Some Computational Properties of Intersection Types*
(Extended Abstract)

Antonio Bucciarelli, Silvia De Lorenzis, Adolfo Piperno, Ivano Salvo
Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza",
Via Salaria 113, 00198 Roma, Italy.
{*buccia, piperno, salvo*}*@dsi.uniroma1.it*

Abstract

This paper presents a new method for comparing computational properties of λ*-terms typeable with intersection types with respect to terms typeable with Curry types. In particular, strong normalization and* λ*-definability are investigated. A translation is introduced from intersection typing derivations to Curry typeable terms; the main feature of the proposed technique is that the translation 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 naturally leads to prove strong normalization in the intersection system by means of purely syntactical techniques. In addition, the presented method enables us to give a proof of a conjecture proposed by Leivant in 1990, namely that all functions uniformly definable using intersection types are already definable using Curry types.*

1. Introduction

The λ-calculus originated 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 to represent a function application. This makes the system so powerful to represent all computable functions. When types are added to the calculus, it is possible to consider them as syntactical objects assigned to pure terms in order to give a description of the functional behaviour of expressions. Two type systems are investigated in this paper: simple types, which were introduced by Curry in [8], and a variant of intersection types; intersection types originate in the works of Barendregt, Coppo and Dezani [6, 5]; the system we will use in this paper, namely *strict intersection types*, has been introduced in [7], and has received a systematic treatment in [1, 2].

The expressive power of a type system can be analyzed from different standpoints. From one hand, one may consider the class of terms typeable in the system; from the other hand one may analyze the definable class of (numeric) functions. From the first perspective, simple types are much less expressive than 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 universal type. Although strict types are a proper subset of intersection types, they preserve, from the point of typeability, the expressive power of the whole system [1, 2].

In this paper, we will 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 in the case of the typeability perspective.

We denote by $\overline{n}$ the n-th Church numeral ($\overline{n} \equiv \lambda xy.x^n y$). We briefly recall that, given a typed lambda calculus λ_T and a numeric function $\varphi : \mathbf{N}^k \to \mathbf{N}$, we say that a lambda term M represents (non uniformly) φ if for every $n_1, \ldots, n_k \in \mathbf{N}$, $M\overline{n}_1 \ldots \overline{n}_k$ can be typed in λ_T and $M\overline{n}_1 \ldots \overline{n}_k \Leftrightarrow\!\!\rightarrow^*_\beta \overline{\varphi(n_1, \ldots, n_k)}$; a natural condition to impose is that the types of arguments do not depend on their values: we say that M represents φ *uniformly* in λ_T, if there are types $\tau_1, \ldots, \tau_k, \tau$ such that, for every $n_1, \ldots, n_k \in \mathbf{N}$, $M\overline{n}_1 \ldots \overline{n}_k$ can be typed in λ_T with τ, with types τ_i assigned to $\overline{n}_i (i \in \{1, \ldots, k\})$ and $M\overline{n}_1 \ldots \overline{n}_k \Leftrightarrow\!\!\rightarrow^*_\beta \overline{\varphi(n_1, \ldots, n_k)}$ [17].

The severe restrictions imposed by the structure of Curry types allows the simply typed λ-calculus to uniformly represent elementary functions, only. Indeed, the class of representable functions has been characterized in [19, 21, 17]. A first attempt to compare this characteriza-

*This work has been partially supported by MURST 40% grants.

tion with computational properties of intersection types appeared in [16], where it is proved that all functions representable uniformly in that system are elementary (elementary functions are a strict subset of total recursive ones), whereas all total computable functions are representable non uniformly. In addition, starting from the result above, Leivant conjectured that the class of functions uniformly representable in the intersection discipline coincides with the class of functions definable in Curry system. The proof of this conjecture, in the case of strict intersection types, is one of the main achievements of this paper.

It has to be noted that Leivant's results have a purely semantical nature, since the considered systems are compared by investigating the class of definable functions. In contrast, we will obtain our results using syntactical techniques, only. We believe that a syntactical approach gives a more direct understanding of the relationships between different calculi. In particular, we will show that, for any term T typeable with strict intersection types, and for any of its typing derivations D, there exists a term T'_D, which is typeable in Curry system and which is able to "represent" the whole computation of T. 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 T by means of reductions of T'_D.

