Under consideration for publication in Math. Struct. in Comp. Science

Intersection Types and λ-definability

ANTONIO BUCCIARELLI[1] ADOLFO PIPERNO[2†] and IVANO SALVO[3†]

[1]*PPS, Université Paris 7, 2 Place Jussieu, 75251 Paris Cedex 05, France.*
E-mail: buccia@pps.jussieu.fr.

[2]*Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza",*
Via Salaria 113, 00198 Roma, Italy. E-mail: piperno@dsi.uniroma1.it.

[3]*Dipartimento di Informatica, Università di Torino, C.so Svizzera 185, 10149 Torino, Italy.*
E-mail: salvo@di.unito.it.

Received 28 September 2001

This paper presents a novel method to compare computational properties of λ-terms typeable with intersection types, with respect to terms typeable with Curry types. We introduce a translation from intersection typing *derivations* to Curry typeable *terms* which is preserved by β-reduction: this allows to simulate a computation starting from a term typeable in the intersection discipline by means of a computation starting from a simply typeable term. Our approach proves strong normalization for the intersection system naturally by means of purely syntactical techniques. The paper extends the results presented in [Bucciarelli, De Lorenzis, Piperno, Salvo, *Some Computational Properties of Intersection Types*, LICS'99] to the whole intersection type system of Barendregt, Coppo and Dezani, thus providing a complete proof of a conjecture proposed by Leivant in 1990: all functions uniformly definable using intersection types are already definable using Curry types.

1. Introduction

The λ-calculus originates as a *type-free* theory of functions: every term may be considered either as a function or as an argument, and no syntactic restriction is imposed on function application. This makes the system powerful enough to represent all computable functions.

Types are syntactical objects assigned to pure terms in order to give a description of their functional behavior. The constraints imposed by types usually restrict expressiveness, since the set of legal (well typed) terms is in general a proper subset of untyped ones, and hence the set of representable functions is in general smaller than the set of computable ones.

In this paper, we compare function definability in intersection type systems with function definability in the simply typed lambda-calculus.

The simply typed lambda calculus ($\lambda_{\rightarrow}$) was introduced by Curry in (Cur34), while intersection types originate in works by Barendregt, Coppo and Dezani (CDC80; BCDC83; Sal78). From the point of view of the set of typeable terms, simple types are much less expressive than

† Partially supported by MURST Cofin TOSCA

intersection types. In particular, intersection types are able to type all untyped terms or, when the universal type is disallowed, all strongly normalizing ones. From here onwards, we will consider intersection types without the universal type.

The leitmotiv of our comparison of such type systems is a translation $|\cdot|^{\mathrm{D}}$, which permits us to mimic the computations of terms typeable in the intersection type discipline by means of the computations of Curry typeable terms. Such translation is defined on typing derivations in the *strict intersection type system* ($\lambda^S_\cap$), which has been introduced in (CDCV81), and has received a systematic treatment in (vB92; vB93). Although strict types are a proper subset of intersection types, they preserve, from the point of typeability, the expressive power of the whole system (vB93, §4.3).

More precisely, we will show that, for any term M typeable with strict intersection types, and for any of its typing derivations $\mathcal{D}$, there exists a term $\widehat{M}_{\mathcal{D}}$, which is typeable in the Curry system and which is able to "represent" the whole computation of M. In other words, the λ-calculus with intersection types can be embedded into the simply typed calculus. This will allow us to simulate all possible reductions starting from M by means of reductions of $\widehat{M}_{\mathcal{D}}$. Hence, using purely syntactic techniques, strong normalization and lambda definability in $\lambda^S_\cap$ are reduced to the same problems for Curry typeable terms.

The first result that we present is a new proof of the strong normalization property for intersection types. We recall here that there is a close relationship between the definability problem and the "difficulty" of a normalization proof in typed λ-calculi (see (FLO83, Sections 2 and 6)). Simply typed λ-calculus allows for normalization proofs which assign a decreasing metric to terms during reduction (Gan80b; Gan80a). On the other hand, normalization in polymorphic λ-calculi is usually proven using variants of the so-called *computability* technique ((Tai67)), which has a merely *semantical* nature (namely, it is not based on a metric approach): consider, as an example, Girard-Reynolds second order λ-calculus (Gir71; Rey74).

We will present a normalization proof for the λ-calculus with intersection types which only makes use of syntactical techniques, in that it reduces the strong normalization problem for intersection types to the case of Curry types. Different syntactical approaches and normalization proofs for λ-calculus with intersection types are (KW95) and (KP99).

After having discussed the normalization property, we compare simple and intersection types with respect to the problem of λ-definability. In such case, the relationship between the systems is not as clear as from the typeability perspective.

Intersection types have been proposed in the design of the type system of concrete programming languages, as an alternative to parametric polymorphism. An example is the language Forsyth proposed by Reynolds (Rey96b; Rey96a). Intersection types allow a form of *discrete polymorphism*, since the same variable can appear inside a term in a finite number of places where different functionalities are required. Observe that this kind of polymorphism is not to be confused with *overloading*, where computations vary according to types (CGL95).

However, as already observed by Leivant (Lei90), typings obtained in the intersection type discipline may be highly *non–uniform*. In particular, it may happen that a term M, representing an unary numeric function φ, needs to be typed with different types depending on its argument n (see Example 4.3). Type inference for intersection types is undecidable, since the typeability problem is equivalent to termination; also for decidable fragments (KW99), it appears quite unnatural to design a compiler which checks the functional behavior of a program statically, taking into account all possible inputs.

These considerations lead to a more natural notion of lambda definability in the presence of types, which requires that a term representing a function must be uniformly typed independently from its possible inputs.

Once we have imposed the uniformity condition, we emerge with the following scenario. The severe restrictions imposed by the structure of Curry types allow the simply typed λ-calculus to uniformly represent only a proper subset of *elementary functions*, a strict subset of total recursive ones. Even simple numeric functions, such as the predecessor function, cannot be represented (see (Sch76)). Indeed, the class of representable functions has been characterized in (Sch76; Sta79; Sta82; Zai91; Lei93). A first attempt to compare the expressiveness of simple and intersection types appears in (Lei90), where it was proved that functions uniformly representable in the intersection system are elementary, whereas all total computable functions are representable in a non-uniform way. In addition, starting from these results, Leivant conjectured that the class of functions uniformly representable in the intersection discipline coincides with the class of functions definable in the Curry system. The proof of this conjecture is the main achievement presented in the present paper.

Note that Leivant's results have a purely semantical nature, since the considered systems are compared by characterizing the class of definable functions. In contrast, we obtain our results using syntactical techniques only.

As already mentioned, we define an embedding which maps every typing derivation in the strict intersection type system to a Curry typeable term. In some sense, the term subject of the typing has the same computational behavior as the Curry typeable term obtained via a translation function $|\cdot|^{\mathrm{D}}$. Since we are able to map computations of terms typeable in $\lambda^{S}_{\cap}$ into computations of terms typeable in $\lambda_{\rightarrow}$, it is natural to ask whether our syntactic approach can be used to compare the expressive power of $\lambda_{\rightarrow}$ and $\lambda^{S}_{\cap}$ from the point of view of representable functions.

As a matter of fact, by translating a typing of a term which uniformly represents a numeric function φ, we obtain a Curry typeable term which represents φ modulo suitable coding of the arguments and decoding of the result. The structure of derivations typing Church numerals in the intersection system, and their translations, will be analyzed. Finally, we define Curry typeable terms which realize the aforementioned coding and the corresponding decoding, thus allowing a proof of Leivant's conjecture in the case of strict intersection types.

A preliminary paper presenting such results appeared in (BDLPS99). In this paper, we complete the proofs in (BDLPS99) and we extend the characterization to the full intersection type system of Baredregt-Coppo-Dezani ($\lambda^{BCD}_{\cap}$), removing the restrictions imposed by strict types. Such an extension is not straightforward, for two main reasons: (i) the system $\lambda^{BCD}_{\cap}$ is not syntax-directed, so that the translation of derivations cannot be adapted to it; (ii) the system $\lambda^{BCD}_{\cap}$ allows more typings to be derived, hence the uniformity condition must be completely re-analyzed. A key role in our final characterization is played by the η-rule of the λ-calculus, which enables us to fill the gap between the systems $\lambda^{BCD}_{\cap}$ and $\lambda^{S}_{\cap}$.

1.1. *Outline of the Paper*

The rest of the paper is organized as follows. In Section 2, we introduce firstly some basic terminology about type systems, and then simply typed lambda calculi (or Curry type system) and the strict intersection type system, together with some of their basic results.

We will then (Section 3) introduce a translation function which transforms a typing derivation in the intersection type assignment system into a term typeable with Curry types. We will then show that the translation is preserved by β-reduction. Using this fact, we will be able to give a simple, syntactic proof of the strong normalization property for the strict intersection type system that stems from strong normalization for the simply typed lambda calculus.

In Section 4, we analyze some pathologies of typings in the intersection type discipline. In particular, we show that some terms, representing functions in the untyped scenario, have to be typed with different types to be applied to a term representing n, for different n. Thus, a more natural definition of definability for typed lambda calculi is introduced. This definition, due to Leivant, requires that a term which *uniformly* represents a function is type independent from any particular input it has to be applied to. Leivant conjectured that intersection types do not increase the set of uniformly representable functions with respect to Curry types.

The translation defined in Section 3 maps a typing derivation in $\lambda_\cap^S$ for a term M to a Curry typeable term $\widehat{M}$. In Section 5, we argue about how to use $\widehat{M}$ to represent a numeric functions φ in $\lambda_\rightarrow$, when M uniformly represents φ in $\lambda_\cap^S$. We show that in a particular, but significant case, $\widehat{M}$ itself “almost represents” φ.

In Section 6, we show that $\widehat{M}$ represents computations over an unusual class of numerals: we characterize such numerals and obtain a general method for exploiting $\widehat{M}$ in order to represent φ by a Curry typeable term. This allow us to give a positive answer to Leivant’s conjecture in the case of strict intersection type system.

Finally, we analyze relationships between different intersection type systems, with respect to the problem of uniform definability of numeric functions, and extend our result to the system $\lambda_\cap^{BCD}$.

Some remarks and directions for further work conclude the paper.

2. The Type Systems

We assume the reader to be familiar with the basic definitions and properties of pure and typed lambda calculus, for which we refer to (Bar84) and (Bar92). In particular, Λ denotes the set of untyped λ-terms. Terms will be considered modulo α-equivalence, and the so-called *variable convention* will be assumed: bound variables are all distinct and different from free ones.

We start giving some general terminology and notations about typed lambda calculi.

Definition 2.1. Let λ_T be a typed λ-calculus:

— $Type_T$ denotes the set of types for λ_T. We use small greek letters for types, with the convention that α, β and γ denote type variables.

— $M{:}\tau$ is called a *statement*, where $M \in \Lambda$ is the *subject* and $\tau \in Type_T$ is the *predicate* of the statement.

— A *basis* is a partial function from term variables to types of λ_T. Sometimes it is convenient to

$$(\text{Var})_{\rightarrow} \qquad \frac{A(x)=\sigma}{A \vdash_{\rightarrow} x{:}\sigma}$$

$$(\rightarrow\text{I})_{\rightarrow} \qquad \frac{A \cup \{x{:}\sigma\} \vdash_{\rightarrow} M{:}\tau}{A \vdash_{\rightarrow} \lambda x.M{:}\sigma \rightarrow \tau}$$

$$(\rightarrow\text{E})_{\rightarrow} \qquad \frac{A \vdash_{\rightarrow} M{:}\sigma \rightarrow \tau \qquad A \vdash_{\rightarrow} N{:}\sigma}{A \vdash_{\rightarrow} (M\,N){:}\tau}$$

Fig. 1. The type assignment system of $\lambda_{\rightarrow}$

consider a basis as a set of statements where subjects are distinct variables. The set of bases is indicated by $Bases_T$. We use uppercase roman letters for bases.

— $B \vdash_T M{:}\tau$ is a *judgment* (or a *typing*). Judgments are derivable from axioms and rules of λ_T. If B is the empty basis, we write $\vdash_T M{:}\tau$ for $\{\} \vdash_T M{:}\tau$.
— A term M is *typeable* in λ_T if there exists a basis $B \in Bases_T$ and a type $\tau \in Type_T$ such that the judgment $B \vdash_T M{:}\tau$ is derivable in λ_T. The set of typeable terms is denoted by Λ_T.
— $\mathcal{D}[A \vdash_T M{:}\sigma]$ denotes a *typing derivation* in λ_T proving the typing $A \vdash_T M{:}\sigma$.
— The set of all typing derivations in λ_T will be denoted by Der_T.

2.1. *Lambda Calculus with Simple (or Curry) Types*

The simply typed λ-calculus originates from Church's work
(Chu40). We are interested in the *implicit typing* approach, introduced by Curry in (Cur34) for the theory of combinators. The system was adapted for the lambda calculus in (CF68).

Definition 2.2. *Simple (or Curry) types* are generated using the following grammar:

$$\sigma ::= \alpha \mid (\sigma \rightarrow \sigma), \tag{1}$$

where α ranges over a countable set of type variables. We call $Type_{\rightarrow}$ the set of types resulting from (1). As usual, the arrow type constructor, $\rightarrow$, associates to the right and hence $\sigma_1 \rightarrow \sigma_2 \rightarrow \cdots \rightarrow \sigma_n \rightarrow \tau$ is an abbreviation for $\sigma_1 \rightarrow (\sigma_2 \rightarrow (\cdots(\sigma_n \rightarrow \tau)\cdots))$.

Note that a type σ always has the shape $\sigma_1 \rightarrow \sigma_2 \rightarrow \cdots \rightarrow \sigma_n \rightarrow \alpha$, for some type variable α and $n \geq 0$.

Definition 2.3. In the *simply typed λ-calculus* $\lambda_{\rightarrow}$, judgments of the shape $A \vdash_{\rightarrow} M{:}\sigma$, derived from the rules in Fig. 1, are proven.

2.2. *Lambda Calculus With Strict Intersection Types*

A family of intersection type systems have been introduced in the literature, starting from the work of Coppo and Dezani in (CDC80). Our approach for comparing intersection type systems and Curry type system works for a syntax directed system, the *strict intersection type system*. We

$$(\text{Var})_s \qquad \frac{A(x) = \tau_1 \cap \ldots \cap \tau_n \qquad 1 \leq i \leq n}{A \vdash_s x{:}\tau_i}$$

$$(\to\text{I})_s \qquad \frac{A \cup \{x{:}\sigma\} \vdash_s M{:}\tau}{A \vdash_s \lambda x.M{:}\sigma \to \tau}$$

$$(\to\text{E})_s \qquad \frac{A \vdash_s M{:}(\tau_1 \cap \cdots \cap \tau_n) \to \tau \qquad A \vdash_s N{:}\tau_1 \ldots A \vdash_s N{:}\tau_n}{A \vdash_s MN{:}\tau}$$

Fig. 2. The type assignment system of $\lambda_\cap^S$

will introduce other intersection type disciplines and discuss the extension of our results to them in Section 7.

Following (vB93, Ch. 4), we define a restricted version of the intersection type assignment system of Coppo and Dezani (CDC80). It is based on a restricted set of types, in which intersections appear in the left-hand side of the arrow constructor only.

Definition 2.4. *Strict intersection types* are generated using the following grammar:

$$\begin{aligned} \sigma &::= \tau_1 \cap \cdots \cap \tau_n \;\; (n \geq 1) \\ \tau &::= \alpha \mid (\sigma \to \tau) \end{aligned} \qquad (2)$$

We call *strict types* (*Type*$_\cap^s$) the set of types resulting from (2) with start symbol τ, and *strict intersection types* (*Type*$_\cap^S$) the set of types originated with start symbol σ. Observe that strict types do not contain intersections as principal type constructor and that strict intersection types are just intersections of strict types.

Definition 2.5.

1 In $\lambda_\cap^S$, judgments of the kind $A \vdash_s M{:}\tau$, derived from the rules of Fig.2, are proven, where A contains statements of the shape $x{:}\sigma$, with $\sigma \in \mathit{Type}_\cap^S$, and $\tau \in \mathit{Type}_\cap^s$.
2 Moreover, the judgment $A \vdash_S M{:}\sigma$ is derived if and only if there exist types $\sigma_1, \ldots, \sigma_n$ such that $\sigma_1 \cap \ldots \cap \sigma_n = \sigma$ and, for all $1 \leq i \leq n$, the judgment $A \vdash_s M{:}\sigma_i$ is derivable.

Observe that intersections of types may appear as predicates in bases, only; types assigned to terms in derivations always belong to *Type*$_\cap^s$. An important property that distinguishes this system from other intersection type systems is that typing derivations are *syntax directed*, i.e. we can guess the last rule applied in a typing derivation just by looking at the syntactic structure of the subject. We exploit this fact in the definition of the translation introduced in Section 3 and in the proofs of its properties.

We end this section by stating basic properties of systems $\lambda_\to$ and $\lambda_\cap^S$ that will be referred to in the sequel.

Proposition 2.6. For the systems $\lambda_\to$ and $\lambda_\cap^S$, the following properties hold (let λ_T stand for both $\lambda_\to$ and $\lambda_\cap^S$):

1 *Basis Lemma*: If $B \vdash_T M{:}\tau$ then $B|_{FV(M)} \vdash_T M{:}\tau$.

2 *Subject Reduction*: If $B \vdash_T M{:}\tau$ and $M \xrightarrow[\beta]{*} M'$, then $B \vdash_T M{:}\tau$.

