x]w^*[u^*/x] = (v^*w^*)[u^*/x] = t^*[u^*/x]$.

(c) Assume $t = \lambda y : B.v$: by possibly renaming y we may also assume $x \neq y \notin \mathrm{FV}(u)$. Then $t[u/x]^* =$ (by $x \neq y$) $(\lambda y : B.v[u/x])^* = \lambda y : B^*.(v[u/x])^* =$ (by induction hypothesis on v) $\lambda y : B^*.(v^*[u^*/x]) =$ (by $x \neq y$) $(\lambda y : B^*.v^*)[u^*/x] = t^*[u^*/x]$.

(d) Assume $t = vT$. The variable x is not free in f_T, g_T because $x \neq f_\beta, g_\beta$ for any β. Then $t[u/x]^* = (v[u/x]T)^* = v[u/x]^*T^*f_Tg_T =$ (by induction hypothesis on v) $v^*[u^*/x]T^*f_Tg_T =$ (by x not free in f_T, g_T) $(v^*T^*f_Tg_T)[u^*/x] = t^*[u^*/x]$.

(e) Assume $t = \lambda\alpha.v$. By possibly renaming we may assume that $f_\alpha, g_\alpha \notin \mathrm{FV}(u^*)$. Then $t[u/x]^* = (\lambda\alpha.v[u/x])^* = \lambda\alpha.\lambda f_\alpha : \alpha \to \mathrm{Tm}.\lambda g_\alpha : \mathrm{Tm} \to \alpha.(v[u/x])^* =$ (by induction hypothesis on v) $\lambda\alpha.\lambda f_\alpha : \alpha \to \mathrm{Tm}.\lambda g_\alpha : \mathrm{Tm} \to \alpha.(v^*[u^*/x]) =$ (by $x \neq f_\alpha, g_\alpha$ and $f_\alpha, g_\alpha \notin \mathrm{FV}(u^*)$) $(\lambda\alpha.\lambda f_\alpha : \alpha \to \mathrm{Tm}.\lambda g_\alpha : \mathrm{Tm} \to \alpha.v^*)[u^*/x] = t^*[u^*/x]$.

(3) Recall that $\sigma = [T^*/\alpha, f_T/f_\alpha, g_T/g_\alpha]$. We prove $f_{A[T/\alpha]} = \sigma(f_A)$ and $g_{A[T/\alpha]} = \sigma(g_A)$ by simultaneous induction on A. We distinguish three cases according to the first symbol of A.

(a) Assume $A = \beta$, a type variable. Then $f_{A[T/\alpha]} = f_{\beta[T/\alpha]}$ and $\sigma(f_A) = (f_\beta)[f_T/f_\alpha]$. If $\beta = \alpha \neq \mathrm{Tm}$ then f_β is a variable and $f_\beta = f_\alpha$: thus, both sides are f_T. If $\beta \neq \alpha$ then either $\beta = \mathrm{Tm}$ and $f_\beta = \mathrm{id}_{\mathrm{Tm}}$, or $\beta \neq \mathrm{Tm}$ and f_β is a variable $\neq f_\alpha$: in both cases, both sides are the term f_β. For the same reason if $\beta = \alpha$ then $g_{A[T/\alpha]} = g_T = \sigma(g_A)$ and if $\beta \neq \alpha$ then $g_{A[T/\alpha]} = g_\beta = \sigma(g_A)$.

(b) Assume $A = B \to C$. Then $f_{A[T/\alpha]} = f_{B[T/\alpha]\to C[T/\alpha]} = \lambda y : (B[T/\alpha] \to C[T/\alpha])^*.\mathrm{lam}(f_{C[T/\alpha]} \circ y \circ g_{B[T/\alpha]}) =$ (by (1) and induction hypothesis on B, C) $\lambda y : (B^* \to C^*)[T^*/\alpha].\mathrm{lam}(\sigma(f_C) \circ y \circ \sigma(g_B)) = \lambda y : \sigma(B^* \to C^*).\mathrm{lam}(\sigma(f_C) \circ y \circ \sigma(g_B)) = \sigma(\lambda y : (B^* \to C^*).\mathrm{lam}(f_C \circ y \circ g_B)) = \sigma(f_A)$. In a similar way we have $g_{A[T/\alpha]} = g_{B[T/\alpha]\to C[T/\alpha]} = \lambda y : \mathrm{Tm}.g_{C[T/\alpha]} \circ \mathrm{ap}(y) \circ f_{B[T/\alpha]} =$ (by induction hypothesis on B, C) $\lambda y : \mathrm{Tm}.\sigma(g_C) \circ \mathrm{ap}(y) \circ \sigma(f_B)) = \sigma(\lambda y : \mathrm{Tm}.g_C \circ \mathrm{ap}(y) \circ f_B) = \sigma(g_A)$.