As a first result given by our method, we will present a new proof of the strong normalization property for intersection types (without universal type). 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 [9]). Simply typed λ-calculus allows for a normalization proof which assigns a decreasing metric to terms during reduction [10, 11]. On the other hand, normalization in polymorphic λ-calculi is usually proven using variants of the so-called *computability* technique ([20]), which has a merely semantical nature: consider Girard-Reynolds second order λ-calculus [12, 18] as an example, but also vanBakel's proof for the system considered in this paper [2].

As a matter of fact, we will present a normalization proof for λ-calculus with intersection types which only makes use of syntactical techniques, in that it (syntactically) reduces the strong normalization problem in the presence of intersection types to the case of Curry types. Different syntactical approaches and normalization proofs for λ-calculus with intersection types are [14] and [15].

Our technique will then allow us to prove Leivant's conjecture for strict intersection types. We will proceed as follows. In the next section, we will briefly describe the type systems we are interested in. 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. Moreover, 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 codings 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, in section 4.3, we define Curry typeable terms which realize the mentioned coding and the corresponding decoding, thus allowing for a proof of Leivant's conjecture in the case of strict intersection types. Some remarks and directions for further work will conclude the paper.

2. Basic Definitions

We assume the reader to be familiar with standard notations for the untyped lambda calculus. Terms will be considered modulo α-equivalence. We also assume that every lambda term M obeys the restriction that no variable is bound more than once and that no variable occurs both free and bound in M. For every natural number n, the n-th Church numeral is the lambda term $\overline{n} \equiv \lambda pq.p^n q$. We use small greek letters to denote types, with the convention that α, β and γ denote type variables.

2.1. The Curry Type Assignment System.

Simple (or Curry) types are generated using the following grammar:

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

where α ranges over a countable set of type variables. We call $Type_{\to}$ the set of types resulting from (1). As usual, $\sigma_1 \to \sigma_2 \to \cdots \to \sigma_n \to \tau$ is an abbreviation for $\sigma_1 \to (\sigma_2 \to (\cdots(\sigma_n \to \tau)\cdots))$. Note that a type σ always has the shape

$$\sigma_1 \to \sigma_2 \to \cdots \to \sigma_n \to \alpha,$$

for some type variable α and $n \geq 0$.

Definition 2.1 The *Curry type assignment system* ($\lambda_{\to}$) proves statements (*typings*) of the kind $A \vdash_{\to} t{:}\sigma$, where A is a *basis* (a partial function from term variables to types),

$t \in \Lambda$ is the *subject* and $\sigma \in \mathit{Type}_{\to}$ is the *predicate* of the typing. The system $\lambda_{\to}$ consists of the following rules:

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

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

$$(\to\mathrm{E})_{\to} \quad \frac{A \vdash_{\to} t{:}\sigma \to \tau \quad A \vdash_{\to} s{:}\sigma}{A \vdash_{\to} ts{:}\tau}$$

A term t has a typing, or, equivalently, t is typeable, if there exists a basis A and a type σ such that $A \vdash_{\to} t{:}\sigma$. The set of terms typeable in $\lambda_{\to}$ will be denoted by $\Lambda_{\to}$. Moreover, $D{:}A \vdash_{\to} t{:}\sigma$ (or $D_{A \vdash_{\to} t:\sigma}$) denotes a derivation in $\lambda_{\to}$ proving the typing $A \vdash_{\to} t{:}\sigma$.

2.2. The Strict Intersection Type Assignment System.

Following the same approach as [1], we define a restricted version of the Intersection Type Assignment System of Coppo and Dezani [6]. This restricted version will consist of a restricted set of types, in which the type constructor $\cap$ can appear only on the left hand side of an arrow type scheme, and of a restricted set of inference rules.