3. A Translation from $\lambda_\cap^S$ to $\lambda_\rightarrow$

The original proof of the fact that any term typeable in $\lambda_\cap^S$ is strongly normalizing relies on a computability argument. This is in sharp contrast with the case of $\lambda_\rightarrow$, where strong normalization can be proven by defining a (well founded) "measure" for typeable terms, which strictly decreases as reductions go on.

In this section, we introduce an embedding of $\lambda_\cap^S$ into $\lambda_\rightarrow$, which allows us to mimic any reduction path rooted in a term typeable in $\lambda_\cap^S$ with a (in general longer) reduction path rooted in a suitable simply typed term. An immediate corollary of this is a syntactic proof of strong normalization for $\lambda_\cap^S$. Moreover, since our embedding allows to represent in $\lambda_\rightarrow$ any "computation" feasible in $\lambda_\cap^S$, it provides a framework for studying λ-definability in these systems. This will be the subject of Section 4.

The mentioned embedding is based on a function, $|\cdot|^{\mathrm{D}}$, which associates to any typing derivation in $\lambda_\cap^S$ a pure λ-term. We prove that the image of such map is a subset of $\Lambda_\rightarrow$, and that the map commutes with respect to β-reduction. To obtain such results, we also define a translation of types, $|\cdot|^{\mathrm{T}}$, which maps strict intersection types to simple types, and a translation of bases, $|\cdot|^{\mathrm{B}}$, which maps Bases_S to $\mathit{Bases}_\rightarrow$.

Notation 3.1. In the next definitions, we use the following notational convention concerning variable names: we consider an injective function

$$f : \mathit{Var} \times \mathbb{N} \rightarrow \mathit{Var},$$

and, for any $x \in \mathit{Var}, n \in \mathbb{N}$, we write x^n for $f(x,n)$.

Definition 3.2. The functions

$$\begin{array}{ll} |\cdot|^{\mathrm{T}} : \mathit{Type}_\cap^s \rightarrow \mathit{Type}_\rightarrow & \textit{(translation of types)}, \\ |\cdot|^{\mathrm{B}} : \mathit{Bases}_S \rightarrow \mathit{Bases}_\rightarrow & \textit{(translation of bases)} \end{array}$$

are inductively defined as follows:

$$\begin{array}{rcl} |\alpha|^{\mathrm{T}} & = & \alpha \\ |(\sigma_1 \cap \cdots \cap \sigma_n) \rightarrow \tau|^{\mathrm{T}} & = & |\sigma_1|^{\mathrm{T}} \rightarrow \cdots \rightarrow |\sigma_n|^{\mathrm{T}} \rightarrow |\tau|^{\mathrm{T}} \\ \\ |\{\}|^{\mathrm{B}} & = & \{\} \\ |A \cup \{x{:}\sigma_1 \cap \cdots \cap \sigma_n\}|^{\mathrm{B}} & = & |A|^{\mathrm{B}} \cup \{x^1{:}\ |\sigma_1|^{\mathrm{T}}, \ldots, x^n{:}\ |\sigma_n|^{\mathrm{T}}\}. \end{array}$$

Note that the term variables $x^1, \ldots, x^n$ are *fresh* with respect to A, i.e. they do not appear in A.

Definition 3.3. Define the function $|\cdot|^{\mathrm{D}} : \mathit{Der}_S \rightarrow \Lambda$ *(translation of derivations)* inductively on the structure of typing derivations:

(i) $$\left| \frac{A(x) = \sigma_1 \cap \cdots \cap \sigma_n}{A \vdash_s x{:}\sigma_i} \right|^{\mathrm{D}} = x^i;$$

(ii) $$\left|\frac{\dfrac{\mathcal{D}_1}{A\cup\{x{:}\sigma_1\cap\cdots\cap\sigma_n\}\vdash_s M{:}\tau}}{A\vdash_s \lambda x.M{:}(\sigma_1\cap\cdots\cap\sigma_n)\to\tau}\right|^{\mathrm{D}} = \lambda x^1\ldots x^n.\left|\frac{\mathcal{D}_1}{A\cup\{x{:}\sigma_1\cap\cdots\cap\sigma_n\}\vdash_s M{:}\tau}\right|^{\mathrm{D}};$$

(iii) $$\left|\frac{\dfrac{\mathcal{D}_0}{A\vdash_s M{:}(\tau_1\cap\cdots\cap\tau_n)\to\tau}\quad\dfrac{\mathcal{D}_1}{A\vdash_s N{:}\tau_1}\quad\ldots\quad\dfrac{\mathcal{D}_n}{A\vdash_s N{:}\tau_n}}{A\vdash_s MN{:}\tau}\right|^{\mathrm{D}} =$$

$$\left|\frac{\mathcal{D}_0}{A\vdash_s M{:}(\tau_1\cap\cdots\cap\tau_n)\to\tau}\right|^{\mathrm{D}}\left|\frac{\mathcal{D}_1}{A\vdash_s N{:}\tau_1}\right|^{\mathrm{D}}\cdots\left|\frac{\mathcal{D}_n}{A\vdash_s N{:}\tau_n}\right|^{\mathrm{D}}.$$

Remark 3.4. Typing with strict intersection types is totally syntax-driven, by considering intersections as equivalent modulo permutations and repetitions of their components. This equivalence is also widely adopted in the literature on intersection types and relies on the intuitive set-interpretation. This is not the case of the present paper. The difference between $\sigma\cap\tau$ and $\tau\cap\sigma$ ($\sigma\cap\sigma$ and σ) becomes significant with respect to the translation in the previous definition. For instance, two derivations, assigning to $\lambda x.x$ the types $\sigma\cap\tau\to\sigma$ and $\tau\cap\sigma\to\sigma$, respectively, are mapped into different lambda terms, $\lambda x^1x^2.x^1$ and $\lambda x^1x^2.x^2$, respectively.

An expected property of the translation defined above is that it maps a typing derivation of $\lambda^{S}_{\cap}$ into a simply typeable term.

Lemma 3.5. For any derivation $\mathcal{D}$ of the typing $A\vdash_s M{:}\sigma$ in $\lambda^{S}_{\cap}$, we have that

$$|A|^{\mathrm{B}}\vdash_{\to}|\mathcal{D}|^{\mathrm{D}} : |\sigma|^{\mathrm{T}}$$

is a typing in $\lambda_{\to}$.

Proof. The proof proceeds by induction on the structure of the typing derivation. We have to consider three cases, depending on the last applied rule in $\mathcal{D}$.

Case 1 The last applied rule is $(\mathrm{Var})_s$. In this case, the derivation $\mathcal{D}$ has the shape:

$$\frac{A(x)=\sigma_1\cap\ldots\cap\sigma_n\qquad 1\le i\le n}{A\vdash_s x{:}\sigma_i}\ .$$

Then $\{x^1{:}\,|\sigma_1|^{\mathrm{T}},\ldots,x^n{:}\,|\sigma_n|^{\mathrm{T}}\}\subseteq|A|^{\mathrm{B}}$ and $|\mathcal{D}|^{\mathrm{D}}=x^i$. Hence, by rule $(\mathrm{Var})_{\to}$, we obtain

$$|A|^{\mathrm{B}}\vdash_{\to} x^i{:}\,|\sigma_i|^{\mathrm{T}}$$

and this case is settled.

Case 2 The last applied rule is $(\to\mathrm{E})_s$. In this case, the derivation $\mathcal{D}$ has the shape:

$$\frac{\mathcal{D}_0\quad\mathcal{D}_1\ \cdots\ \mathcal{D}_n}{A\vdash_s MN{:}\tau}$$

where

$$\mathcal{D}_0=\frac{\mathcal{D}_0'}{A\vdash_s M{:}(\tau_1\cap\cdots\cap\tau_n)\to\tau}\quad\mathcal{D}_1=\frac{\mathcal{D}_1'}{A\vdash_s N{:}\tau_1}\ \ldots\ \mathcal{D}_n=\frac{\mathcal{D}_n'}{A\vdash_s N{:}\tau_n}\ .$$

By the induction hypothesis, we have:

$$|A|^{\mathrm{B}} \vdash_{\rightarrow} |\mathcal{D}_0|^{\mathrm{D}} : |\tau_1|^{\mathrm{T}} \rightarrow \cdots \rightarrow |\tau_n|^{\mathrm{T}} \rightarrow |\tau|^{\mathrm{T}},$$
$$|A|^{\mathrm{B}} \vdash_{\rightarrow} |\mathcal{D}_i|^{\mathrm{D}} : |\tau_i|^{\mathrm{T}}, \text{ for } 1 \leq i \leq n.$$

By definition of $|\cdot|^{\mathrm{D}}$, we have:

$$|\mathcal{D}|^{\mathrm{D}} = |\mathcal{D}_0|^{\mathrm{D}} |\mathcal{D}_1|^{\mathrm{D}} \ldots |\mathcal{D}_n|^{\mathrm{D}},$$

and this case is settled, applying n times the rule $(\rightarrow\mathrm{E})_{\rightarrow}$.

Case 3 The last applied rule is $(\rightarrow\mathrm{I})_s$. In this case, the derivation $\mathcal{D}$ has the shape:

$$\frac{\mathcal{D}'}{A \vdash_s \lambda x.M : (\sigma_1 \cap \cdots \cap \sigma_n) \rightarrow \tau}$$

where, for some derivation $\mathcal{D}''$, the shape of $\mathcal{D}'$ is

$$\frac{\mathcal{D}''}{A \cup \{x : \sigma_1 \cap \cdots \cap \sigma_n\} \vdash_s M : \tau} \ .$$

By the induction hypothesis, we have:

$$|A|^{\mathrm{B}} \cup \{x^1 : |\sigma_1|^{\mathrm{T}}, \ldots, x^n : |\sigma_n|^{\mathrm{T}}\} \vdash_{\rightarrow} |\mathcal{D}'|^{\mathrm{D}} : |\tau|^{\mathrm{T}}$$

so that, applying n times the rule $(\rightarrow\mathrm{I})_{\rightarrow}$,

$$|A|^{\mathrm{B}} \vdash_{\rightarrow} \lambda x^1 \ldots x^n. |\mathcal{D}'|^{\mathrm{D}} : |\sigma_1|^{\mathrm{T}} \rightarrow \cdots \rightarrow |\sigma_n|^{\mathrm{T}} \rightarrow |\tau|^{\mathrm{T}} \ .$$

The case is settled by observing that

$$|\mathcal{D}|^{\mathrm{D}} = \lambda x^1 \ldots x^n. |\mathcal{D}'|^{\mathrm{D}}$$

and

$$|(\sigma_1 \cap \cdots \cap \sigma_n) \rightarrow \tau)|^{\mathrm{T}} = |\sigma_1|^{\mathrm{T}} \rightarrow \cdots \rightarrow |\sigma_n|^{\mathrm{T}} \rightarrow |\tau|^{\mathrm{T}} \ .$$

$\square$

A crucial property of the presented translation is that it enjoys a commutation property with respect to β-reduction.

Lemma 3.6. Let $\mathcal{D}$ be a derivation of the typing $A \vdash_s M : \tau$ in $\lambda^S_\cap$ and let $M \xrightarrow[\beta]{} N$. Then there exists a derivation $\mathcal{D}'$ of the typing $A \vdash_s N : \tau$ such that[†]:

$$|\mathcal{D}|^{\mathrm{D}} \xrightarrow[\beta]{+} |\mathcal{D}'|^{\mathrm{D}} \ .$$

Proof. The proof proceeds by induction on the structure of the derivation $\mathcal{D}$. The only interesting case is when the last rule applied in $\mathcal{D}$ is $(\rightarrow\mathrm{E})_s$, and the subject of the judgment is the contracted redex. The other cases are settled by straightforward applications of induction

[†] Observe that this statement implies the subject reduction property.

hypothesis. In the considered case, $\mathcal{D}$ has the shape:

$$\dfrac{\dfrac{\mathcal{D}_0}{A \vdash_s \lambda x.P\colon \tau_1 \cap \cdots \cap \tau_n \to \tau} \quad \dfrac{\mathcal{D}_1}{A \vdash_s Q\colon \tau_1} \;\cdots\; \dfrac{\mathcal{D}_n}{A \vdash_s Q\colon \tau_n}}{A \vdash_s (\lambda x.P)Q\colon \tau} \tag{3}$$

The variable x appears in $\mathcal{D}_0$ as the subject of q instances (for some $q \geq 0$) of the rule $(\text{Var})_s$. Hence, a set $\mathsf{Pairs}(\mathcal{D},x)$ can be defined as follows, where $1 \leq j \leq q$:

$$(i,j) \in \mathsf{Pairs}(\mathcal{D},x) \;\Leftrightarrow\; \dfrac{A_j(x) = \tau_1 \cap \cdots \cap \tau_n}{A_j \vdash_s x\colon \tau_i} \text{ appears in } \mathcal{D}.$$

Clearly, for every $j \in \{1,\ldots,q\}$, $A_j \supseteq A$. By the *Basis Lemma* (Proposition 2.6), for any $(i,j) \in \mathsf{Pairs}(\mathcal{D},x)$, a derivation $\mathcal{D}_i^j$ of the typing $A_j \vdash_s Q\colon \tau_i$ is built by replacing in $\mathcal{D}_i$ every occurrence of a basis B with $B \cup A_j$. It follows that the derivation $\mathcal{D}'$ is obtained out of $\mathcal{D}_0$ in two steps:

1 replacing every occurrence of

$$\dfrac{A_j(x) = \tau_1 \cap \cdots \cap \tau_n}{A_j \vdash_s x\colon \tau_i}$$

with $\mathcal{D}_i^j$;

2 replacing every occurrence of the variable x with Q, in the subjects of the derivation obtained from $\mathcal{D}_0$ by applying step 1.

It turns out that $\mathcal{D}'$ is a derivation of the typing

$$A \vdash_s P[x := Q] : \tau.$$

Moreover, by a straightforward induction on the structure of $\mathcal{D}_0$, observing that

$$\forall i,j.(i,j) \in \mathsf{Pairs}(\mathcal{D},x) \;\Rightarrow\; |\,\mathcal{D}_i^j\,|^{\mathrm{D}} = |\,\mathcal{D}_i\,|^{\mathrm{D}},$$

the following holds:

$$|\,\mathcal{D}'\,|^{\mathrm{D}} = |\,\mathcal{D}_0\,|^{\mathrm{D}}\,[x^1 := |\,\mathcal{D}_1\,|^{\mathrm{D}},\ldots,x^n := |\,\mathcal{D}_n\,|^{\mathrm{D}}]. \tag{4}$$

Now,

$$\begin{aligned}
|\,\mathcal{D}\,|^{\mathrm{D}} &= \left|\dfrac{\mathcal{D}_0}{A \vdash_s \lambda x.P\colon (\tau_1 \cap \cdots \cap \tau_n) \to \tau}\right|^{\mathrm{D}} |\,\mathcal{D}_1\,|^{\mathrm{D}} \ldots |\,\mathcal{D}_n\,|^{\mathrm{D}} \\
&= (\lambda x^1 \ldots x^n.\,|\,\mathcal{D}_0\,|^{\mathrm{D}})\,|\,\mathcal{D}_1\,|^{\mathrm{D}} \ldots |\,\mathcal{D}_n\,|^{\mathrm{D}} \\
&\xrightarrow[\beta]{+} |\,\mathcal{D}_0\,|^{\mathrm{D}}\,[x^1 := |\,\mathcal{D}_1\,|^{\mathrm{D}},\ldots,x^n := |\,\mathcal{D}_n\,|^{\mathrm{D}}] \\
&= |\,\mathcal{D}'\,|^{\mathrm{D}}\text{, by (4),}
\end{aligned}$$

which proves the lemma. □

It is easy to see that the commutation property holds for the reflexive and transitive closure of $\xrightarrow[\beta]{}$, too.

Lemma 3.7. Let $\mathcal{D}$ be a derivation of the typing $B \vdash_s M : \sigma$ in $\lambda^S_\cap$, and let $M \xrightarrow[\beta]{+} N$. Then there exists a derivation $\mathcal{D}'$ of the typing $B \vdash_s N : \sigma$ such that

$$|\,\mathcal{D}\,|^{\mathrm{D}} \xrightarrow[\beta]{+} |\,\mathcal{D}'\,|^{\mathrm{D}} .$$

Proof. The proof proceeds by induction on the length of the reduction of $M \xrightarrow[\beta]{+} N$, using Lemma 3.6. □

We are now able to prove strong normalization in $\lambda^S_\cap$, reducing it to strong normalization in $\lambda_\rightarrow$.

Theorem 3.8 (Strong Normalization for $\lambda^S_\cap$). For any $M \in \Lambda$, if M is typeable in $\lambda^S_\cap$, then every β-reduction path starting from M is finite.

Proof. If M is typeable in $\lambda^S_\cap$, then there exists a basis B and a type σ such that $B \vdash_s M : \sigma$ is derivable. Let $\mathcal{D}$ be a derivation of such typing. If M has an infinite β–reduction path, then, by Lemma 3.7, $|\,\mathcal{D}\,|^{\mathrm{D}}$ also has an infinite β-reduction path, and by Lemma 3.5, $|\,B\,|^{\mathrm{B}} \vdash_\rightarrow |\,\mathcal{D}\,|^{\mathrm{D}} : |\,\sigma\,|^{\mathrm{T}}$, so that $|\,\mathcal{D}\,|^{\mathrm{D}}$ has a typing in $\lambda_\rightarrow$, hence it is strongly normalizing, a contradiction. □

4. Lambda Definability

In this section, we discuss lambda definability in typed lambda calculi. We refer to (Bar84, Ch. 6) for basic definitions of numeral system and lambda definability in the untyped lambda-calculus. For the sake of simplicity, we focus on unary functions; every definition and result can be easily extended to the case of functions with $k > 1$ arguments.