(c) Assume $A = \forall\beta.B$: by possibly renaming y, β we may assume: $f_\alpha, g_\alpha \neq y$, β is not free in T, and $\alpha \neq \beta$. Therefore $f_\alpha, g_\alpha \neq f_\beta, g_\beta$. As a consequence we have $B[T/\alpha][\mathrm{Tm}/\beta] = B[\mathrm{Tm}/\beta][T/\alpha]$. Then $f_{A[T/\alpha]} = \lambda y : (\forall\beta.B)[T/\alpha]^*.f_{B[T/\alpha][\mathrm{Tm}/\beta]}(y(\mathrm{Tm}, \mathrm{id}_{\mathrm{Tm}}, \mathrm{id}_{\mathrm{Tm}})) =$ (by (1), $B[T/\alpha][\mathrm{Tm}/\beta] = B[\mathrm{Tm}/\beta][T/\alpha]$, and induction hypothesis on $B[\mathrm{Tm}/\beta]$) $\lambda y : (\forall\beta.B)^*[T^*/\alpha].\sigma(f_{B[\mathrm{Tm}/\beta]})(y(\mathrm{Tm}, \mathrm{id}_{\mathrm{Tm}}, \mathrm{id}_{\mathrm{Tm}})) = \lambda y : \sigma((\forall\beta.B)^*).\sigma(f_{B[\mathrm{Tm}/\beta]})(y(\mathrm{Tm}, \mathrm{id}_{\mathrm{Tm}}, \mathrm{id}_{\mathrm{Tm}})) = \sigma(\lambda y : (\forall\beta.B)^*.f_{B[\mathrm{Tm}/\beta]}(y(\mathrm{Tm}, \mathrm{id}_{\mathrm{Tm}}, \mathrm{id}_{\mathrm{Tm}}))) = \sigma(f_A)$. In a similar way we have $g_{A[T/\alpha]} =$ (by $\alpha \neq \beta$) $g_{\forall\beta.B[T/\alpha]} = \lambda y : \mathrm{Tm}.\lambda\beta.\lambda f_\beta : \beta \to \mathrm{Tm}.\lambda g_\beta : \mathrm{Tm} \to \beta.g_{B[T/\alpha]}(y) =$ (by induction hypothesis on B) $\lambda y : \mathrm{Tm}.\lambda\beta.\lambda f_\beta : \beta \to \mathrm{Tm}.\lambda g_\beta : \mathrm{Tm} \to \beta.\sigma(g_B) =$ (by $f_\alpha, g_\alpha \neq f_\beta, g_\beta, y$) $\sigma(\lambda y : \mathrm{Tm}.\lambda\beta.\lambda f_\beta : \beta \to \mathrm{Tm}.\lambda g_\beta : \mathrm{Tm} \to \beta.g_B) = \sigma(g_A)$.

(4) Recall that $\sigma = [T^*/\alpha, f_T/f_\alpha, g_T/g_\alpha]$. We argue by induction on t. We distinguish five cases, according to the first symbol of t.

(a) Assume $t = y$. We have $y \neq f_\alpha, g_\alpha$ because f_α, g_α are not free in t. Then $t[T/\alpha]^* = y^* = y$ and $\sigma(t^*) = \sigma(y) = y$.

(b) Assume $t = vw$. Then $t[T/\alpha]^* = v[T/\alpha]^*w[T/\alpha]^* =$ (by induction hypothesis on v, w) $\sigma(v^*)\sigma(w^*) = \sigma(v^*w^*) = \sigma((vw)^*) = \sigma(t^*)$.

(c) Assume $t = \lambda y : B.v$: by possibly renaming y we have $f_\beta, g_\beta \neq y \notin \mathrm{FV}(T)$. Then $t[T/\alpha]^* = \lambda y : B[T/\alpha]^*.v[T/\alpha]^* =$ (by (1) and induction hypothesis on v) $\lambda y : \sigma(B^*).\sigma(v^*) =$ (by $f_\beta, g_\beta \neq y$) $\sigma(\lambda y : B^*.v^*) = \sigma(t^*)$.

(d) Assume $t = vU$. Then $t[T/\alpha]^* = v[T/\alpha]^*(U[T/\alpha]^*, f_{U[T/\alpha]}, g_{U[T/\alpha]}) =$ (by induction hypothesis on v, (1) and (3)) $\sigma(v^*)(U^*[T^*/\alpha], \sigma(f_U), \sigma(g_U)) = \sigma(v^*)(\sigma(U^*), \sigma(f_U), \sigma(g_U)) = \sigma(v^*(U^*, f_U, g_U)) = \sigma(t^*)$.