Strict intersection types are generated using the following grammar:

$$\begin{array}{l} \sigma ::= \tau_1 \cap \ldots \cap \tau_n \ \ (n \geq 1) \\ \tau ::= \alpha \mid (\sigma \to \tau) \end{array} \qquad (2)$$

We call $\mathit{Type}^S_{\cap}$ (resp. $\mathit{Type}_{\cap}$) the set of types resulting from (2) with startsymbol τ (resp. σ).

Definition 2.2 The *Strict Intersection type assignment system* ($\lambda^S_{\cap}$) proves statements (*typings*) of the kind: $A \vdash_{\cap} M{:}\tau$, where A is a *basis*, i.e. a partial function from term variables to $\mathit{Type}_{\cap}$, $M \in \Lambda$ and $\tau \in \mathit{Type}^s_{\cap}$. The system $\lambda^S_{\cap}$ consists of the rules in Fig. 1.

We say that $D{:}A \vdash_{\cap} t{:}\tau$ (or $D_{A \vdash_{\cap} t:\tau}$) if D is a derivation in $\lambda^S_{\cap}$ proving the typing $A \vdash_{\cap} t{:}\tau$. Note that intersections of types may appear as predicates in bases, only; types assigned to terms always belong to $\mathit{Type}^s_{\cap}$.

The system $\lambda^S_{\cap}$ has been introduced in [1], where it is proven that the terms typeable in $\lambda^S_{\cap}$ are exactly the strongly normalizing ones. In particular, the original proof of the fact that any term typeable in $\lambda^S_{\cap}$ is strongly normalizing relies on a computability argument, semantic in nature. This is in sharp contrast with the case of $\lambda_{\to}$, where strong normalization can be proven by defining a (well founded) "measure" for typeable terms, which decreases strictly as reductions go on. In the next section, we introduce a translation of typings from $\lambda^S_{\cap}$ to $\lambda_{\to}$, with the property that any reduction path rooted in a term typeable in $\lambda^S_{\cap}$, is mimicked by a (longer, in general) reduction path rooted in a suitable simply typed term. An immediate corollary of this is an original, syntactic proof of strong normalization for $\lambda^S_{\cap}$. Moreover, since our translation allows to represent in $\lambda_{\to}$ any "computation" feasible in $\lambda^S_{\cap}$, it provides a framework for studying λ-definability in these systems. This will be the subject of section 4.

3. A Translation from $\lambda^S_{\cap}$ to $\lambda_{\to}$

In the definitions below, we use the following notational convention concerning variable names: we associate to any variable x, subject of a statement $x : \sigma_1 \cap \ldots \cap \sigma_n$ in a basis B, a set $x^1, \ldots, x^n$ of fresh variables, that we use in particular in translating the rules $(\mathrm{Var})_{\cap}$ and $(\to \mathrm{I})_{\cap}$.

Definition 3.1 Define the functions

$$\begin{array}{ll} T_T{:}\mathit{Type}^s_{\cap} \to \mathit{Type}_{\to} & \textit{(translation of types)}, \\ T_B{:}\mathit{Bases}_{\cap} \to \mathit{Bases}_{\to} & \textit{(translation of bases)} \end{array}$$

as follows:

$$\begin{array}{l} T_T(\alpha) = \alpha, \\ T_T((\sigma_1 \cap \ldots \cap \sigma_n) \to \tau) = \\ \qquad\qquad T_T(\sigma_1) \to \cdots \to T_T(\sigma_n) \to T_T(\tau); \end{array}$$

$$\begin{array}{l} T_B(\emptyset) = \emptyset, \\ T_B(A \cup \{x{:}\sigma_1 \cap \ldots \cap \sigma_n\}) = \\ \qquad\qquad T_B(A) \cup \{x^1{:}T_T(\sigma_1), \ldots, x^n{:}T_T(\sigma_n)\}. \end{array}$$

Definition 3.2 Define the function $T_D{:}\mathit{Der}_{\cap} \to \Lambda$ *(translation of derivations)* inductively as in Fig. 2.