In this work, we consider the numeral system introduced by Church.

Definition 4.1 (CHURCH NUMERALS). The Church numeral c_n is the lambda term:

$$\lambda pq.p^n q.$$

Definition 4.2 (NUMERAL TYPES). We say that $\tau \in \mathit{Type}_T$ is a *(Church) numeral type* if there exists $n \in \mathbb{N}$ such that τ can be assigned to c_n in λ_T. We say that τ is a *full (Church) numeral type* if τ can be assigned to all Church numerals in λ_T.

Church numerals can be uniformly typed in $\lambda_\rightarrow$. Each Church numeral can be typed with an instance of the principal type of Church numerals in the Curry type system, $(\gamma \rightarrow \gamma) \rightarrow \gamma \rightarrow \gamma$. We will write $\mathbf{N}[\gamma]$ as an abbreviation for such type.

Observe that Church numerals are essentially *iterators*. In Example 5.3, we use this fact to define the exponential function as the iteration of multiplication and addition.

4.1. *Lambda Definability in Type Systems*

From the definition of function representation given in the previous subsection, it is easy to show that there are functions representable in $\lambda^S_\cap$ which are not representable in $\lambda_\rightarrow$. In particular, using the fact that all strongly normalizing terms are typeable in $\lambda^S_\cap$, one can show that all total computable functions are definable in $\lambda_\cap^S$, whereas only a subset of elementary functions are representable in $\lambda_\rightarrow$. We consider the following example.

Example 4.3. Let $E \equiv \lambda x.x(\lambda y.yx)x$. For any Church numeral c_n, $E\mathsf{c}_n$ reduces to $N \equiv \underbrace{\mathsf{c}_n \dots \mathsf{c}_n}_{n+1}$, hence providing the Church numeral $\mathsf{c}_{\varphi(n)}$, where:

$$\varphi(n) = \left.n^{n^{\cdot^{\cdot^{n}}}}\right\} n \text{ times} \quad .$$

Thus E computes a non–elementary function; moreover, for all n, $E\mathsf{c}_n$ is typeable in $\lambda_\cap^S$, since it is strongly normalizing. We observe that the typing of $E\mathsf{c}_n$ depends on n, as is clear from the structure of N. This example shows that terms which represent functions may have highly "non–uniform" typings in $\lambda_\cap^S$, depending on its arguments.

It is interesting to investigate the set of representable functions under a reasonable uniformity condition on typings of $M\mathsf{c}_n$, where M represents a numeric function (Lei90): intuitively, we require that there exist types σ and τ such that M is typeable with $\sigma \rightarrow \tau$, and, for all n, c_n is typeable with σ.

Definition 4.4 (Uniform Representation of Functions I). Let λ_T be a typed lambda calculus and $\varphi : \mathbb{N} \rightarrow \mathbb{N}$ a partial numeric function. We say that a lambda term M represents φ *uniformly* in λ_T, if there are types σ and τ such that:

1 for all $n \in \mathbb{N}$, such that $\varphi(n)$ is defined, $M\mathsf{c}_n \xrightarrow[\beta]{*} \mathsf{c}_{\varphi(n)}$;
2 σ is a full numeral type in λ_T;
3 the judgment $\{x{:}\sigma\} \vdash_T Mx{:}\tau$ is derivable in λ_T.

We call σ the *input type*, and τ the *output type* of M.

If σ and τ are equal, we say that M represents φ *strictly* in λ_T.

For the type systems we are interested in, the following definition of uniform function representation is equivalent.

Definition 4.5 (Uniform Representation of Functions II). Let λ_T be a typed lambda calculus and $\varphi : \mathbb{N} \rightarrow \mathbb{N}$ a partial numeric function. A lambda term M uniformly represents φ, if M is a closed term of the shape $\lambda x.M'$ for some M', and the judgment $\vdash_T M{:}\sigma \rightarrow \tau$ is derivable in λ_T, with σ a full numeral type in λ_T.

In the rest of the paper, we will use Definition 4.4 or Definition 4.5 up to convenience. Since we consider type systems enjoying the subject reduction property, from the assumption that $M\mathsf{c}_n \xrightarrow[\beta]{*} \mathsf{c}_{\varphi(n)}$, we have that τ is a numeral type assignable, at least, to each Church numeral representing a natural number in the range of φ.

In (Lei90), Leivant proved that all functions uniformly representable in $\lambda_\cap^S$ are elementary, as is the case for $\lambda_\rightarrow$. Moreover, the argument showing that subtraction is not uniformly representable in $\lambda_\rightarrow$ seems to apply also to $\lambda_\cap^S$. Therefore, Leivant proposed the following conjecture.

Conjecture 4.6 (Leivant 1990). Functions uniformly (resp. strictly) representable in λ_S are already uniformly (resp. strictly) representable in $\lambda_\rightarrow$.

The rest of the paper is devoted to prove Leivant's conjecture in the case of strict intersection types and to extend the result to other intersection type disciplines.

5. A Technical Description of the Syntactical Approach

In this section, we discuss how our translation function of Section 3 can be used to compare the sets of representable functions in $\lambda_{\rightarrow}$ and $\lambda_{\cap}^{S}$. Our syntactic approach differs strongly from previous attempts to solve Leivant's conjecture, which were based on semantic characterizations of the set of representable functions in $\lambda_{\rightarrow}$ and $\lambda_{\cap}^{S}$. Indeed, for every term M which represents a numeric function in $\lambda_{\cap}^{S}$, we exploit our $\lambda_{\cap}^{S}$ embedding into $\lambda_{\rightarrow}$, to construct a Curry typeable term which represents the same function.

First we describe the general approach, then we discuss a restricted, yet meaningful case, and finally we present an example. In the next section, we prove formally Leivant's conjecture.

5.1. *General Description of the Approach*

Let $\varphi : \mathbb{N} \rightarrow \mathbb{N}$ be a numeric function uniformly represented in $\lambda_{\cap}^{S}$, say by a term $M \equiv \lambda x.M'$. Therefore, a type assignable to M in $\lambda_{\cap}^{S}$ has the shape $\tau_1 \cap \ldots \cap \tau_k \rightarrow \tau_0$.

Moreover, it follows from the uniformity condition that $\tau_1, \ldots, \tau_k$ are full numeral types in $\lambda_{\cap}^{S}$ and, by Subject Reduction property, we have that any Church numeral c_m such that $\varphi(n) = m$, for some n, can be typed with τ_0.

Let $\mathcal{D}$ be a derivation of the judgment $\vdash_S M : \tau_1 \cap \ldots \cap \tau_k \rightarrow \tau_0$, and define

$$\widehat{M} \equiv \mid \mathcal{D}[\vdash_S M : \tau_1 \cap \ldots \cap \tau_k \rightarrow \tau_0] \mid^{\mathrm{D}} . \tag{5}$$

Since M represents φ uniformly, and using the definition of the translation function $\mid \cdot \mid^{\mathrm{D}}$, we have that for all $n \in \mathbb{N}$ and for all $\mathcal{D}_n^1[\vdash_S \mathsf{c}_n : \tau_1], \ldots, \mathcal{D}_n^k[\vdash_S \mathsf{c}_n : \tau_k]$ there exists a derivation $\mathcal{D}_n[\vdash_S M\mathsf{c}_n : \tau_0]$ such that:

$$\mid \mathcal{D}_n \mid^{\mathrm{D}} = \widehat{M} \mid \mathcal{D}_n^1 \mid^{\mathrm{D}} \ldots \mid \mathcal{D}_n^k \mid^{\mathrm{D}} . \tag{6}$$

Our aim is to use the term $\widehat{M}$ in order to construct a term representing uniformly φ in $\lambda_{\rightarrow}$, since we know, by Lemma 3.7, that:

$$\widehat{M} \mid \mathcal{D}_n^1 \mid^{\mathrm{D}} \ldots \mid \mathcal{D}_n^k \mid^{\mathrm{D}} \xrightarrow[\beta]{*} \mid \mathcal{D}'[\vdash_S \mathsf{c}_{\varphi(n)}] \mid^{\mathrm{D}} \equiv \widehat{\mathsf{c}_{\varphi(n)}}$$

for some derivation $\mathcal{D}'$.

However, the simply typed term $\widehat{M}$ is not a representation of φ in general: indeed, in (6), $\widehat{M}$ needs k arguments which may not be Church numerals; moreover, also $\widehat{\mathsf{c}_{\varphi(n)}}$ may not be a Church numeral.

To overcome such problems, we will find suitable simply typeable terms,

$$D_{[\tau]}, E_{[\tau_1]}, \ldots, E_{[\tau_k]},$$

such that:

$$\begin{array}{lll}\text{“encoders”:} & E_{[\tau_i]}\mathsf{c}_n \xrightarrow[\beta]{*} |\,\mathcal{D}[\vdash_S \mathsf{c}_n : \tau_i]\,|^{\mathrm{D}} & \text{for some } \mathcal{D}, \\ \text{“decoder”:} & D_{[\tau]}(|\,\mathcal{D}[\vdash_S \mathsf{c}_n : \tau_0]\,|^{\mathrm{D}}) \xrightarrow[\beta]{*} \mathsf{c}_n & \text{for all } \mathcal{D},\end{array}$$

in such a way that the term

$$\lambda x.D_{[\tau]}(\widehat{M}(E_{[\tau_1]}x)\ldots(E_{[\tau_k]}x)) \tag{7}$$

is Curry typeable, and, for all n, reduces to $\mathsf{c}_{\varphi(n)}$. As we will see, types of encoders $E_{[\tau_i]}$ depend on the type $|\,\tau_i\,|^{\mathrm{T}}$, and therefore the term in (7) cannot be typeable in $\lambda_{\rightarrow}$, because we need to give different types to the variable x. We will therefore introduce a Curry typeable encoder $E_{[\tau_1,\ldots,\tau_k,\tau]}$, such that:

$$E_{[\tau_1,\ldots,\tau_k,\tau]}\mathsf{c}_n \xrightarrow[\beta]{*} \lambda z.z\widehat{\mathsf{c}_n}^{1}\ldots\widehat{\mathsf{c}_n}^{k},$$

where $\widehat{\mathsf{c}_n}^{i} = |\,\mathcal{D}^i_n[\vdash_S \mathsf{c}_n : \tau_i]\,|^{\mathrm{D}}$ for some derivation $\mathcal{D}^i_n$. Hence we have that the term:

$$P = D_{[\tau]}(E_{[\tau_1,\ldots,\tau_k,\tau]}\mathsf{c}_n\widehat{M}) \tag{8}$$

reduces as follows:

$$P \xrightarrow[\beta]{*} D_{[\tau]}(\widehat{M}\widehat{\mathsf{c}_n}^{1}\ldots\widehat{\mathsf{c}_n}^{k}) \xrightarrow[\beta]{*} D_{[\tau]}\widehat{\mathsf{c}_{\varphi(n)}} \xrightarrow[\beta]{*} \mathsf{c}_{\varphi(n)}.$$

Note that the terms $D_{[\tau]}$ and $E_{[\tau_1,\ldots,\tau_k,\tau]}$ will be proven to have simple types. In our notation, they are indexed over intersection types, because their construction depends on intersection types.

We will build terms satisfying all the mentioned requirements. For the sake of clarity, we start with a simple case.

5.2. *A Strengthened Uniformity Condition*

Since Church numerals are essentially iterators, it is interesting to consider the significant case in which $\tau_1,\ldots,\tau_k,\tau_0$ are instances of the principal simple type of Church numerals, $(\alpha\rightarrow\alpha)\rightarrow\alpha\rightarrow\alpha$.

Fact 5.1. If $\tau = (\tau'\rightarrow\tau')\rightarrow\tau'\rightarrow\tau' \in \mathit{Type}^s_{\cap}$, then

$$|\,\mathcal{D}[\vdash_S \mathsf{c}_n : \tau]\,|^{\mathrm{D}} \equiv \mathsf{c}_n.$$

Proof. Merely observe that, since τ' is not an intersection, in any derivation of $\vdash_S \lambda pq.p^n q : \tau$ we use the statements $p : \tau'\rightarrow\tau'$ and $q : \tau'$, and hence the translation $|\cdot|^{\mathrm{D}}$ does not generate new variables. $\square$

Using this fact, the definition of $|\cdot|^{\mathrm{D}}$, and Lemma 3.7, we obtain the following.

Proposition 5.2. Let M be a term that uniformly represents $\varphi : \mathbb{N}\rightarrow\mathbb{N}$ in $\lambda^S_{\cap}$, with type $\tau_1\cap\ldots\cap\tau_k\rightarrow\tau_0$, and let $\tau_1,\ldots,\tau_k,\tau_0$ be instances of $(\alpha\rightarrow\alpha)\rightarrow\alpha\rightarrow\alpha$. Then the term

$$\widehat{M} \equiv |\,\mathcal{D}[\vdash_S M : (\tau_1\cap\ldots\cap\tau_k)\rightarrow\tau_0]\,|^{\mathrm{D}}$$

is such that, for all $n \in \mathbb{N}$,

$$\widehat{M}\underbrace{\mathsf{c}_n \dots \mathsf{c}_n}_{k} = \mathsf{c}_{\varphi(n)}.$$

thatuniformly represents in $\lambda_\rightarrow$ the

In comes out that $\widehat{M}$ needs $k > 1$ copies of c_n to compute $\varphi(n)$. Of course, the function φ could be uniformly represented in $\lambda_\rightarrow$ by a term, totally unrelated to $\widehat{M}$, which does not require k copies of the input. This is shown by the following example.

Example 5.3. The function $\varphi(x) = x^x$ is representable in $\lambda^S_\cap$ by the term $\omega = \lambda x.xx$, typeable with $\sigma = ((\rho \to \tau) \cap \rho) \to \tau$, for arbitrary types ρ and τ. The term ω is the typical example of a non Curry typeable term (the restrictions imposed by Curry types prevent self application). The translation $|\,(\mathcal{D}[\vdash_S \omega : \sigma])\,|^{\mathsf{D}}$ gives the term $\lambda xy.xy$ which represents the binary exponential function in $\lambda_\rightarrow$. We now show a term typeable in $\lambda_\rightarrow$ that represents φ. Let $\tau_0 = (o \to o) \to (o \to o)$ and $\tau_{i+1} = \tau_i \to \tau_i$. Observe that $\tau_0 = \mathbf{N}(o)$, $\tau_1 = \mathbf{N}(o \to o)$ and $\tau_{n+2} = \mathbf{N}(\tau_n)$. Consider the terms A (typed with $\tau_0 \to \tau_0 \to \tau_0$) and M (typed with $\tau_2 \to \tau_0 \to \tau_0$), which respectively compute addition and multiplication on natural numbers (we write types used in the derivation as superscript, to increase readability):

$$A = \lambda x^{\tau_0} y^{\tau_0} p^{o \to o} q^{o}.xp(ypq), \qquad M = \lambda x^{\tau_2} y^{\tau_0}.x(Ay)\mathsf{c}_0.$$

Then the term:

$$E = \lambda x^{\tau_2}.x(Mx)\mathsf{c}_1$$

computes the unary exponential function. As this example shows, strict intersection types add expressive power at least in the sense of compact representation of functions.

Example 5.4. In order to anticipate the general techniques of the next sections, we show another Curry typeable term which computes φ, where we use the term $\lambda xy.xy$ that we obtain from the translation. Observe that we can type the successor function $\mathbf{S}$ with both $\tau_0 \to \tau_0$ and $\tau_1 \to \tau_1$, and the Church numeral c_0 with both τ_0 and τ_1. As a consequence, the pair $Z^\star = <\mathsf{c}_0, \mathsf{c}_0>$ can be typed with $(\tau_1 \to \tau_0 \to \gamma) \to \gamma$, for an arbitrary type γ. The term:

$$S^\star = \lambda z^{(\tau_1 \to \tau_0 \to \gamma) \to \gamma} w^{\tau_1 \to \tau_0 \to \gamma}.z(\lambda y_1^{\tau_1} y_2^{\tau_0}.w(Sy_1)(Sy_2))$$

maps a pair of Church numerals, $<\mathsf{c}_n, \mathsf{c}_m>$, to the pair $<\mathsf{c}_{n+1}, \mathsf{c}_{m+1}>$. Following types indicated inside terms, it easy to see that $Z^\star$ can be typed with $\tau_1 \times \tau_0$ and $S^\star$ with $(\tau_1 \times \tau_0) \to (\tau_1 \times \tau_0)$; hence we can iterate them. Choosing $\gamma = \tau_0$, the term:

$$\lambda x^{\mathbf{N}(\tau_1 \times \tau_0)}.(xS^\star Z^\star)(\lambda xy.xy)$$

is typeable in $\lambda_\rightarrow$ with the type $\mathbf{N}(\tau_1 \times \tau_0) \to \tau_0$, and represents the function $\varphi(x) = x^x$.