(e) Assume $t = \lambda\beta.v$: by possibly renaming β, f_β, g_β we have $\alpha \neq \beta$ and $f_\alpha, g_\alpha \neq f_\beta, g_\beta$ and $f_\beta, g_\beta, \beta \notin \mathrm{FV}(T), \mathrm{FV}(f_T), \mathrm{FV}(g_T)$. Therefore $t[T/\alpha]^* = \lambda\beta.\lambda f_\beta : \beta \to \mathrm{Tm}.\lambda g_\beta : \mathrm{Tm} \to \beta.(v[T/\alpha]^*) =$ (by induction hypothesis on v) $\lambda\beta.\lambda f_\beta : \beta \to \mathrm{Tm}.\lambda g_\beta : \mathrm{Tm} \to \beta.\sigma(v^*) =$ (by $\alpha, f_\alpha, g_\alpha \neq \beta, f_\beta, g_\beta$) $\sigma(\lambda\beta.\lambda f_\beta : \beta \to \mathrm{Tm}.\lambda g_\beta : \mathrm{Tm} \to \beta.v^*) = \sigma(t^*)$.

(5) By definition of g, the term $g_{\forall\alpha.A}(y)(T^*, f_T, g_T)$ β-reduces to $g_A[T^*/\alpha, f_T/f_\alpha, g_T/g_\alpha](y)$, that is $\sigma(g_A)(y)$. By (3) and $\alpha \neq \mathrm{Tm}$ we conclude $\sigma(g_A)(y) = g_{A[T/\alpha]}(y)$. □

References

[1] A. Abel, Weak beta-normalization and normalization by evaluation for system F, in: Proceedings of Logic for Programming Artificial Intelligence and Reasoning'08, in: LNCS, vol. 5330, 2008, pp. 497–511.
[2] A. Abel, Typed applicative structures and normalization by evaluation for system F_ω, in: Proceedings of Computer Science Logic 2009, in: LNCS, vol. 5771, 2009, pp. 40–54.
[3] K. Aehlig, F. Joachimski, Operational aspects of untyped normalisation by evaluation, Math. Struct. Comput. Sci. 14 (4) (2004) 587–611.
[4] H.P. Barendregt, The Lambda Calculus: Its Syntax and Semantics, North Holland, Amsterdam, 1984, Appendix C.
[5] U. Berger, M. Eberl, H. Schwichtenberg, Normalisation by evaluation, in: B. Moller, J.V. Tucker (Eds.), Prospects for Hardware Foundations, 1998, pp. 117–137.
[6] F. Barbanera, S. Berardi, A full continuous model of polymorphism, Theoret. Comput. Sci. 290 (1) (2003) 407–428.
[7] S. Berardi, C. Berline, $\beta\eta$-complete models for system F, Math. Struct. Comput. Sci. 12 (6) (2002) 823–874.
[8] S. Berardi, C. Berline, Building continuous webbed models for system F, Theoret. Comput. Sci. 315 (1) (2004) 3–34.
[9] H. Friedman, Classically and intuitionistically provably recursive functions, in: D.S. Scott, G.H. Muller (Eds.), Higher Set Theory, in: Lecture Notes in Mathematics, vol. 699, 1978, pp. 21–28.
[10] Jean-Yves Girard, The system F of variable types, fifteen years later, Theoret. Comput. Sci. 45 (2) (1986) 159–192.
[11] T. Joly, Codage, separabilite et representation, These de doctorat, Universite de Paris VII, 2000. http://www.cs.ru.nl/~joly/these.ps.gz.
[12] F. Garillot, B. Werner, Simple types in type theory: deep and shallow encodings, in: Proceedings of Theorem Proving in Higher Order Logics 2007, in: LNCS, vol. 4732, 2007, pp. 368–382.
[13] P. Lescanne, Jocelyne Rouyer-Degli, Explicit substitutions with de Bruijn's levels, in: Proceedings of the 6th International Conference on Rewriting Techniques and Applications, in: Lecture Notes in Computer Science, vol. 914, 1995, pp. 294–308.
[14] G. Longo, E. Moggi, Constructive natural deduction and its 'omega-set' interpretation, Math. Struct. Comput. Sci. 1 (2) (1991) 215–254.
[15] J.C. Mitchell, Semantic models for second-order lambda calculus, in: Proceedings of 25th Annual Symposium on Foundations of Computer Science, 1984, pp. 289–299.
[16] E. Moggi, R. Statman, The maximum consistent theory of second order lambda calculus, Types mailing list, July 24, 1986. http://www.di.unito.it/~stefano/MoggiStatman1986.zip.
[17] F. Pfenning, C. Elliott, Higher-order abstract syntax, in: Proceedings of PLDI 88, ACM SIGPLAN Notices 23 (7) (1988) 199–208.
[18] F. Pfenning, P. Lee, LEAP: a language with eval and polymorphism, in: Proceedings of Theory and Practice of Software Development 1989, in: LNCS, vol. 352, 1989, pp. 345–359.
[19] V.B. Tannen, J.H. Gallier, Polymorphic rewriting conserves algebraic strong normalization and confluence, in: Proceedings of International Colloquium on Automata, Languages and Programming 1989, in: LNCS, vol. 372, 1989, pp. 137–150.