An expected property of the translation(s) defined above is the following:

Lemma 3.3 *For any derivation D of the typing $A \vdash_{\cap} M{:}\sigma$ in $\lambda^S_{\cap}$,*

$$T_B(A) \vdash_{\to} T_D(D){:}T_T(\sigma)$$

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

Proof. By induction on the structure of the derivation D. Let us show for instance the case in which the last rule applied in D is $(\to\mathrm{I})_{\cap}$, i.e.:

$$\frac{A \cup \{x{:}\sigma_1 \cap \ldots \cap \sigma_n\} \vdash_{\cap} M{:}\tau}{A \vdash_{\cap} \lambda x.M{:}(\sigma_1 \cap \ldots \cap \sigma_n) \to \tau;}$$

$$(\mathrm{Var})_\cap \quad \frac{A(x) = \tau_1 \cap \ldots \cap \tau_n \quad 1 \le i \le n}{A \vdash_\cap x{:}\tau_i}$$

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

$$(\to\mathrm{E})_\cap \quad \frac{A \vdash_\cap M{:}(\tau_1 \cap \ldots \cap \tau_n) \to \tau \quad A \vdash_\cap N{:}\tau_1 \ldots A \vdash_\cap N{:}\tau_n}{A \vdash_\cap MN{:}\tau}$$

Figure 1. The System $\lambda^S_\cap$

$$T_D\left(\frac{A(x) = \sigma_1 \cap \ldots \cap \sigma_n}{A \vdash_\cap x{:}\sigma_i}\right) = x^i;$$

$$T_D\left(\frac{\dfrac{D_1}{A \cup \{x{:}\sigma_1 \cap \ldots \cap \sigma_n\} \vdash_\cap M{:}\tau}}{A \vdash_\cap \lambda x.M{:}(\sigma_1 \cap \ldots \cap \sigma_n) \to \tau}\right) = \left(\lambda x^1 \ldots x^n.T_D\left(\frac{D_1}{A \cup \{x{:}\sigma_1 \cap \ldots \cap \sigma_n\} \vdash_\cap M{:}\tau}\right)\right);$$

$$T_D\left(\frac{\dfrac{D_0}{A \vdash_\cap M{:}(\tau_1 \cap \ldots \cap \tau_n) \to \tau} \quad \dfrac{D_1}{A \vdash_\cap N{:}\tau_1} \quad \ldots \quad \dfrac{D_n}{A \vdash_\cap N{:}\tau_n}}{A \vdash_\cap MN{:}\tau}\right) =$$

$$T_D\left(\frac{D_0}{A \vdash_\cap M{:}(\tau_1 \cap \ldots \cap \tau_n) \to \tau}\right) T_D\left(\frac{D_1}{A \vdash_\cap N{:}\tau_1}\right) \ldots T_D\left(\frac{D_n}{A \vdash_\cap N{:}\tau_n}\right).$$

Figure 2. Translation of Typings

let D' be a derivation of the typing $A \cup \{x{:}\sigma_1 \cap \ldots \cap \sigma_n\} \vdash_\cap M{:}\tau$ which appears in the premise of such rule. By induction hypothesis:

$$T_B(A) \cup \{x^1{:}T_T(\sigma_1), \ldots, x^n{:}T_T(\sigma_n)\} \vdash_\to T_D(D'){:}T_T(\tau)$$

so that

$$T_B(A) \vdash_\to \lambda x^1 \ldots x^n.T_D(D'){:} T_T(\sigma_1) \to \cdots \to T_T(\sigma_n) \to T_T(\tau).$$

The case is settled observing that

$$T_D(D) = \lambda x^1 \ldots x^n.T_D(D')$$

and

$$T_T((\sigma_1 \cap \ldots \cap \sigma_n) \to \tau) = T_T(\sigma_1) \to \cdots \to T_T(\sigma_n) \to T_T(\tau).$$

□

We have seen so far how to translate (derivations of) typings of $\lambda^S_\cap$ into terms of $\lambda_\to$. The crucial property of this translation is that it is preserved by β-reduction, as expressed in the following proposition:

Lemma 3.4 *Let D be a derivation of the typing $A \vdash_\cap M{:}\sigma$ in $\lambda^S_\cap$ and let $M \leftrightarrow\!\!\!\to_\beta N$. Then there exists a derivation D' of the typing $A \vdash_\cap N{:}\sigma$ such that*

$$T_D(D) \leftrightarrow\!\!\!\to^+_\beta T_D(D').$$

Proof. By induction on the structure of the derivation D. The only interesting case is when the last rule applied in D is $(\to\mathrm{E})_\cap$, and the redex is M, so that D has the shape

$$\frac{\dfrac{D_0}{A \vdash_\cap \lambda x.P{:}\sigma \to \tau} \quad \dfrac{D_1}{A \vdash_\cap Q{:}\tau_1} \quad \cdots \quad \dfrac{D_n}{A \vdash_\cap Q{:}\tau_n}}{A \vdash_\cap (\lambda x.P)Q{:}\tau} \qquad (3)$$

where $\sigma \equiv \tau_1 \cap \ldots \cap \tau_n$. The variable x appears in D_0 as subject of q instances (for some $q \ge 0$) of the rule $(\mathrm{Var})_\cap$. Hence, a set $\mathsf{Pairs}(D,x)$ can be defined as follows, where $1 \le j \le q$:

$$(i,j) \in \mathsf{Pairs}(D,x) \;\Leftrightarrow\; \frac{A_j(x) = \tau_1 \cap \ldots \cap \tau_n}{A_j \vdash_\cap x{:}\tau_i} \text{ appears in } D.$$

Clearly, for every $j \in \{1, \ldots, n\}$, $A_j \supseteq A$. By the *basis lemma* (see e.g. [4]), for any $(i,j) \in \mathsf{Pairs}(D,x)$ a derivation D^j_i of the typing $A_j \vdash_\cap Q{:}\tau_i$ is built replacing in D_i every occurrence of a basis B with $B \cap A_j$.

It follows that the derivation D' is obtained out of D_0 in two steps:

1. replacing every occurrence of

$$\frac{A_j(x) = \tau_1 \cap \ldots \cap \tau_n}{A_j \vdash_\cap x : \tau_i}$$

with D_i^j;

2. replacing in the subjects of the so obtained derivation every occurrence of the variable x with Q.

It comes out that D' is a derivation of the typing

$$A \vdash_\cap P[Q/x] : \tau.$$

Moreover, by a straightforward induction on the structure of D_0, observing that

$$\forall i, j.(i,j) \in \mathsf{Pairs}(D,x) \;\Rightarrow\; T_D(D_i^j) = T_D(D_i),$$

the following holds:

$$T_D(D') = T_D(D_0)[T_D(D_1)/x^1, \ldots, T_D(D_n)/x^n]. \qquad (4)$$

Now,

$$\begin{aligned} T_D(D) &= T_D\left(\frac{D_0}{A \vdash_\cap \lambda x.P : (\tau_1 \cap \ldots \cap \tau_n) \to \tau}\right) T_D(D_1) \ldots T_D(D_n) \\ &= (\lambda x^1 \ldots x^n.T_D(D_0))T_D(D_1) \ldots T_D(D_n) \\ &\Leftrightarrow\!\!\rightarrow_\beta^+ T_D(D_0)[T_D(D_1)/x^1, \ldots, T_D(D_n)/x^n] \\ &= T_D(D'), \text{ by (4)}, \end{aligned}$$

which proves the lemma. □

It is easy to see that the property of T_D just shown also holds for the reflexive and transitive closure of $\Leftrightarrow\!\!\rightarrow_\beta$.

Lemma 3.5 *Let D be a derivation for the typing $B \vdash_\cap M : \sigma$ in $\lambda_\cap^S$ and let $M \Leftrightarrow\!\!\rightarrow_\beta^* N$. Then there exists a derivation D' of the typing $B \vdash_\cap N : \sigma$ such that*

$$T_D(D) \Leftrightarrow\!\!\rightarrow_\beta^* T_D(D').$$

Proof. By induction on the length of the reduction of $M \Leftrightarrow\!\!\rightarrow_\beta^* N$, using Lemma 3.4. □

We are now able to prove the main result of this section.