6. A Proof of Leivant's Conjecture

In the Curry system, types of Church numerals are instances of the *principal type scheme* $(\alpha \to \alpha) \to \alpha \to \alpha$. This is no longer the case in $\lambda^S_\cap$. However, we can analyze structural properties

$$\dfrac{\dfrac{\dfrac{\begin{matrix}A(x)=\mu_1\cap\cdots\cap\mu_\ell\\ \begin{pmatrix}\text{and } \mu_i\equiv\mu^i_1\cap\ldots\cap\mu^i_{k_i}\to\mu^i_0\\ \text{for some } 1\le i\le\ell\end{pmatrix}\end{matrix}}{A\vdash_s x{:}\,\mu^i_1\cap\ldots\cap\mu^i_{k_i}\to\mu^i_0} \qquad \dfrac{\mathcal{D}_1}{A\vdash_s x^{n-1}y{:}\,\mu^i_1} \quad \ldots \quad \dfrac{\mathcal{D}_{k_i}}{A\vdash_s x^{n-1}y{:}\,\mu^i_{k_i}}}{\dfrac{A\vdash_s x^ny{:}\,\tau'\ (\equiv\mu^i_0)}{B\vdash_s \lambda y.x^ny{:}\,\rho\to\tau'}}}{C\vdash_s\lambda xy.x^ny{:}\,\tau\ (\equiv\mu\to\rho\to\tau')}}{}$$

where

$B=C\cup\{x{:}\,\mu\equiv\mu_1\cap\cdots\cap\mu_\ell\}$ and, for $1\le j\le\ell,\mu_i\equiv\mu^j_1\cap\ldots\cap\mu^j_{k_j}\to\mu^j_0$;
$A=B\cup\{y{:}\,\rho\equiv\rho_1\cap\cdots\cap\rho_m\}$;

Fig. 3. Type structure of Church numerals

of typing derivations (vB92, §4.1.5) in order to characterize the shape of a type τ which can be assigned in $\lambda^S_\cap$ to a Church numeral. Indeed, Figure 3 shows that

$$\tau\equiv\mu\to\rho\to\tau',$$

where $\mu\equiv\mu_1\cap\ldots\cap\mu_\ell$ and $\rho\equiv\rho_1\cap\ldots\cap\rho_m$, for some $\ell,m\in\mathbb{N}$. Moreover, every type μ_i $(1\le i\le\ell)$ has the shape $\mu^i_1\cap\ldots\cap\mu^i_{k_i}\to\mu^i_0$.

Our first goal is to characterize the shape of terms produced by the translation of typing derivations of Church numerals, since such terms are not, in general, Church numerals themselves.

Definition 6.1. Let $\tau\equiv\mu\to\rho\to\tau'$ be a Church numeral type in $\lambda^S_\cap$, where

$$\mu\equiv\mu_1\cap\ldots\cap\mu_\ell,\rho\equiv\rho_1\cap\ldots\cap\rho_m \text{ and } \mu_i\equiv\mu^i_1\cap\ldots\cap\mu^i_{k_i}\to\mu^i_0\ (1\le i\le\ell).$$

We inductively define a family of sets of strict intersection types:

$$\begin{array}{rcl} T^0_\tau &=& \{\rho_1,\ldots,\rho_m\},\\ T^{h+1}_\tau &=& \bigcup_{1\le i\le\ell}\{\mu^i_0 \mid \forall s\in\{1\ldots,k_i\}.\mu^i_s\in T^h_\tau\}.\end{array}$$

Moreover, we define $T_\tau=\bigcup_{n\in\mathbb{N}}T^n_\tau$.

We use the next example to clarify the intended meaning of Definition 6.1.

Example 6.2. Let $\tau\equiv\mu\to\rho\to\beta$, where

$$\mu\equiv(\alpha\cap\beta\to\beta)\cap(\gamma\cap\beta\to\beta)\cap(\alpha\to\gamma)\cap(\gamma\to\alpha) \text{ and } \rho\equiv\alpha\cap\beta.$$

We have

$$\begin{array}{llll} T^0_\tau=\{\alpha,\beta\} & T^1_\tau=\{\beta,\gamma\} & T^2_\tau=\{\alpha,\beta\} & T^3_\tau=\{\beta,\gamma\}\\ \cdots & T^{2k-1}_\tau=\{\beta,\gamma\} & T^{2k}_\tau=\{\alpha,\beta\} & \cdots\end{array}$$

Therefore $T_\tau=\{\alpha,\beta,\gamma\}$.

A straightforward induction shows that T^n_τ is exactly the set of types which can be assigned in $\lambda^S_\cap$ to the term p^nq (the *body* of the Church numeral c_n) with basis $\{p{:}\,\mu,q{:}\,\rho\}$. Therefore, if a given type σ belongs to T^n_τ for all n, then $\mu\to\rho\to\sigma$ is a full numeral type. We observe that, given the general structure of Church numeral types (Figure 3), we have

$$\forall n. T^n_\tau \subseteq \{\mu^1_0, \dots, \mu^\ell_0, \rho_1, \dots, \rho_m\}.$$

In this example, the typing

$$\{p{:}\,\mu, q{:}\,\rho\} \vdash_S p^n q{:}\,\beta \tag{9}$$

is derivable for all n. But T_τ contains types α, γ such that the typings

$$\{p{:}\,\mu, q{:}\,\rho\} \vdash_S p^n q{:}\,\alpha \text{ and } \{p{:}\,\mu, q{:}\,\rho\} \vdash_S p^n q{:}\,\gamma$$

are *not* derivable for every n. More precisely, in order to obtain the typing (9), for some $n > 0$, either the typing $\{p{:}\,\mu, q{:}\,\rho\} \vdash_S p^{n-1} q{:}\,\alpha$ or the typing $\{p{:}\,\mu, q{:}\,\rho\} \vdash_S p^{n-1} q{:}\,\gamma$ must be derived. The former can be derived only if $n-1$ is even and the latter only if $n-1$ is odd, since

$$T^k_\tau = \begin{cases} \{\alpha, \beta\} & \text{if } k \text{ is even,} \\ \{\beta, \gamma\} & \text{if } k \text{ is odd.} \end{cases}$$

Remark 6.3. Let $\tau \equiv \mu \to \rho \to \tau'$ be a Church numeral type in $\lambda^S_\cap$, with $\mu \equiv \mu_1 \cap \ldots \cap \mu_\ell, \rho \equiv \rho_1 \cap \ldots \cap \rho_m$ and $\mu_i \equiv \mu^i_1 \cap \ldots \cap \mu^i_{k_i} \to \mu^i_0$ $(1 \le i \le \ell)$. We will assume w.l.o.g. that

$$\{\mu^i_1, \dots, \mu^i_{k_i}, \mu^i_0\} \subseteq T_\tau, \tag{10}$$

for each $i \in \{1, \dots, \ell\}$. Indeed, if (10) does not hold for some $1 \le i \le \ell$, then the judgment $B \vdash_S p{:}\,\mu_i$ is never used in any derivation of $\vdash_S \mathsf{c}_n{:}\,\tau$.

Given a Church numeral type τ as in Figure 3, we characterize the translations in $\lambda_\to$ of any possible derivation $\mathcal{D}$ of the judgment $\{p{:}\,\mu, q{:}\,\rho\} \vdash_S p^n q{:}\,\sigma$, for all n and for all $\sigma \in T_\tau$. In the next definition, q^j and p^j are term variables generated by the translation $|\cdot|^{\mathrm{D}}$ (Definition 3.2). In particular, q^j is assigned type $|\rho_j|^{\mathrm{T}}$, while p^i is assigned type $|\mu_i|^{\mathrm{T}}$.

Definition 6.4 (PSEUDONUMERALS). Let τ be a Church numeral type. For any $\sigma \in T_\tau$, we define a set of terms $\mathbf{B}^{\tau,\sigma} = \bigcup_{n \in \mathbb{N}} \mathbf{B}^{\tau,\sigma}_n$ as follows:

$$\begin{aligned} \mathbf{B}^{\tau,\sigma}_0 &= \{q^j \mid \rho_j = \sigma\}, \\ \mathbf{B}^{\tau,\sigma}_{k+1} &= \{p^j Q_1 \ldots Q_{k_j} \mid 1 \le j \le l,\ \sigma = \mu^j_0 \text{ and } Q_r \in \mathbf{B}^{\tau,\mu^j_r}_k (1 \le r \le k_j)\}. \end{aligned}$$

Moreover, we define the set $\mathbf{N}^\tau = \bigcup_{n \in \mathbb{N}} \mathbf{N}^\tau_n$ of τ-*pseudonumerals* as follows:

$$\mathbf{N}^\tau_k = \{\lambda p^1 \ldots p^\ell q^1 \ldots q^m.b \mid b \in \mathbf{B}^{\tau,\tau'}_k\}.$$

The structure of (Böhm trees of) pseudonumerals is shown in Figure 4.

Example 6.5. Let $\tau \equiv \mu \to \rho \to \beta$ be as in Example 6.2, i.e.

$$\mu \equiv (\alpha \cap \beta \to \beta) \cap (\gamma \cap \beta \to \beta) \cap (\alpha \to \gamma) \cap (\gamma \to \alpha) \text{ and } \rho \equiv \alpha \cap \beta.$$

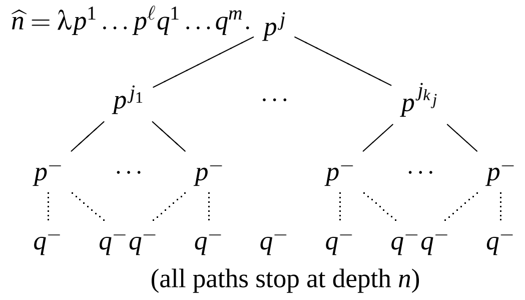

$\widehat{n} = \lambda p^1 \dots p^\ell q^1 \dots q^m.\ p^j$
p^{j_1}
$\cdots$
$p^{j_{k_j}}$
p^- $\cdots$ p^- p^- $\cdots$ p^-
q^- q^-q^- q^- q^- q^- q^-q^- q^-
(all paths stop at depth n)

Fig. 4. Structure of pseudonumerals

We have $T_\tau = \{\alpha, \beta, \gamma\}$ and

$$\begin{array}{lll}
\mathbf{B}_0^{\tau,\beta} = \{q^2\} & \mathbf{B}_0^{\tau,\alpha} = \{q^1\} & \mathbf{B}_0^{\tau,\gamma} = \{\,\} \\[2ex]
\mathbf{B}_1^{\tau,\beta} = \{p^1 q^1 q^2\} & \mathbf{B}_1^{\tau,\alpha} = \{\,\} & \mathbf{B}_1^{\tau,\gamma} = \{p^3 q^1\} \\[2ex]
\mathbf{B}_2^{\tau,\beta} = \{p^2(p^1q^1q^2)(p^3q^1)\} & \mathbf{B}_2^{\tau,\alpha} = \{p^4(p^3q^1)\} & \mathbf{B}_2^{\tau,\gamma} = \{\,\} \\[2ex]
\mathbf{B}_3^{\tau,\beta} = \{p^1(p^2(p^1q^1q^2)(p^3q^1))(p^4(p^3q^1))\} & & \\
 & \mathbf{B}_3^{\tau,\alpha} = \{\,\} & \mathbf{B}_3^{\tau,\gamma} = \{p^3(p^4(p^3q^1))\} \\[2ex]
\dots & \dots & \dots
\end{array}$$

We observe that any τ-pseudonumeral $\mathsf{p}_n \in \mathbf{N}_n^\tau$ shares with c_n the depth n of its Böhm tree. Hence, the whole set of terms $\mathbf{N}_n^\tau$ can be considered as a redundant representation of the natural number n.

Each τ-pseudonumeral $\mathsf{p}_n \in \mathbf{N}_n^\tau$ carries the same information provided by a typing derivation of the judgment $\vdash_S \mathsf{c}_n : \tau$.

Proposition 6.6. Let $A = \{p : \mu, q : \rho\} \in Bases_S$. For any $n \in \mathbb{N}$ and $\sigma \in T_\tau$,

$$\{ |\mathcal{D}|^{\mathrm{D}} \mid \mathcal{D} \text{ is a typing derivation for } A \vdash_S p^n q : \sigma \} = \mathbf{B}_n^{\tau,\sigma}.$$

Proof. Induction on n.
($n = 0$). If $q^j \in \mathbf{B}_0^{\tau,\sigma}$, then there exists ρ_j such that $\rho_j = \sigma$. In such a case, we have $A \vdash_s q : \rho_j$ and

$$\left| \frac{A(q) = \rho_1 \cap \dots \cap \rho_m}{A \vdash_s q : \rho_j} \right|^{\mathrm{D}} = q^j.$$

Conversely, if the typing $A \vdash_S q : \sigma$ is derivable, then for some j, $\sigma = \rho_j$ and $q^j \in \mathbf{B}_0^{\tau,\sigma}$.
($n > 0$). By definition of $\mathbf{B}_n^{\tau,\sigma}$, we have that a pseudonumeral body b belongs to $\mathbf{B}_n^{\tau,\sigma}$ if and only if $\mathsf{b} \equiv p^i Q_1 \dots Q_r$ and for all r $(1 \le r \le k_i)$, $Q_r \in \mathbf{B}_{n-1}^{\tau,\sigma_r}$, $\sigma \equiv \mu_0^i$ and $\sigma_r = \mu_r^i$. By the inductive

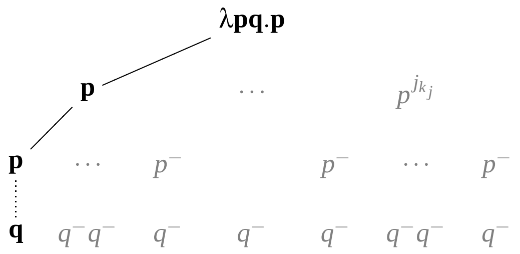


Fig. 5. Extracting a Church numeral

hypothesis, we have that $Q_r \in \mathbf{B}^{\tau,\sigma_r}_{n-1}$ if and only if $Q_r = |\,\mathcal{D}_r[A \vdash_S p^{n-1}q{:}\,\mu^i_r]\,|^{\mathrm{D}}$ for some derivation $\mathcal{D}_r$ and hence if and only if there exists a derivation of the following shape:

$$\dfrac{\dfrac{A(p) = \mu_1 \cap \cdots \cap \mu_\ell}{A \vdash_s p{:}\,\mu^i_1 \cap \ldots \cap \mu^i_{k_i} \to \mu^i_0} \quad \dfrac{\mathcal{D}_1}{A \vdash_s p^{n-1}q{:}\,\mu^i_1} \quad \cdots \quad \dfrac{\mathcal{D}_{k_i}}{A \vdash_s p^{n-1}q{:}\,\mu^i_{k_i}}}{A \vdash_s p^n q{:}\,\mu^i_0 \equiv \sigma}$$

By definition of $|\cdot|^{\mathrm{D}}$, $|\,\mathcal{D}\,|^{\mathrm{D}} = p^i Q_1 \ldots Q_r \equiv \mathsf{b}$. □

6.1. *Construction of the decoder $D_{[\tau]}$*

Given a Church numeral type τ, we are now ready to address the problem of constructing a λ-term $D_{[\tau]}$, which we have already called *decoder*, such that

$$D_{[\tau]}\ |\,\mathcal{D}[\vdash_S \mathsf{c}_n{:}\,\tau]\,|^{\mathrm{D}} \xrightarrow[\beta]{*} \mathsf{c}_n,$$

for every derivation $\mathcal{D}$. The idea is to prune the tree of the τ-pseudonumeral p_n, (Figure 4), keeping its leftmost branch and collapsing the non–leaf variables of this branch into a single one, hence reconstructing a Church numeral, as shown in Figure 5.

Lemma 6.7. Let $\tau \equiv \mu \to \rho \to \tau'$ be a Church numeral type in $\lambda^S_\cap$, with $\mu \equiv \mu_1 \cap \ldots \cap \mu_\ell, \rho \equiv \rho_1 \cap \ldots \cap \rho_m$ and $\mu_i \equiv \mu^i_1 \cap \ldots \cap \mu^i_{k_i} \to \mu^i_0$ $(1 \le i \le \ell)$. There exist a term $D_{[\tau]}$ and a basis $B_{[\tau]}$ such that

(i) $B_{[\tau]} \vdash_{\to} D_{[\tau]}{:}\ |\,\tau\,|^{\mathrm{T}} \to (o \to o) \to o \to o$;
(ii) $D_{[\tau]}\mathsf{p}_n \xrightarrow[\beta]{*} \mathsf{c}_n$, for any $n \in \mathbb{N}$ and $\mathsf{p}_n \in \mathbf{N}^\tau_n$.

Proof. We first define a Curry typeable term D such that, for every n and for every τ-pseudonumeral $\mathsf{p}_n \in \mathbf{N}^\tau_n$,

$$D\mathsf{p}_n \xrightarrow[\beta]{*} \mathsf{c}_n.$$

We observe that, for any n and for any $\mathsf{p}_n \in \mathbf{N}^\tau_n$, every free occurrence of the variable p^i in the body b_n of p_n is followed by exactly k_i arguments, since $\mu_i = \mu^i_1 \cap \ldots \cap \mu^i_{k_i} \to \mu^i_0$. Thus, considering the types

$$\phi_i \equiv \underbrace{o \to \ldots \to o}_{k_i \text{ times}} \to o \quad (1 \leq i \leq \ell),$$

we are able to derive the typings

$$\{p^i : \phi_i, q^j : o\}_{1 \leq i \leq \ell, 1 \leq j \leq m} \vdash_{\to} \mathsf{b}_n : o,$$

for all $\sigma \in T_\tau, n \in \mathbb{N}$ and $\mathsf{b}_n \in \mathbf{B}_n^{\tau,\sigma}$.
We now consider, for $1 \leq i \leq \ell$ and $1 \leq j \leq m$, the terms

$$P_i = \lambda x_1 \ldots x_{k_i}.px_1 \text{ and } Q_j = q.$$

By assigning the type o to every $x \in \{x_1 \ldots x_{k_i}\}$, we have

$$\{p : o \to o, q : o\} \vdash_{\to} P_i : \phi_i \text{ and } \{p : o \to o, q : o\} \vdash_{\to} Q_j : o.$$

In addition, a straightforward induction shows that

$$\forall n \in \mathbb{N}, \mathsf{p}_n \in \mathbf{N}_n^\tau.\ \mathsf{p}_n P_1 \ldots P_\ell Q_1 \ldots Q_m \xrightarrow[\beta]{*} p^n q.$$

It follows that the term

$$D \equiv \lambda xpq.xP_1 \ldots P_\ell Q_1 \ldots Q_m$$

has the required behavior. Moreover,

$$\vdash_{\to} D : \psi \to (o \to o) \to o \to o,$$

where

$$\psi \equiv \phi_1 \to \cdots \to \phi_\ell \to \underbrace{o \to \ldots \to o}_{m \text{ times}} \to o.$$

Hence, D accepts as argument a pseudonumeral having type ψ and transforms it into the corresponding Church numeral. In order to complete the proof, we slightly modify the previous construction and we build a term $D_{[\tau]}$, which depends on τ, accepting as argument a pseudonumeral with type $|\,\tau\,|^{\mathrm{T}}$. We first observe that, w.l.o.g., a term can be typed using only one type variable, say o. Under such assumption, we have $(1 \leq i \leq \ell)$

$$|\,\mu_0^i\,|^{\mathrm{T}} = \xi_1 \to \cdots \to \xi_{a_i} \to o$$

and

$$|\,\mu_1^i\,|^{\mathrm{T}} = \nu_1 \to \cdots \to \nu_{b_i} \to o$$

for some $a_i, b_i \in \mathbb{N}$ and some simple types $\xi_1, \ldots, \xi_{a_i}, \nu_1, \ldots, \nu_{b_i}$. Moreover, for $1 \leq j \leq m$,

$$|\,\rho_j\,|^{\mathrm{T}} \equiv \varphi_1 \to \cdots \to \varphi_{r_j} \to o,$$

for some $r_j \in \mathbb{N}$ and some simple types $\varphi_1, \ldots, \varphi_{r_j}$, and

$$|\,\tau'\,|^{\mathrm{T}} \equiv \vartheta_1 \to \cdots \to \vartheta_t \to o,$$

for some $t \in \mathbb{N}$ and some simple types $\vartheta_1, \ldots, \vartheta_t$. For $1 \leq i \leq \ell$ and $1 \leq j \leq m$, we consider the terms

$$P_i = \lambda x_1 \ldots x_{k_i} s_1 \ldots s_{a_i}.p(x_1 v^i_1 \ldots v^i_{b_i})$$

and

$$Q_j = \lambda s_1 \ldots s_{r_j}.q,$$

where the v^-_-'s are free variables with suitable types, allowing a uniform typing for p and q with $o \to o$ and o, respectively. Therefore, every P_i can be assigned the type $|\,\mu_i\,|^{\mathrm{T}}$, while every Q_j can be assigned the type $|\,\rho_j\,|^{\mathrm{T}}$. Finally, we define

$$D_{[\tau]} = \lambda xpq.xP_1 \ldots P_l Q_1 \ldots Q_m v^0_1 \ldots v^0_t.$$

$\square$

6.2. *Construction of the encoder* $E_{[\tau_1,\ldots,\tau_k,\tau]}$

The construction of terms which transform a Church numeral into a pseudonumeral (as obtained from the translation function), is as follows.

Lemma 6.8. Let τ be a full numeral type. Then there exists a Curry typeable term $E_{[\tau]}$, such that, for any n, $E_{[\tau]}\mathsf{c}_n \overset{*}{\to}_\beta \mathsf{p}_n$, for some $\mathsf{p}_n \in \mathbf{N}^\tau_n$.

Proof. In general, the body of the τ-pseudonumeral p_n contains pseudonumeral bodies $\mathsf{b}_k \in \mathbf{B}^{\tau,\sigma}_k$ as sub-terms, for $k < n$ and $\sigma \in T_\tau$ (see Proposition 6.6). Let τ be a type as in Fig. 3 and let $t = |\,T_\tau\,|$. For each $\sigma_i \in T_\tau$, we indicate by δ_i the strict type $\mu \to \rho \to \sigma_i$. The idea is to construct a term that uses a numeral c_n to generate iteratively a t-tuple of δ_i pseudonumerals.

At the k-th step of the iteration, the mentioned t-tuple contains an element (if it exists) for each set $\mathbf{N}^{\delta_i}_k$. Hence it has the shape

$$<\mathsf{p}^1_k, \ldots, \mathsf{p}^t_k>, \text{ for } \mathsf{p}^i_k \in \mathbf{N}^{\delta_i}_k.$$

We first show the encoders construction under the hypothesis that for all $1 \le i \le t$, δ_i is a full numeral type (we recall that this is not always true, as previously shown in Example 6.2). Under this hypothesis, $T_\tau = \{\rho_1, \ldots, \rho_m\}$ and hence $t = m$, since types in T_τ have to be assignable to the body of c_0.

Moreover we can choose, for each $\sigma_j \in T_\tau$, a μ_i such that $\mu_i \equiv \mu^i_1 \cap \ldots \cap \mu^i_{k_i} \to \mu^i_0$, $\mu^i_0 = \sigma_j$, and all μ^i_r are in T_τ, for $1 \le r \le k_i$. Such a type exists under the above hypothesis since, for all n, $T^n_\tau = T_\tau$. We can rearrange indexes of types in T_τ, μ and ρ in such a way that $\sigma_i = \mu^i_0$.

We define, for each $1 \le i \le t$, an index function

$$g_i \colon \{1, \ldots, k_i\} \to \{1, \ldots, t\}$$

such that $\sigma_{g_i(j)} = \mu^i_j$. We also use the convention that q^i has type $|\,\rho_i\,|^{\mathrm{T}}$ and p^i has type

$$|\,\mu^i_1\,|^{\mathrm{T}} \to \cdots \to |\,\mu^i_{k_i}\,|^{\mathrm{T}} \to |\,\mu^i_0\,|^{\mathrm{T}}.$$

We now describe the behavior of two terms Q and P (fully defined later), that will serve as arguments of c_n in the iteration outlined above.

1 Q is simply the t–tuple $<\mathsf{p}^1_0, \ldots, \mathsf{p}^t_0>$, with $\mathsf{p}^i_0 \in \mathbf{N}^{\delta_i}_0$. Observe that such a term can be typed in $\lambda_\to$ with $\phi \equiv (|\,\delta_1\,|^{\mathrm{T}} \to \cdots \to |\,\delta_t\,|^{\mathrm{T}} \to \gamma) \to \gamma$, for an arbitrary type γ;

2 P is a term that, when applied to a t–tuple of the form $<\mathsf{p}_k^1,\ldots,\mathsf{p}_k^t>$, reduces to the t-tuple $<\mathsf{p}_{k+1}^1,\ldots,\mathsf{p}_{k+1}^t>$ with $\mathsf{p}_k^i \in \mathbf{N}_k^{\delta_i}$. We will show that P can be typed in $\lambda_\to$ with $\phi \to \phi$.

Let us consider the following "pseudosuccessors" ($1 \leq i \leq t$):

$$S_i = \lambda x_1 \ldots x_t p^1 \ldots p^l q^1 \ldots q^m . p^i (x_{g_i(1)} p^1 \ldots p^l q^1 \ldots q^m) \ldots (x_{g_i(k_i)} p^1 \ldots p^l q^1 \ldots q^m).$$

S_i is typeable with

$$|\,\delta_1\,|^{\mathrm{T}} \to \cdots \to |\,\delta_t\,|^{\mathrm{T}} \to |\,\delta_i\,|^{\mathrm{T}}$$

and it reduces to the δ_i pseudonumeral p_{k+1} once applied to the $\mathsf{p}_k^1,\ldots,\mathsf{p}_k^t \in \mathbf{N}_k^{\delta_r}$ $(r = 1,\ldots,t)$; hence each S_i builds an element of $\mathbf{N}_{k+1}^{\delta_i}$, taking as arguments t elements, one for each $\mathbf{N}_k^{\delta_r}$, $1 \leq r \leq t$. We can now define the term P as follows:

$$P \equiv \lambda z w . z(\lambda x_1 \ldots x_t . w(S_1 x_1 \ldots x_t) \ldots (S_t x_1 \ldots x_t)).$$

Typing, for an arbitrary type γ:

— x_i with $|\,\delta_i\,|^{\mathrm{T}}$,

— w with $|\,\delta_1\,|^{\mathrm{T}} \to \cdots \to |\,\delta_t\,|^{\mathrm{T}} \to \gamma$,

— z with $(|\,\delta_1\,|^{\mathrm{T}} \to \cdots \to |\,\delta_t\,|^{\mathrm{T}} \to \gamma) \to \gamma$,

the term P has type $\phi \to \phi$.

Finally, we can extract, from the constructed t–tuple, the τ pseudonumeral p_n, since we know its position in the t–tuple, say r, by applying the term:

$$N = \lambda x_1 \ldots x_t . x_r,$$

which is typeable in $\lambda_\to$ with $|\,\delta_1\,|^{\mathrm{T}} \to \cdots \to |\,\delta_t\,|^{\mathrm{T}} \to |\,\tau\,|^{\mathrm{T}}$. Observe that, by the hypothesis that τ is a full numeral type, the τ pseudonumeral p_k appears in the t–tuple at each stage of the iteration. Choosing $\gamma = |\,\tau'\,|^{\mathrm{T}}$, we can type in $\lambda_\to$ the encoder

$$E_{[\tau]} = \lambda x . x P Q N$$

with the type $\mathbf{N}(\phi) \to |\,\tau\,|^{\mathrm{T}}$.

This construction can be adapted to the general case in which we remove the hypothesis that for each $\sigma \in T_\tau$ and for each n, the judgment $\vdash_S \mathsf{c}_n : \mu \to \rho \to \sigma$ is derivable. In this case we must take care of the fact that some pseudonumerals could not be constructed at some stage of the iteration, and hence some successors could not be applicable at later stages. This difficulty can be overcome by introducing t boolean values $B_1,\ldots,B_t$ representing the existence of pseudonumerals: at the n^{th} stage of the iteration, B_i is True if and only if a δ_i pseudonumeral p_n can be constructed. Moreover the arbitrary choice of a pseudosuccessor for each δ_i is no longer justified[‡]. We must check the applicability of each successor, as it is induced by a μ_i. When successors are looked-up in order to perform an iteration step, the existence of their arguments is checked and the first applicable successor is picked up. It is worth stressing that in this case we have to consider, for

[‡] In Example 6.2, we have two ways to construct a $\mu \to \rho \to \beta$ pseudonumeral p_{n+1}, one applicable only for n odd, and the other only for n even.

a given σ_i in T_τ, all the successors constructing a pseudonumeral in $\mathbf{N}^{\delta_i}_{k+1}$, whereas in the simple case the arbitrary choice of one of these was sufficient.

More formally, consider for each $\sigma_i \in T_\tau$, the index set $X_i = \{j \ : \ \mu_0^j = \sigma_i\}$, and for each $\mu_r, 1 \leq r \leq l$, the index set $Y_r = \{j \ : \ \mu_n^r = \sigma_j, 1 \leq n \leq k_r\}$.

Let $X_i = \{r_1, \ldots, r_n\}$. Using syntactic sugar to avoid usual lambda-calculus encoding of boolean values and operators, the "pseudosuccessor" that builds a pseudonumeral of type $\mid \delta_i \mid^{\mathrm{T}}$ has the shape:

```
$S_i \equiv \lambda x_1 \ldots x_t b_1 \ldots b_t p^i \ldots p^l q^1 \ldots q^m.$
        **if** $\bigwedge_{j \in Y_{r_1}} b_j$
          **then** $p^{r_1}(x_{g_{r_1}(1)} p^1 \ldots p^l q^1 \ldots q^m) \ldots (x_{g_{r_1}(k_{r_1})} p^{r_1} \ldots p^l q^1 \ldots q^m)$
          **else if** $\bigwedge_{j \in Y_{r_2}} b_j$
               **then** $p^{r_2}(x_{g_{r_2}(1)} p^1 \ldots p^l q^1 \ldots q^m) \ldots (x_{g_{r_2}(k_{r_2})} p^1 \ldots p^l q^1 \ldots q^m)$
               $\vdots$
          **else if** $\bigwedge_{j \in Y_{r_n}} b_j$
               **then** $p^{r_n}(x_{g_{r_n}(1)} p^1 \ldots p^l q^1 \ldots q^m) \ldots (x_{g_{r_n}(k_{r_n})} p^1 \ldots p^l q^1 \ldots q^m)$
               **else** $Z_i$
```

where Z_i is an arbitrary term of type $\mid \delta_i \mid^{\mathrm{T}}$.

Moreover, the invariant that, for all n, the i^{th} boolean value is True if and only if there exists a δ_i pseudonumeral p_n, is preserved using t terms T_i, which recalculate boolean values. Each T_i is defined as follows:

$$T_i = \lambda b_1 \ldots b_t. \bigvee_{r \in X_i} \bigwedge_{j \in Y_r} b_j.$$

We can now redefine terms Q and P that will serve as argument of c_n in the iteration:

1 Q is the $2t$–tuple $<Q_1, \ldots, Q_t, B_1 \ldots, B_t>$, where:

$$Q_i = \begin{cases} \mathsf{p}_0 & \text{if } \sigma_i = \rho_j, \text{for some } j \\ Z_i & \text{for an arbitrary term } Z_i \text{ of type } \delta_i \text{ otherwise} \end{cases}$$

$$B_i = \begin{cases} \mathsf{True} & \text{if } \sigma_i = \rho_j, \text{for some } j \\ \mathsf{False} & \text{otherwise} \end{cases}$$

2 P is the term:

$$P \equiv \lambda zw.z(\lambda x_1 \ldots x_t b_1 \ldots b_t.w(S_1 x_1 \ldots x_t b_1 \ldots b_t) \ldots (S_t x_1 \ldots x_t b_1 \ldots b_t) \\ (T_1 b_1 \ldots b_t) \ldots (T_t b_1 \ldots b_t)).$$

□

By properties of the translation function, the Curry typeable term $\widehat{M}$ expects as arguments $k \geq 1$ pseudonumerals $\mathsf{p}_n^1, \ldots, \mathsf{p}_n^k$ ($\mathsf{p}_n^r \in \mathbf{N}^n_{\tau_r}$). However, the term

$$\lambda x.\widehat{M}(E_{[\tau_1]}x) \ldots (E_{[\tau_k]}x)$$

is not, in general, Curry typeable, because different encoders require arguments of different types, and hence they cannot be applied to the same variable in $\lambda_{\rightarrow}$. Thus, we use an encoder which constructs in parallel pseudonumerals $\mathsf{p}_n^1, \ldots \mathsf{p}_n^k$ and puts them finally into a k–tuple.

Lemma 6.9. Let M be a term typeable in $\lambda_{\cap}^S$, which uniformly represents a numeric unary function φ. Let $\widehat{M} = \mid (\mathcal{D}[\vdash_S M\colon \tau_1 \cap \ldots \cap \tau_k \rightarrow \tau]) \mid^{\mathrm{D}}$. Then there exists a term $E_{[\tau_1,\ldots,\tau_k,\tau]}$, such that $E_{[\tau_1,\ldots,\tau_k,\tau]}\mathsf{c}_n\widehat{M}$ is a Curry typeable term which reduces to $\widehat{M}\mathsf{p}_n^1 \ldots \mathsf{p}_n^k$, $\mathsf{p}_n^r \in \mathbf{N}_n^{\tau_r}$, for $1 \leq r \leq k$.

Proof. The construction of Lemma 6.8 works the same for the set of types $T = \biguplus_{i=1}^k T_{\tau_i}$, instead of a single T_τ.

We extract the k pseudonumerals of types $\mid \tau_1 \mid^{\mathrm{T}}, \ldots, \mid \tau_k \mid^{\mathrm{T}}$ from the t–tuple generated by the term $\mathsf{c}_n PQ$, knowing their positions, say $r_1, \ldots, r_k$, in the t–tuple, using the term:

$$N = \lambda x_1 \ldots x_t z.z x_{r_1} \ldots x_{r_k}.$$

We can type the term N with

$$(\mid \delta_1 \mid^{\mathrm{T}} \rightarrow \cdots \rightarrow \mid \delta_t \mid^{\mathrm{T}} \rightarrow \gamma),$$

and hence, instantiating the arbitrary type γ in the type ϕ with $(\mid \tau_1 \mid^{\mathrm{T}} \rightarrow \cdots \rightarrow \mid \tau_k \mid^{\mathrm{T}} \rightarrow \mid \tau \mid^{\mathrm{T}}) \rightarrow \mid \tau \mid^{\mathrm{T}}$ and typing the numeral c_n with $(\phi \rightarrow \phi) \rightarrow \phi \rightarrow \phi$, the term $(\mathsf{c}_n PQ)N$ has type γ. Since $\widehat{M}$ is typeable with $\mid \tau_1 \mid^{\mathrm{T}} \rightarrow \cdots \rightarrow \mid \tau_k \mid^{\mathrm{T}} \rightarrow \mid \tau \mid^{\mathrm{T}}$ (by definition of $\mid \cdot \mid^{\mathrm{T}}$ and Lemma 3.5), the term $(\mathsf{c}_n PQ)N\widehat{M}$ is typeable in $\lambda_{\rightarrow}$ with $\mid \tau \mid^{\mathrm{T}}$ and it reduces to $\widehat{M}\mathsf{p}_n^1 \ldots \mathsf{p}_n^k$, $\mathsf{p}_n^r \in \mathbf{N}_n^{\tau_r}$. Hence the term

$$E_{[\tau_1,\ldots,\tau_k,\tau]} = \lambda xy.xPQNy$$

satisfies the statement. □

Using the above lemmas we prove the main result of this section.