Theorem 3.6 (Strong normalization) *For any $M \in \Lambda$, if M is typeable in $\lambda_\cap^S$, then every β-reduction path starting from M is finite.*

Proof. If M is typeable in $\lambda_\cap^S$, then there exists a basis B and a type σ such that $B \vdash_\cap M : \sigma$. Let D be a derivation of such typing. If M has an infinite β-reduction path, then, by Lemma 3.5, $T_D(D)$ has an infinite β-reduction path, too. But, by Lemma 3.3, $T_B(B) \vdash_\to T_D(D) : T_T(\sigma)$, so that $T_D(D)$ has a typing in $\lambda_\to$, hence it is strongly normalizing, a contradiction. □

4. λ-definability in $\lambda_\cap^S$ and $\lambda_\to$

Since we are able to map computations of terms typeable in $\lambda_\cap^S$ into computations of terms typeable in $\lambda_\to$ it is natural to ask whether our syntactic approach can be used to compare the expressive power of $\lambda_\to$ with respect to $\lambda_\cap^S$ from the point of view of representable functions.

Since all strongly normalizing terms are typeable in $\lambda_\cap^S$, it easy to show that there are functions, representable in $\lambda_\cap^S$, which are not representable in $\lambda_\to$. Take as an example the term $E = \lambda x.x(\lambda y.yx)x$. Taking as input a Church numeral $\overline{n}$, E reduces to $N \equiv \underbrace{\overline{n} \ldots \overline{n}}_{n+1}$ and hence yields as output the Church numeral $\overline{n^{n^{\cdot^{\cdot^{\cdot^{n}}}}}}$ (n times). Thus E computes a non–elementary function; moreover, for all n, $E\overline{n}$ is typeable in $\lambda_\cap^S$, because it is strongly normalizing. However, the typing of $E\overline{n}$ depends on n, as it is clear from the structure of N. This example shows that typings in $\lambda_\cap^S$ may be highly non–uniform. It is interesting to investigate the set of representable functions under a reasonable uniformity condition on typings [16]:

Definition 4.1 (Uniform Representation of Functions) Let λ_T be a typed lambda calculus and $\varphi : \mathbf{N}^k \to \mathbf{N}$ a numeric function. We say that a lambda term M represents φ *uniformly* in λ_T, if there are types $\tau_1, \ldots, \tau_k, \tau$ such that, for every $n_1, \ldots, n_k \in \mathbf{N}$, $M\overline{n}_1 \ldots \overline{n}_k$ can be typed in λ_T with τ, with types τ_i assigned to $\overline{n}_i$ ($i \in \{1, \ldots, k\}$) and

$$M\overline{n}_1 \ldots \overline{n}_k \Leftrightarrow\!\!\rightarrow_\beta^* \overline{\varphi(n_1, \ldots, n_k)}.$$

We call $\tau_1, \ldots, \tau_k$ the *input types*, and the type τ the *output type*.

In [16], Leivant proved that all functions uniformly representable in $\lambda_\cap^S$ are elementary, as it is the case for $\lambda_\to$. Moreover, he proposed the following conjecture.

Functions uniformly representable in $\lambda_\cap$ are already uniformly representable in $\lambda_\to$.

The rest of the paper is devoted to prove Leivant's conjecture in the case of strict intersection types.