Theorem 6.10. Every function $\varphi\colon \mathbb{N} \rightarrow \mathbb{N}$, uniformly representable in $\lambda_{\cap}^S$, is uniformly representable in $\lambda_{\rightarrow}$.

Proof. If M represents a function in $\lambda_{\cap}^S$, let $\widehat{M}$ be the term obtained by the translation function from a type derivation of M. Using Lemmas 6.7 and 6.9, we have that the term

$$M' \equiv \lambda x.D_{[\tau]}(E_{[\tau_1,\ldots,\tau_k,\tau]}\widehat{M}x), \tag{11}$$

represents φ in $\lambda_{\rightarrow}$. □

7. Beyond Strict Types

In the previous section, we proved that the set of uniformly representable functions in the strict intersection type system is exactly the set of uniformly representable functions in the simply typed lambda calculus. It is worth analyzing whether the same result can be proven for $\lambda_{\rightarrow}$ with respect to other intersection type systems. In this section, we consider the Barendregt-Coppo-Dezani system (BCDC83) and the essential intersection type system (Hin82). Even if the sets of typeable terms are exactly the same as the one of $\lambda_{\cap}^S$, such systems allow more type judgments to be derived, and implications on uniform definability of functions are not straightforward. Nevertheless, we will prove that the set of uniformly representable functions in these systems coincides with the set of uniformly representable functions in the simply typed λ-calculus.

$$(\mathrm{Var})_{BCD} \qquad \frac{A(x)=\sigma}{A\vdash_{BCD} x{:}\sigma}$$

$$(\to\mathrm{I})_{BCD} \qquad \frac{A\cup\{x{:}\sigma\}\vdash_{BCD} M{:}\tau}{A\vdash_{BCD} \lambda x.M{:}\sigma\to\tau}$$

$$(\to\mathrm{E})_{BCD} \qquad \frac{A\vdash_{BCD} M{:}\sigma\to\tau \qquad A\vdash_{BCD} N{:}\sigma}{A\vdash_{BCD} MN{:}\tau}$$

$$(\cap\mathrm{I})_{BCD} \qquad \frac{A\vdash_{BCD} M{:}\sigma \qquad A\vdash_{BCD} M{:}\tau}{A\vdash_{BCD} M{:}\sigma\cap\tau}$$

$$(\cap\mathrm{E})_{BCD} \qquad \frac{A\vdash_{BCD} M{:}\sigma\cap\tau}{A\vdash_{BCD} M{:}\sigma} \quad \frac{A\vdash_{BCD} M{:}\sigma\cap\tau}{A\vdash_{BCD} M{:}\tau}$$

$$(\leq)_{BCD} \qquad \frac{A\vdash_{BCD} M{:}\sigma \qquad \sigma\leq\tau}{A\vdash_{BCD} M{:}\tau}$$

Fig. 6. The type assignment system of $\lambda_\cap^{BCD}$

7.1. *Lambda Calculus with Intersection Types*

We briefly recall the system of Barendregt-Coppo-Dezani ((BCDC83)).

Definition 7.1. *Intersection types* are generated using the following grammar:

$$\sigma ::= \alpha \mid (\sigma\to\sigma) \mid (\sigma\cap\sigma) \tag{12}$$

where α ranges over a countable set of type variables. We call *intersection types*, notation $Type_\cap^{BCD}$, the set of types resulting from (12). The intersection type constructor, $\cap$, takes precedence over the arrow type constructor and hence $\sigma_1\cap\sigma_2\to\tau$ is an abbreviation for $(\sigma_1\cap\sigma_2)\to\tau$.

Definition 7.2. The *type inclusion relation* $\leq$ $(\subseteq Type_\cap^{BCD}\times Type_\cap^{BCD})$ is inductively defined by:

1 $\sigma\leq\sigma$;
2 $\sigma\cap\tau\leq\sigma$ and $\sigma\cap\tau\leq\tau$;
3 $(\sigma\to\tau)\cap(\sigma\to\rho)\leq\sigma\to\tau\cap\rho$;
4 $\sigma\leq\tau \;\;\&\;\; \tau\leq\rho\Rightarrow\sigma\leq\rho$;
5 $\sigma\leq\tau \;\;\&\;\; \sigma\leq\rho\Rightarrow\sigma\leq\tau\cap\rho$;
6 $\sigma'\leq\sigma \;\;\&\;\; \tau\leq\tau'\Rightarrow\sigma\to\tau\leq\sigma'\to\tau'$.

We will write $\sigma\sim\tau$ if $\sigma\leq\tau \;\;\&\;\; \tau\leq\sigma$.

Definition 7.3. $\lambda_\cap^{BCD}$, the *Barendregt-Coppo-Dezani intersection type system* proves judgments of the kind $A\vdash_{BCD} M{:}\tau$. The basis A contains statements of the shape $x{:}\sigma$, with $\sigma\in Type_\cap^{BCD}$. The type τ belongs to the set of intersection types, $Type_\cap^{BCD}$. The system $\lambda_\cap^{BCD}$ consists of the rules in Fig. 6.

$$(\text{Var})_e \qquad \frac{A(x) = \sigma \quad \sigma \leq_e \tau}{A \vdash_e x{:}\tau}$$

$$(\to\text{I})_e \qquad \frac{A \cup \{x{:}\sigma\} \vdash_e M{:}\tau}{A \vdash_e \lambda x.M{:}\sigma \to \tau}$$

$$(\to\text{E})_e \qquad \frac{A \vdash_e M{:}(\tau_1 \cap \cdots \cap \tau_n) \to \tau \quad A \vdash_e N{:}\tau_1 \ldots A \vdash_e N{:}\tau_n}{A \vdash_e MN{:}\tau}$$

Fig. 7. The type assignment system of $\lambda^E_\cap$

7.2. *Essential Intersection Type System*

A restricted version of the $\lambda^{BCD}_\cap$ system, called *essential intersection type assignment system* in (vB93, Ch. 5), was introduced by Hindley (Hin82). The only difference with respect to the strict system is the $(\text{Var})_e$-rule, where the type inclusion relation $\leq_E$ is used.

Definition 7.4. The *type inclusion relation* $\leq_E$ ($\subseteq \mathit{Type}^S_\cap \times \mathit{Type}^S_\cap$) is inductively defined by:

1 $\sigma \leq_E \sigma$;
2 $\sigma \cap \tau \leq_E \sigma$ and $\sigma \cap \tau \leq_E \tau$;
3 $\sigma \leq_E \tau \;\;\&\;\; \tau \leq_E \rho \Rightarrow \sigma \leq_E \rho$;
4 $\sigma \leq_E \tau \;\;\&\;\; \sigma \leq_E \rho \Rightarrow \sigma \leq_E \tau \cap \rho$;
5 $\sigma' \leq_E \sigma \;\;\&\;\; \tau \leq_E \tau' \Rightarrow \sigma \to \tau \leq_E \sigma' \to \tau'$.

We will write $\sigma \sim_E \tau$ if $\sigma \leq_E \tau \;\;\&\;\; \tau \leq_E \sigma$.

Definition 7.5.

1 The *essential intersection type system* ($\lambda^E_\cap$) proves judgments of the kind $A \vdash_e M{:}\tau$, where $\tau \in \mathit{Type}^s_\cap$. The system $\lambda^E_\cap$ consists of the rules of Fig. 7. The basis A contains statements of the shape $x{:}\sigma$, with $\sigma \in \mathit{Type}^S_\cap$.
2 Judgments of the shape $A \vdash_E M{:}\sigma$ are derivable in $\lambda^E_\cap$ if and only if there exist types $\sigma_1, \ldots, \sigma_n$ such that $\sigma \equiv \sigma_1 \cap \ldots \cap \sigma_n$ and, for every $1 \leq i \leq n$, the judgment $A \vdash_e M{:}\sigma_i$ is derivable.

7.3. *Full Numeral Types in $\lambda^E_\cap$ and $\lambda^S_\cap$*

As stated by Hindley (Hin82), for any equivalence class of intersection types there is a strict intersection type which belongs to it. This is proven observing that intersections in the right-hand sides of arrows can be removed, using the equivalence on types $\sigma \to \tau \cap \rho \sim (\sigma \to \tau) \cap (\sigma \to \rho)$.

Proposition 7.6 (Hindley (Hin82)). There exists a map

$$(\cdot)^* : \mathit{Type}^{BCD}_\cap \to \mathit{Type}^S_\cap$$

such that, for every $\sigma \in \mathit{Type}^{BCD}_\cap$,

$$\sigma^* \in \mathit{Type}^S_\cap \text{ and } \sigma^* \sim \sigma.$$

Such translation of BCD types into strict intersection types allows for a strong relation between $\lambda_\cap^{BCD}$ and $\lambda_\cap^{E}$ to be established. The following (vB93, Prop. 5.3.2) holds (translation of bases is obtained just by applying the translation of types to all types that appear in bases).

Proposition 7.7.

1 For all $\sigma, \tau \in Type_\cap^S$, we have $\sigma \leq \tau \Leftrightarrow \sigma \leq_E \tau$;
2 $B \vdash_{BCD} M{:}\tau \Leftrightarrow B^* \vdash_E M{:}\tau^*$.

As a consequence we can easily prove that uniformly representable functions in $\lambda_\cap^{BCD}$ are uniformly representable in $\lambda_\cap^{E}$.

Theorem 7.8. Let φ be a numeric function. φ is uniformly representable in $\lambda_\cap^{BCD}$ if and only if it is uniformly representable in $\lambda_\cap^{E}$.

Proof. The only if part is obvious, since all judgments in $\lambda_\cap^{E}$ can be derived also in $\lambda_\cap^{BCD}$. Concerning the if part, let us consider a term M which uniformly represents φ in $\lambda_\cap^{BCD}$. By Proposition 7.7, if σ is a full numeral type in $\lambda_\cap^{BCD}$ then so is σ^* in $\lambda_\cap^{E}$, and $x{:}\sigma \vdash_{BCD} Mx{:}\tau$ implies $x{:}\sigma^* \vdash_E Mx{:}\tau^*$. Hence M uniformly represents φ in $\lambda_\cap^{E}$. $\square$

In (vB92) the system $\lambda_\cap^{S}$ has been proven powerful as the $\lambda_\cap^{BCD}$ system from the point of view of typeability. The relationship between the two systems (and between $\lambda_\cap^{E}$ and $\lambda_\cap^{S}$) is stated in the following.

Proposition 7.9.

1 If $B \vdash_{BCD} M{:}\sigma$, then there are B' and σ' such that $B' \vdash_S M{:}\sigma'$, $\sigma' \leq \sigma$, and $B \leq B'$;
2 If $B \vdash_E M{:}\sigma$, then there are B' and σ' such that $B' \vdash_S M{:}\sigma'$, $\sigma' \leq_E \sigma$, and $B \leq_E B'$.

These results do not allow us to conclude that uniformly representable functions in $\lambda_\cap^{BCD}$ are uniformly representable in $\lambda_\cap^{S}$. Indeed, the existentially quantified type σ' in Proposition 7.9 might not be a full numeral type in $\lambda_\cap^{S}$, even when σ is a full numeral type in $\lambda_\cap^{BCD}$ or $\lambda_\cap^{E}$. As an example, take

$$\tau = ((\sigma \cap \delta \to \rho) \to (\sigma \cap \delta \to \rho)) \to (\sigma \to \rho) \to (\sigma \cap \delta \to \rho).$$

It is easy to see that τ is full in the essential system, whereas it cannot be assigned to the numeral c_0 in the strict one.

However, numeral types in the essential system have the shape of numeral types in the strict system (see Fig. 3), namely: $\tau = \mu \to \rho \to \tau'$, where $\mu = \mu_1 \cap \ldots \cap \mu_l$, $\rho = \rho_1 \cap \ldots \cap \rho_m$ and $\mu_i = \mu_1^i \cap \ldots \cap \mu_{k_i}^i \to \mu_0^i$ for some $l, m, k_1, \ldots, k_l \in \mathbb{N}$.

We prove that full numeral types in the essential and strict systems are related as follows: if τ is full in the essential system, then there exists $\tau^\bullet$ such that $\tau^\bullet \leq_E \tau$ and $\tau^\bullet$ is full in the strict system.

Definition 7.10. A type $\sigma = \sigma_1 \cap \ldots \cap \sigma_k \in Type_\cap^S$ is a *full numeral type* in $\lambda_\cap^{E}$ (in $\lambda_\cap^{S}$, resp.) if all the σ_i are full numeral types in $\lambda_\cap^{E}$ (in $\lambda_\cap^{S}$, resp.).

Proposition 7.11. If $\sigma = \sigma_1 \cap \ldots \cap \sigma_k \in Type_\cap^S$ is a full numeral type in $\lambda_\cap^{E}$, then there exists a numeral type $\sigma^\bullet = \sigma_1^\bullet \cap \ldots \cap \sigma_k^\bullet \in Type_\cap^S$ which is full in $\lambda_\cap^{S}$, and moreover $\sigma^\bullet \leq_E \sigma$.

$$(\mathrm{Var})_{S'} \qquad \frac{A(x) = \tau_1 \cap \ldots \cap \tau_n \qquad 1 \leq i \leq n}{A \vdash_{S'} x{:}\tau_i}$$

$$(\to\mathrm{I})_{S'} \qquad \frac{A \cup \{x{:}\sigma\} \vdash_{S'} M{:}\tau}{A \vdash_{S'} \lambda x.M{:}\sigma \to \tau}$$

$$(\to\mathrm{E})_{S'} \qquad \frac{A \vdash_{S'} M{:}(\tau_1 \cap \cdots \cap \tau_n) \to \tau \qquad A \vdash_{S'} \overline{N}_1{:}\tau_1 \ldots A \vdash_{S'} \overline{N}_n{:}\tau_n \qquad \overline{N}_i \xrightarrow[\eta]{*} N}{A \vdash_{S'} MN{:}\tau}$$

Fig. 8. The type assignment system of $\lambda_\cap^{S'}$

Proof. The proof makes uses of some lemmas and auxiliary definitions. For the sake of readability, it is deferred to Section 8. □

7.4. *A Modified Strict Intersection Type System*

Let M be a uniform representant of $\varphi : \mathbb{N} \to \mathbb{N}$ in $\lambda_\cap^E$; then there exist a numeral type $\sigma = \sigma_1 \cap \ldots \cap \sigma_n$, full in $\lambda_\cap^E$, and a type τ such that

1 $\{x : \sigma\} \vdash_e Mx{:}\tau$;
2 $\forall n \in \mathbb{N}, M\mathsf{c}_n \xrightarrow[\beta]{*} \mathsf{c}_{\varphi(n)}$.

By Proposition 7.11, there exists a numeral type $\sigma^\bullet$, full in $\lambda_\cap^S$, such that

$$\{x : \sigma^\bullet\} \vdash_e Mx{:}\tau.$$

Therefore, φ can be uniformly represented in $\lambda_\cap^E$ using as input type some full numeral type in $\lambda_\cap^S$.

We observe that the translation function (Section 3) cannot be adapted naturally to transform derivations of $\lambda_\cap^E$. Indeed, the definition of the translation function is based on the fact that a variable-introduction rule in the strict system consists of a simple extraction of a strict type from an intersection of strict types. Therefore, a finite amount of types can be derived for a variable in $\lambda_\cap^S$, each of which is present in the type assumption for that variable. We call such a property the *type extraction (from bases)* property. The type extraction property does not hold in the essential system, where the variable-introduction rule makes a crucial use of the type inclusion relation $\leq_E$.

As pointed out in (vB93), typings are preserved by η-reduction in the essential system, but not in the strict one. We therefore introduce a variant ($\lambda_\cap^{S'}$) of the strict intersection type assignment system, such that

— $\lambda_\cap^{S'}$ has the extraction property;
— typings in $\lambda_\cap^{S'}$ are expressed *modulo* η-reduction.

We call $\lambda_\cap^{S'}$ the *strict extensional type system.*

Definition 7.12. The system $\lambda_\cap^{S'}$ proves judgments of the kind $A \vdash_{S'} M{:}\tau$, derived from the rules of Fig. 8. The basis A contains statements of the shape $x{:}\sigma$, with $\sigma \in \mathit{Type}_\cap^S$, and $\tau \in \mathit{Type}_\cap^s$.

We first note that the system $\lambda_\cap^{S'}$ enjoys the type extraction property. Moreover, given a typing in $\lambda_\cap^{E}$ having a term M as subject, there exists a typing in $\lambda_\cap^{S'}$ for a suitable η-expansion of M, which preserves both the predicate and the type assumptions.

Definition 7.13 (Type Extraction). We write $\sigma \preccurlyeq \tau$ iff $\sigma \equiv \sigma_1 \cap \cdots \cap \sigma_n$ and $\tau \equiv \sigma_i$, for some $1 \le i \le n$.

Lemma 7.14. If $A \vdash_e M\colon\tau$, then there exists $M' \in \Lambda$ such that

$$M' \xrightarrow[\eta]{*} M \quad \text{and} \quad A \vdash_{S'} M'\colon\tau.$$

Proof. By induction on the structure of the derivation of $A \vdash_e M\colon\tau$. The only interesting case is when the applied rule is of the form $(\mathrm{Var})_e$. In such case, we note that, for some suitable types, a $(\mathrm{Var})_e$-rule has the shape

$$(\mathrm{Var})_e \quad \frac{A(x) = \sigma \qquad \sigma \preccurlyeq \varphi \le_E \tau}{A \vdash_e x\colon\tau}$$

where, by Definition 7.4,

$$\begin{array}{rcl} \varphi & \equiv & \varphi_1 \to \cdots \to \varphi_k \to \alpha \\ \tau & \equiv & \tau_1 \to \cdots \to \tau_k \to \alpha \\ \tau_j & \le_E & \varphi_j \quad (j = 1, \dots, k). \end{array} \tag{13}$$

If we derive $A \vdash_e x\colon\tau$ by rule $(\mathrm{Var})_e$, we associate to the above judgment a derivation of the typing $A \vdash_{S'} X\colon\tau$, by induction on the structure of τ, and $X \xrightarrow[\eta]{*} x$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{B(x)=\sigma \quad \sigma \preccurlyeq \varphi}{B \vdash_{S'} x\colon\varphi} \qquad \dfrac{\overbrace{B(x_1)=\tau_1 \quad \tau_1 \le_E \varphi_1}^{(1)}}{B \vdash_E x_1\colon\varphi_1}}{B \vdash_{S'} x x_1\colon\varphi_2 \to \cdots \to \varphi_k \to \alpha}}{\begin{array}{c}\vdots\\ B \vdash_{S'} x x_1 \dots x_{k-1}\colon\varphi_k \to \alpha\end{array}} \qquad \dfrac{\overbrace{B(x_k)=\tau_k \quad \tau_k \le_E \varphi_k}^{(k)}}{B \vdash_E x_k\colon\varphi_k}}{B \vdash_{S'} x x_1 \dots x_k\colon\alpha}}{B - \{x_k\colon\tau_k\} \vdash_{S'} \lambda x_k . x x_1 \dots x_k\colon\tau_k \to \alpha}}{\begin{array}{c}\vdots\\ A \vdash_{S'} \lambda x_1 \dots x_k . x x_1 \dots x_k\colon\tau\end{array}}$$

By applying the induction hypothesis to sub-derivations indicated by $(1), \dots, (k)$, we obtain a derivation in $\lambda_\cap^{S'}$ of the typing

$$A \vdash_{S'} \lambda x_1 \dots x_k . x x_1 \dots x_k\colon\tau.$$

$\square$

Corollary 7.15. Let $\{x : \sigma\} \vdash_e Mx\colon\tau$, where M is a uniform representant of $\varphi : \mathbb{N} \to \mathbb{N}$, and σ is a full numeral type in $\lambda_\cap^{E}$; then there exists an η-expansion $(Mx)'$ of Mx and a type $\sigma^\bullet$ such that

$$\{x : \sigma^\bullet\} \vdash_{S'} (Mx)'\colon\tau$$

and $\sigma^{\bullet}$ is a full numeral type in $\lambda_{\cap}^{S}$ (and therefore in $\lambda_{\cap}^{S'}$).

Proof. By Proposition 7.11 and Lemma 7.14. □

As for the system $\lambda_{\cap}^{S}$ (Lemma 3.5), the translation of a typing derivation in $\lambda_{\cap}^{S'}$ is a simply typeable term.

Lemma 7.16. For any derivation $\mathcal{D}$ of the typing $A \vdash_{S'} M{:}\sigma$ in $\lambda_{\cap}^{S'}$, we have that

$$|A|^{\mathrm{B}} \vdash_{S'} |\mathcal{D}|^{\mathrm{D}} : |\sigma|^{\mathrm{T}}$$

is a typing in $\lambda_{\rightarrow}$.

Proof. Identical to the proof of Lemma 3.5. □

The translation still enjoys a kind of commutation property with respect to β-reduction (see Theorems 3.6 and 3.7).

Lemma 7.17.

1 Let $\mathcal{D}$ be a derivation of the typing $A \vdash_{S'} M{:}\tau$ in $\lambda_{\cap}^{S'}$, and let $M \xrightarrow[\beta]{} N$. Then there exists an η-expansion N' of N and a derivation $\mathcal{D}'$ of the typing $A \vdash_{S'} N'{:}\tau$ such that:

$$|\mathcal{D}|^{\mathrm{D}} \xrightarrow[\beta]{+} |\mathcal{D}'|^{\mathrm{D}}\ .$$

2 Let $\mathcal{D}$ be a derivation of the typing $A \vdash_{S'} M{:}\tau$ in $\lambda_{\cap}^{S'}$, and let $M \xrightarrow[\beta]{*} N$. Then there exists an η-expansion N' of N and a derivation $\mathcal{D}'$ of the typing $A \vdash_{S'} N'{:}\tau$ such that:

$$|\mathcal{D}|^{\mathrm{D}} \xrightarrow[\beta]{*} |\mathcal{D}'|^{\mathrm{D}}\ .$$

Proof.

1 The proof follows exactly the same pattern as the proof of Lemma 3.6. The only difference is that the derivations $\mathcal{D}_1,\dots,\mathcal{D}_n$ appear in the proof of Lemma 3.6 as different derivations having the same subject, while now they are derivations having as subjects different η-expansions of the same term.

2 From (Bar84, §3.3.8), $\xrightarrow[\beta]{*}$ commutes with $\xrightarrow[\eta]{*}$, i.e.

$$\begin{tikzcd} P \arrow[r, "*", "\beta"'] \arrow[d, "\eta"', "*"] & Q \arrow[d, dotted, "\eta"', "*"] \\ R \arrow[r, dotted, "*", "\beta"'] & S \end{tikzcd}$$

Therefore, using such a diagram, if N is obtained from M by means of a sequence of β-reductions and η-expansions, then

$$M \xrightarrow[\eta]{*} P \;\;\Rightarrow\;\; \exists Q \text{ such that } N \xrightarrow[\eta]{*} Q \text{ and } P \xrightarrow[\beta]{*} Q. \tag{14}$$

The thesis follows from point 1., using (14). Graphically:

$$\begin{tikzcd}
\widehat{M} \arrow[rr, "*", "\beta"'] & & \widehat{N_1} \arrow[rrr, "*", "\beta"'] & & & \widehat{N_k} \\
M \arrow[u, squiggly, "|\cdot|^D"] \arrow[r, "*", "\beta"'] \arrow[d, "\eta"', "*"] & Q_1 & N_1 \arrow[u, squiggly, "|\cdot|^D"] \arrow[l, "*"', "\eta"] \arrow[r, "*", "\beta"'] & \cdots \arrow[r, "*", "\beta"'] & Q_k & N_k \equiv N' \arrow[u, squiggly, "|\cdot|^D"] \arrow[l, "*"', "\eta"] \arrow[d, "\eta"', "*"] \\
P \arrow[rrrrr, "*", "\beta"'] & & & & & N
\end{tikzcd} \tag{15}$$

where $\widehat{M}, \widehat{N_1}, \widehat{N_k}$ are obtained translating suitable typing derivations having as subjects M, N_1, N_k, respectively.

□

7.5. *Lambda-definability in* $\lambda_\cap^{S'}$

Let M be a uniform representant of $\varphi : \mathbb{N} \rightarrow \mathbb{N}$ in $\lambda_\cap^E$. By Corollary 7.15, there exists a numeral type $\sigma^\bullet$, full in $\lambda_\cap^S$, and an η-expansion $(Mx)'$ of Mx such that, for some type τ, $\{x : \sigma^\bullet\} \vdash_{S'} (Mx)' {:}\, \tau$. For all $n \in \mathbb{N}$, $(\lambda x.Mx)\mathsf{c}_n \xrightarrow[\beta]{*} \mathsf{c}_{\varphi(n)}$. Hence, by η-postponement ((Bar84, §15.1.6)),

$$(\lambda x.(Mx)')\mathsf{c}_n \xrightarrow[\beta]{*} \mathsf{e}_{\varphi(n)} \xrightarrow[\eta]{*} \mathsf{c}_{\varphi(n)}.$$

It follows that, by Lemma 7.17, the translation of the typing derivation of $(\lambda x.(Mx)')\mathsf{c}_n$ β-reduces to the translation of an η-expansion $(\mathsf{e}_{\varphi(n)})$ of a Church numeral. It is therefore necessary to study the structure of terms obtained by translating type derivations (in $\lambda_\cap^{S'}$) of η-expansions of Church numerals.

We use the following notation: if F is a normal form and X is a λ-free normal form (a variable, in particular), we denote by

$$F(\!(X)\!)$$

the normal form of FX.

Definition 7.18. A *hereditary finite combinator* is a λ-term

$$\mathsf{F} \equiv \lambda a x_1 \ldots x_n.a(\mathsf{F}_1(\!(x_{g(1)})\!)) \ldots (\mathsf{F}_m(\!(x_{g(m)})\!))$$

where $m, n \geq 0$, $g : \{1, \ldots, m\} \rightarrow \{1, \ldots, n\}$ and, for $i = 1, \ldots, m$, F_i is a hereditary finite combinator. We denote by $\mathfrak{F}$ the set of hereditary finite combinators.

The relevance of hereditary finite combinators in our setting is exemplified in Table 1, where three different typing derivations are shown for the η-expansion of a variable x, together with their translations. For the sake of simplicity, we have omitted bases from derivations in Table 1; type assumptions for variables are drawn as superscripts in the subjects of variable-introduction rules.

The next lemma will enable us to determine the shape of terms obtained from the translation of $\lambda_\cap^{S'}$-typing derivations of η-expansions of Church numerals.

$\mathcal{D}$	$\mid\mathcal{D}\mid^{\mathrm{D}}$
$\dfrac{\dfrac{x^{(\alpha\cap\beta)\to\gamma}\!:(\alpha\cap\beta)\to\gamma \quad y^{\beta\cap\alpha}\!:\alpha \quad y^{\beta\cap\alpha}\!:\beta}{xy\!:\gamma}}{\lambda y.xy\!:(\beta\cap\alpha)\to\gamma}$	$\lambda y^1y^2.xy^2y^1$ $\Downarrow$ $\mid\mathcal{D}\mid^{\mathrm{D}}=\mathsf{F}_1(\!(x)\!)$, where $\mathsf{F}_1\equiv\lambda ax_1x_2.ax_2x_1$
$\dfrac{\dfrac{x^{\alpha\to\beta}\!:\alpha\to\beta \quad y^{\alpha\cap\gamma}\!:\alpha}{xy\!:\beta}}{\lambda y.xy\!:(\alpha\cap\gamma)\to\beta}$	$\lambda y^1y^2.xy^1$ $\Downarrow$ $\mid\mathcal{D}\mid^{\mathrm{D}}=\mathsf{F}_2(\!(x)\!)$, where $\mathsf{F}_2\equiv\lambda ax_1x_2.ax_1$
Let $\tau\equiv(\alpha\cap\beta\to\gamma)\cap(\alpha\cap\delta\to\gamma)\to\zeta$ $\dfrac{\dfrac{x^{\tau}\!:\tau \quad \dfrac{\dfrac{y^{\alpha\to\gamma}\!:\alpha\to\gamma \quad z^{\alpha\cap\beta}\!:\alpha}{yz\!:\gamma}}{\lambda z.yz\!:\alpha\cap\beta\to\gamma} \quad \dfrac{\dfrac{y^{\alpha\to\gamma}\!:\alpha\to\gamma \quad z^{\alpha\cap\delta}\!:\alpha}{yz\!:\gamma}}{\lambda z.yz\!:\alpha\cap\delta\to\gamma}}{xy\!:\zeta}}{\lambda y.xy\!:(\alpha\to\gamma)\to\zeta}$	$\lambda y.x(\lambda z^1z^2.yz^1)(\lambda z^1z^2.yz^1)$ $\Downarrow$ $\mid\mathcal{D}\mid^{\mathrm{D}}=\mathsf{F}_3(\!(x)\!)$, where $\mathsf{F}_3\equiv\lambda ax_1.a(\mathsf{F}_3^1(\!(x_1)\!))(\mathsf{F}_3^2(\!(x_1)\!))$ $\mathsf{F}_3^1=\mathsf{F}_3^2=\lambda av_1v_2.av_1$

Table 1. *Hereditary finite combinators and translation of derivations*

Lemma 7.19. Let $\mathcal{D}$ be a derivation of the typing

$$A\cup\{x\!:\sigma_1\cap\cdots\cap\sigma_n\}\vdash_{S'}X\!:\sigma,$$

where $X \xrightarrow[\eta]{*} x$, a variable. Then there exists x^i, for some $1\leq i\leq n$, such that

$$\mid\mathcal{D}\mid^{\mathrm{D}}=\mathsf{F}(\!(x^i)\!),$$

for some $\mathsf{F}\in\mathfrak{F}$.

Proof. By induction on the length n of the reduction $X \xrightarrow[\eta]{*} x$. The case $n=0$ is trivial. If $n>0$, then

$$X=\lambda\vec{z}.(\lambda y.x\vec{Z}Y),$$

where $Y \xrightarrow[\eta]{*} y$ and $Z_i \xrightarrow[\eta]{*} z_i$, for any $Z_i\in\vec{Z}$. Let $B=A\cup\{x\!:\sigma_1\cap\cdots\cap\sigma_n\}$. In such case, the derivation $\mathcal{D}$ has the shape

$$\dfrac{\begin{array}{c}\dfrac{\dfrac{\dfrac{\mathcal{D}_0}{B_2\vdash_{S'}x\vec{Z}\!:(\tau_1\cap\cdots\cap\tau_m)\to\tau} \quad \dfrac{\mathcal{D}_1}{B_2\vdash_{S'}Y_1\!:\tau_1} \quad \ldots \quad \dfrac{\mathcal{D}_m}{B_2\vdash_{S'}Y_m\!:\tau_m}}{B_2\vdash_{S'}x\vec{Z}Y\!:\tau}}{B_1\vdash_{S'}\lambda y.x\vec{Z}Y\!:\rho}\\ \vdots\end{array}}{B\vdash_{S'}\lambda\vec{z}.(\lambda y.x\vec{Z}Y)\!:\sigma}$$

where, for $i = 1, \dots, m$,

$$Y_i \xrightarrow[\eta]{*} Y. \tag{16}$$

We observe that $B_2 = B_1 \cup \{y{:}\varphi_1 \cap \dots \cap \varphi_r\}$. By inductive hypothesis, there exists

$$\mathsf{F}_0, \mathsf{F}_1, \dots, \mathsf{F}_m \in \mathfrak{F}$$

such that

$$\left| \begin{array}{c} \dfrac{\mathcal{D}_0}{B_2 \vdash_{S'} x\vec{Z}{:}\,(\tau_1 \cap \dots \cap \tau_m) \to \tau} \\ \vdots \\ \overline{B \vdash_{S'} \lambda\vec{z}.x\vec{Z}{:}\sigma'} \end{array} \right|^{\mathrm{D}} = \mathsf{F}_0(\!(x^i)\!)$$

and, for $k = 1, \dots, m$,

$$\left| \dfrac{\mathcal{D}_k}{B_2 \vdash_{S'} Y_k{:}\,\tau_k} \right|^{\mathrm{D}} = \mathsf{F}_k(\!(y^{g(k)})\!),$$

for some function $g : \{1, \dots, m\} \to \{1, \dots, r\}$. Now,

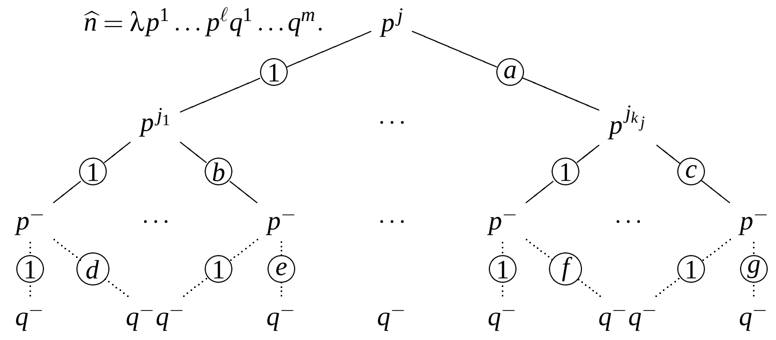
$$\mathsf{F}_0 \equiv \lambda a\vec{w}.F_0',$$

and hence $|\,\mathcal{D}\,|^{\mathrm{D}} = \mathsf{F}(\!(x^i)\!)$, where

$$\mathsf{F} \equiv \lambda a\vec{w}y^1 \dots y^r.F_0'(\mathsf{F}_1(\!(y^{g(1)})\!)) \dots (\mathsf{F}_m(\!(y^{g(m)})\!)).$$

$\square$

We recall that pseudonumerals have the following Böhm tree, where $a, \dots, g \in \mathbb{N}$:



(all paths stop at the same depth)

A subtree in such a tree is identified by a sequence $\mathtt{s}$ of integers, as shown above, with the empty sequence identifying the whole tree. Given a pseudonumeral $\mathtt{p}$, let us denote by $\mathtt{Seq}(\mathtt{p})$ the set of sequences identifying all nodes in the Böhm tree of $\mathtt{p}$. Moreover, we denote by $\mathtt{p@s}$ the subtree of $\mathtt{p}$ identified by $\mathtt{s}$. An *extended pseudonumeral* is obtained from a pseudonumeral $\mathtt{p}$ substituting in the node $\mathtt{p@s}$, for any $\mathtt{s} \in \mathtt{Seq}(\mathtt{p})$, the label $r \in \{p^1, \dots, p^\ell, q^1, \dots, q^m\}$ with

$$\mathsf{F}_{\mathtt{s}}(\!(r)\!),$$

where $\mathsf{F}_{\mathtt{s}} \in \mathfrak{F}$ is a hereditary finite combinator which depends on $\mathtt{s}$.