From now on, for the sake of simplicity, we focus on unary functions; every result can be easily extended to the case of function with $k > 1$ arguments. Let $\varphi : \mathbf{N} \to \mathbf{N}$ be a numeric function uniformly represented in $\lambda_\cap^S$, by a term $M = \lambda x.M'$. In $\lambda_\cap^S$ a type assignable to M must have the shape $\tau_1 \cap \ldots \cap \tau_k \to \tau$, and moreover, by uniformity, each Church numeral has to be typed with $\tau_1, \ldots, \tau_k$, and each Church numeral $\overline{n}$ such that $\varphi(m) = n$ for some m has to be typed with τ. By definition of T_D, we have:

$$T_D(D_{\vdash_\cap M\overline{n}:\tau}) = T_D(D^0_{\vdash_\cap M:\tau_1\cap\ldots\cap\tau_k\to\tau})\, T_D(D^1_{\vdash_\cap \overline{n}:\tau_1}) \ldots T_D(D^k_{\vdash_\cap \overline{n}:\tau_k}). \quad (5)$$

Our aim is to use $\widehat{M} = T_D(D^0_{\vdash_\cap M:\tau_1\cap\ldots\cap\tau_k\to\tau})$ in order to find a term which represents φ in $\lambda_\to$. However, since in general the terms $\widehat{n_i} = T_D(D^i_{\vdash_\cap \overline{n}:\tau_i})$ and $\widehat{\varphi(n)} = T_D(D_{\vdash_\cap M\overline{n}:\tau})$ are not Church numerals, our approach requires the definition of suitable simply typeable terms, $D_{[\tau]}, E_{[\tau_1]}, \ldots, E_{[\tau_k]}$, such that

$$E_{[\tau_i]}\overline{n} = T_D(D_{\vdash_\cap \overline{n}:\tau_i}) \quad \text{the "encoders"}$$
$$D_{[\tau]}(T_D(D_{\vdash_\cap \overline{n}:\tau})) = \overline{n} \quad \text{the "decoder".}$$

and the term

$$P = \lambda x.D_{[\tau]}(\widehat{M}(E_{[\tau_1]}x) \ldots (E_{[\tau_k]}x)) \quad (6)$$

is Curry typeable, so that, for all n, $P\overline{n} \leftrightarrow\!\!\!\!\to^*_\beta \overline{\varphi(n)}$. Note that the terms $D_{[\tau]}, E_{[\tau_1]}, \ldots, E_{[\tau_k]}$ will be proven to have simple types. However, 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.

4.1. A Strengthened Uniformity Condition

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

Fact 4.2 *If $\tau = (\tau' \to \tau') \to \tau' \to \tau' \in Type^s_\cap$, then*

$$T_D(D_{\vdash_\cap \overline{n}:\tau}) \equiv \overline{n}.$$

Proof. Just observing that, since τ' is not an intersection, in every derivation of $\vdash_\cap \lambda pq.p^n q : \tau$ we use the statements $p : \tau' \to \tau'$ and $q : \tau'$, and hence the translation T_D does not generate new variables. □

Using this fact and the equality (5), we obtain the following:

Proposition 4.3 *Let M be a term that uniformly represent $\varphi : \boldsymbol{N} \to \boldsymbol{N}$ in $\lambda_\cap^S$, with type $\tau_1 \cap \ldots \cap \tau_k \to \tau$, and let $\tau_1, \ldots, \tau_k, \tau$ all be instances of $(\alpha \to \alpha) \to \alpha \to \alpha$. Then the term*

$$\widehat{M} \equiv T_D(D_{\vdash_\cap M:(\tau_1\cap\ldots\cap\tau_k)\to\tau})$$

uniformly represents in $\lambda_\to$ the function $\varphi' : \boldsymbol{N}^k \to \boldsymbol{N}$, such that for all $n \in \boldsymbol{N}$, we have: $\varphi'(n, \ldots, n) = \varphi(n)$.

Even in this simple case, the obtained representation might be unsatisfactory, since in general $\widehat{M}$ needs $k > 1$ copies of $\overline{n}$ to compute $\varphi(n)$. In effect, the function φ could be uniformly represented in $\lambda_\to$ by another term, totally unrelated to $\widehat{M}$, which does not require k copies of the input. This is shown by the following example.