Lemma 7.20. Let $\mathcal{D}$ be a derivation of the typing $A \vdash_{S'} \mathsf{b}'_n{:}\sigma$, where b'_n is an η-expansion of the body of the Church numeral $\mathsf{c}_n \equiv \lambda pq.p^n q$. Then $|\mathcal{D}|^{\mathrm{D}}$ is an extended pseudonumeral.

Proof. By induction on n. The case $n = 0$ follows from Lemma 7.19. In the case $n = k+1$, b'_n has the shape

$$\mathsf{b}'_{k+1} \equiv \lambda z_1 \dots z_m.x(\mathsf{b}'_k)Z_1 \dots Z_m,$$

where, for $i = 1,\dots,m$, $Z_i \xrightarrow[\eta]{*} z_i$. The thesis follows by induction and by invoking Lemma 7.19 over $Z_1,\dots,Z_m$. $\square$

A decoder for extended pseudonumerals can be constructed exactly in the same way as in Lemma 6.7.

Lemma 7.21. Let τ be a Church numeral type in $\lambda^{E}_{\cap}$. Then there exist a term $D_{[\tau]}$ and a basis $B_{[\tau]}$ such that

(i) $B_{[\tau]} \vdash_{\rightarrow} D_{[\tau]}{:}\ |\tau|^{\mathrm{T}} \rightarrow (o \rightarrow o) \rightarrow o \rightarrow o$;

(ii) $D_{[\tau]}\mathsf{p}_n \xrightarrow[\beta]{*} \mathsf{c}_n$, for any $n \in \mathbb{N}$ and any extended pseudonumeral p_n resulting from the translation of a derivation of the typing $\vdash_e \mathsf{c}'_n{:}\,\tau$, where $\mathsf{c}'_n \xrightarrow[\eta]{*} \mathsf{c}_n$.

Proof. We build $D_{[\tau]}$ using the construction of Lemma 6.7. Therefore *(i)* holds. Moreover, $D_{[\tau]}$ prunes the tree of p_n exactly as in the previous case. In addition, $D_{[\tau]}\mathsf{p}_n$ has type

$$(o \rightarrow o) \rightarrow o \rightarrow o.$$

Therefore, all additional information introduced by hereditary finite combinators is erased by the decoder, which produces a Church numeral. $\square$

7.6. *Main Results*

We are now able to prove the main results of this Section.

Theorem 7.22. Every function $\varphi{:}\,\mathbb{N} \rightarrow \mathbb{N}$, uniformly representable in $\lambda^{S'}_{\cap}$, is uniformly representable in $\lambda_{\rightarrow}$.

Proof. Identical to the proof of Theorem 6.10. $\square$

Theorem 7.23. Every function $\varphi{:}\,\mathbb{N} \rightarrow \mathbb{N}$, uniformly representable in $\lambda^{BCD}_{\cap}$, is uniformly representable in $\lambda_{\rightarrow}$.

Proof. Let φ be uniformly representable in $\lambda^{BCD}_{\cap}$, say by a term M. By Theorem 7.8, φ is uniformly representable in $\lambda^{E}_{\cap}$, by the same term M. The thesis follows by Corollary 7.15 and Lemma 7.21, using the same construction of the proof of Theorem 6.10; observe that the encoder does not need to be modified since the input type coming from Corollary 7.15 is a full numeral type in $\lambda^{S}_{\cap}$. $\square$

8. Proof of Proposition 7.11

Definition 8.1. Let $\tau = \mu \to \rho \to \tau'$ be a numeral type in the essential system, and let

$$\mu = \mu_1 \cap \ldots \cap \mu_l, \rho = \rho_1 \cap \ldots \cap \rho_m, \mu_i = \mu_1^i \cap \ldots \cap \mu_{k_i}^i \to \mu_0^i \ (1 \leq i \leq l).$$

Define:

— $T_\tau = \{\tau', \rho_j, \mu_h^i \mid 1 \leq j \leq m,\ 1 \leq i \leq l,\ 0 \leq h \leq k_i\}$;
— $I_\tau = \{(\rho,\delta) \mid \delta \in T_\tau\} \bigcup \{(\mu, \mu_1^i \cap \ldots \cap \mu_{k_i}^i \to \delta) \mid 1 \leq i \leq l,\ \delta \in T_\tau\}$;
— $B_\tau = \{p : \mu, q : \rho\}$.

For $n \in N$, let $\mathsf{b}_n = p^n q$ be the body of the Church numeral $\mathsf{c}_n = \lambda pq.\ p^n q$. If $\mathcal{D}$ is a derivation of a typing in the essential system, let $I(\mathcal{D})$ be the set

$$I(\mathcal{D}) = \{(\gamma,\gamma') \mid \gamma \leq_{\mathcal{E}} \gamma' \dashv_{\surd\surd} \rceil\dashv\nabla\int \rangle\backslash\ \mathcal{D}\}.$$

Lemma 8.2. If τ is a numeral type in the essential system, then for all $\delta \in T_\tau$ and for all $n \in N$, if there exists a derivation $\mathcal{D}_n$ of the typing $B_\tau \vdash_e \mathsf{b}_n : \delta$, then there exists a derivation $\mathcal{D}_n'$ of the typing $B_\tau \vdash_e \mathsf{b}_n : \delta$ such that $I(\mathcal{D}_n') \subseteq I_\tau$.

Proof. The proof is by induction on n. If $n = 0$, and $\delta \in T_\tau$, then any derivation $\mathcal{D}_0[B_\tau \vdash_e \mathsf{b}_0 : \delta]$ is obviously such that $I(\mathcal{D}_0) \subseteq I_\tau$.

Let $n = k+1$. $\delta \in T_\tau$, and $\mathcal{D}_n[B_\tau \vdash_e \mathsf{b}_n : \delta]$. The leftmost premise of the last rule of $\mathcal{D}_n$ is $B_\tau \vdash_e p : \gamma$ for some strict $\gamma = \gamma_1 \cap \ldots \cap \gamma_s \to \delta \geq_E \mu$. By (vB93, Lemma 5.1.2 (v)), we have that $\gamma \geq \mu_1^i \cap \ldots \cap \mu_{k_i}^i \to \mu_0^i$ for some $1 \leq i \leq l$. Hence $\gamma_1 \cap \ldots \cap \gamma_s \leq_E \mu_1^i \cap \ldots \cap \mu_{k_i}^i$ and $\mu_0^i \leq_E \delta$, by (vB93, Lemma 5.1.2 (iv)). Moreover, for $1 \leq r \leq s$, there exists $\mathcal{D}_k^r[B_\tau \vdash_e \mathsf{b}_k : \gamma_r]$.

Now, since $\gamma_1 \cap \ldots \cap \gamma_s \leq_E \mu_1^i \cap \ldots \cap \mu_{k_i}^i$ we have that, for all $1 \leq h \leq k_i$, there exists $1 \leq j \leq s$ such that $\gamma_j \leq_E \mu_h$, (vB93) 5.1.3 (i). This implies that, for all $1 \leq h \leq k_i$, there exists a derivation $\mathcal{E}_k^h[B_\tau \vdash_e \mathsf{b}_k : \mu_h^i]$, by (vB93) 5.1.6. We can now use the inductive hypothesis, and conclude that, for all $1 \leq h \leq k_i$ there exists a derivation $\mathcal{F}_k^h[B_\tau \vdash_e \mathsf{b}_k : \mu_h^i]$ such that $I(\mathcal{F}_k^h) \subseteq I_\tau$.

We can now construct easily the required $\mathcal{D}_n'[B_\tau \vdash_e \mathsf{b}_n : \delta]$, such that $I(\mathcal{D}_n') \subseteq I_\tau$:

$$\dfrac{\dfrac{B_\tau \vdash_e p : \mu \quad \mu \leq_E \mu_1^i \cap \ldots \cap \mu_{k_i}^i \to \delta}{B_\tau \vdash_e p : \mu_1^i \cap \ldots \cap \mu_{k_i}^i \to \delta} \qquad \dfrac{\mathcal{F}_k^1}{B_\tau \vdash_e \mathsf{b}_k : \mu_1^i} \quad \ldots \quad \dfrac{\mathcal{F}_k^{k_i}}{B_\tau \vdash_e \mathsf{b}_k : \mu_{k_i}^i}}{B_\tau \vdash_e \mathsf{b}_{k+1} : \delta}$$

and we are done. □

Corollary 8.3. If $\tau = \mu \to \rho \to \tau'$ is a full numeral type in the essential system, then for all $n \in N$, there exists a derivation $\mathcal{D}_n[B_\tau \vdash_e \mathsf{b}_n : \tau']$ such that $I(\mathcal{D}_n) \subseteq I_\tau$.

Proof. Take $\delta = \tau'$ in the lemma above. □

Corollary 8.4. If $\tau = \mu \to \rho \to \tau'$ is a full numeral type in the essential system, then there exists a type $\tau^\bullet \leq_E \tau$ which is a full numeral type in the strict system.

Proof. Let $I = \bigcup_{n \in N} I(\mathcal{D}_n)$, where $\mathcal{D}_n[B_\tau \vdash_e \mathsf{b}_n : \tau']$ is given by the previous corollary. I is a finite set. Define the intersection types $\overline{\mu}$ and $\overline{\rho}$ as follows:

$\overline{\mu} = \bigcap_{(\mu,\delta)\in I} \delta$,
$\overline{\rho} = \bigcap_{(\rho,\delta)\in I} \delta$,
and let $B_{\tau^\bullet} = \{p : \overline{\mu}, q : \overline{\rho}\}$. It is easy to conclude that, for all $n \in N$, $B_{\tau^\bullet} \vdash_S \mathsf{b}_n : \tau'$ has a derivation, which is essentially the same as $\mathcal{D}_n$, up to the substitutions:

$$\frac{p : \mu \quad \mu \leq_E \delta}{p : \delta} \quad \Rightarrow \quad \frac{p : \ldots \cap \delta \cap \ldots}{p : \delta}$$

(and similarly for q and ρ).

Hence $\tau^\bullet = \overline{\mu} \to \overline{\rho} \to \tau'$ is a full numeral type in the strict system.

The relation $\tau^\bullet \leq_E \tau$ holds since, if $(\mu,\delta) \in I$ (resp. $(\rho,\delta) \in I$), then $\mu \leq_E \delta$ (resp. $\rho \leq_E \delta$), hence $\mu \leq_E \overline{\mu}$, $\rho \leq_E \overline{\rho}$, and finally $\tau^\bullet \leq_E \tau$. $\square$

We extend the notion of fullness to types in $\mathit{Type}^S_\cap$ as follows:

Definition 8.5. A type $\sigma = \sigma_1 \cap \ldots \cap \sigma_k \in \mathit{Type}^S_\cap$ is a *full numeral type* in $\lambda^E_\cap$ (resp. in $\lambda^S_\cap$) if all the σ_i are full numeral types in $\lambda^E_\cap$ (resp. in $\lambda^S_\cap$).

Proposition 8.6. If $\sigma = \sigma_1 \cap \ldots \cap \sigma_k \in \mathit{Type}^S_\cap$ is a full numeral type in $\lambda^E_\cap$, then $\sigma^\bullet = \sigma^\bullet_1 \cap \ldots \cap \sigma^\bullet_k$ is a full numeral type in $\lambda^S_\cap$, and $\sigma^\bullet \leq_E \sigma$.

Proof. For $1 \leq i \leq n$, $\sigma^\bullet_i \leq_E \sigma_i$ by Corollary 8.4. The statement $\sigma_1 \cap \ldots \cap \sigma_m \leq_E \tau_1 \cap \ldots \cap \tau_n$ follows easily from the clauses 2, 3 and 4 of the definition of $\leq_E$. $\square$

9. Concluding Remarks

A new technique has been proposed to compare computational aspects of typed lambda calculi. The presented technique, which is syntactic in nature, has been successfully applied to obtain a new proof of strong normalization, and to characterize definable functions in intersection type systems. Several directions for future work are suggested by this new approach.

It is interesting to investigate the algebraic structure of the redundant representation of numbers, which naturally comes out of our translation function, since they implicitly define recursive schemas. In this work we chose Church numerals as representation of integers. It would be interesting to investigate whether our results can be extended to arbitrary numeral systems.

Expressiveness is not only a matter of definability. A final remark is concerned with complexity: the term that we construct via $|\cdot|^{\mathrm{D}}$ is much more complex than the original one, typeable in the intersection type discipline. It would be interesting to analyze whether simply typed representations of functions lead to more complex "algorithms".

Acknowledgments

We are grateful to Mario Coppo for discussions about the topics of this paper, and to four anonymous referees for their useful remarks.

References

H. P. Barendregt. *The lambda calculus. Its syntax and semantics.* North-Holland Publishing Co., Amsterdam, revised edition, 1984.

H. P. Barendregt. Lambda calculi with types. In *Handbook of logic in computer science, Vol. 2*, pages 117–309. Oxford Univ. Press, New York, 1992.

H.P. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *J. Symbolic Logic*, 48(4):931–940 (1984), 1983.

A. Bucciarelli, S. De Lorenzis, A. Piperno, and I. Salvo. Some computational properties of intersection types. In *Proceedings of the Fourteenth Symposium on Logic in Computer Science (LICS'99)*, pages 109–118. IEEE Computer Society Press, 1999.

M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame J. Formal Logic*, 21(4):685–693, 1980.

M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Functional characters of solvable terms. *Z. Math. Logik Grundlag. Math.*, 27(1):45–58, 1981.

H. B. Curry and R. Feys. *Combinatory logic. Vol. I.* North-Holland Publishing Co., Amsterdam, 1968.

G. Castagna, G. Ghelli, and G. Longo. A calculus for overloaded functions with subtyping. *Inform. and Comput.*, 117(1):115–135, 1995.

A. Church. A formulation of the simple theory of types. *J. Symbolic logic*, 5:56–68, 1940.

H.B. Curry. Functionality in Combinatory Logic. In *Proc. Nat. Acad Science USA 20*, pages 584–590, 1934.

S. Fortune, D. Leivant, and M. O'Donnell. The expressiveness of simple and second-order type structures. *J. Assoc. Comput. Mach.*, 30(1):151–185, 1983.

R. O. Gandy. An early proof of normalization by A. M. Turing. In *To H. B. Curry: essays on combinatory logic, lambda calculus and formalism*, pages 453–455. Academic Press, London, 1980.

R. O. Gandy. Proofs of strong normalization. In *To H. B. Curry: essays on combinatory logic, lambda calculus and formalism*, pages 457–477. Academic Press, London, 1980.

J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In *Proceedings of the Second Scandinavian Logic Symposium (Univ. Oslo, Oslo, 1970)*, pages 63–92. Studies in Logic and the Foundations of Mathematics, Vol. 63, Amsterdam, 1971. North-Holland.

J. R. Hindley. The simple semantics for Coppo-Dezani-Sallé types. In *International symposium on programming (Turin, 1982)*, pages 212–226. Springer, Berlin, 1982.

Z. Khasidashvili and A. Piperno. Normalization of typable terms by superdevelopments. In *Computer science logic (Brno, 1998)*, pages 260–282. Springer, Berlin, 1999.

A.J. Kfoury and J.B. Wells. New Notions of Reduction and Non-Semantic Proofs of β-Strong Normalization in Typed λ-Calculi. In *Proceedings of the Tenth Symposium on Logic in Computer Science (LICS'95)*, pages 311–321. IEEE Computer Society Press, 1995.

A.J. Kfoury and J.B. Wells. Principality and Decidable Type Inference for Finite-Rank Intersection Types. In *Proceedings of the 26th ACM Symposium on Principles of Programming Languages, POPL'99*, pages 161–174. ACM Press, 1999.

D. Leivant. Discrete Polymorphism. In *Proc. of the ACM conference on Lisp and Functional Programming*, pages 288–297, 1990.

D. Leivant. Functions over free algebras definable in the simple typed lambda calculus. *Theoret. Comput. Sci.*, 121(1-2):309–321, 1993. A collection of contributions in honour of Corrado Böhm on the occasion of his 70th birthday.

J.C. Reynolds. Towards a theory of type structure. In *Programming Symposium (Proc. Colloq. Programmation, Paris, 1974)*, pages 408–425. Lecture Notes in Comput. Sci., Vol. 19. Springer, Berlin, 1974.

J.C. Reynolds. Design of the programming language Forsythe. Technical Report CMU–CS–96–146, Carnegie Mellon University, 1996.

J.C. Reynolds. Replacing complexity with generality: The programming language Forsythe. Technical report, Marked Carnegie Mellon University, 1996.

P. Sallé. Une extension de la théorie des types en λ-calcul. In *Automata, languages and programming (Fifth Internat. Colloq., Udine, 1978)*, pages 398–410. Springer, Berlin, 1978.

H. Schwichtenberg. Definierbare Funktionen im λ-Kalkül mit Typen. *Arch. Math. Logik Grundlagenforsch.*, 17(3-4):113–114, 1975/76.

R. Statman. The typed λ-calculus is not elementary recursive. *Theoret. Comput. Sci.*, 9(1):73–81, 1979.

R. Statman. Completeness, invariance and λ-definability. *J. Symbolic Logic*, 47(1):17–26, 1982.

W. W. Tait. Intensional interpretations of functionals of finite type. I. *J. Symbolic Logic*, 32:198–212, 1967.

S. van Bakel. Complete restrictions of the intersection type discipline. *Theoret. Comput. Sci.*, 102(1):135–163, 1992.

S. van Bakel. *Intersection Type Disciplines in Lambda Calculus and Applicative Term Rewriting Systems*. PhD thesis, Matematisch Centrum Amsterdam, 1993.

M. Zaionc. λ-definability on free algebras. *Ann. Pure Appl. Logic*, 51(3):279–300, 1991.