Example 4.4 The function $\varphi(x) = x^x$ is trivially representable in $\lambda_\cap^S$ by the term $\omega = \lambda x.xx$, with type $\sigma = ((\tau \to \tau) \cap \tau) \to \tau$. The translation $T_D(D_{\vdash_\cap \omega:\sigma})$ gives the term $\lambda xy.xy$ which represents the binary exponential function in $\lambda_\to$. We now show a term typeable in $\lambda_\to$ that represent φ. Let $\tau_0 = o$ and $\tau_{i+1} = (\tau_i \to \tau_i) \to \tau_i \to \tau_i$. Consider the terms A (typed with $\tau_1 \to \tau_1 \to \tau_1$) and M (typed with $\tau_2 \to \tau_1 \to \tau_1$), which respectively compute addition and multiplication on natural numbers (we write types used in the derivation as superscript, to increase readability):

$$A = \lambda x^{\tau_1} y^{\tau_1} p^{o\to o} q^o.xp(ypq) \qquad M = \lambda x^{\tau_2} y^{\tau_1}.x(Ay)\overline{0}.$$

Then the term:

$$E = \lambda x^{\tau_2}.x(Mx)\overline{1}$$

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.

4.2. A Redundant Representation of Numbers

Let us consider the general case, when we do not require that types $\tau_1, \ldots, \tau_k, \tau$ are instances of $(\alpha \to \alpha) \to \alpha \to \alpha$. Throughout the rest of the section we use some notations introduced below.

Notation 4.5 We first observe that a type τ in $\lambda_\cap^S$ that can be assigned to a Church numeral $\overline{n}$ has the shape $\mu \to \rho \to \tau'$, where $\mu \equiv \mu_1 \cap \ldots \cap \mu_l$ and $\rho \equiv \rho_1 \cap \ldots \cap \rho_m$. Moreover every μ_i has the shape $\mu^i_1 \cap \ldots \cap \mu^i_{k_i} \to \mu^i_0$ (see Fig 3). To increase readability, for any type $\tau \in Type^s_\cap$, we denote $T_T(\tau)$ by $\widehat{\tau}$ and $T_D(D_{\vdash_\cap \overline{n}:\tau})$ by $\widehat{n}_{[\tau]}$.

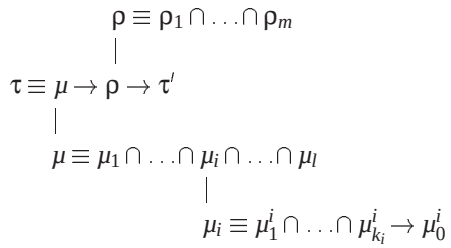

Figure 3. Type structure of Church numerals

The aim of the next definitions and propositions is to characterize the general shape of a term $\hat{n}_{[\tau]}$, namely to study the structure of translations of typing derivations of Church numerals.

Definition 4.6 Let τ be as in Fig. 3. We define a family of sets of intersection types, as follows:

- $T_\tau^0 = \{\rho_1, \ldots, \rho_m\}$,
- $T_\tau^{h+1} = \bigcup_{1 \le i \le l} \{\mu_0^i \mid \forall s, 1 \le s \le k_i, \mu_s^i \in T_\tau^h, \ \}$.

We define $T_\tau = \bigcup_n T_\tau^n$.

Some remarks on the previous definition:

- T_τ^n is exactly the set of types which can be assigned to the term $p^n q$ with basis $\{p : \mu, q : \rho\}$.
- If a given type σ belongs to T_τ^n for all n, then $\mu \to \rho \to \sigma$ can be assigned to all Church numerals.
- For all n, $T_\tau^n \subseteq \{\mu_0^1, \ldots, \mu_0^l, \rho_1, \ldots, \rho_m\}$

Example 4.7 In this example, we show that T_τ can contain a type σ such that the typing $\{p : \mu, q : \rho\} \vdash_\cap p^n q : \sigma$ is not derivable for every n, but there exist typings of $\{p : \mu, q : \rho\} \vdash_\cap p^n q : \tau'$, which can only be derived using the fact that $\{p : \mu, q : \rho\} \vdash_\cap p^k q : \sigma$, for some $k < n$. Consider $\mu \equiv (\alpha \to \beta) \cap (\gamma \to \beta) \cap (\alpha \to \gamma) \cap (\gamma \to \alpha)$ and $\rho \equiv \alpha \cap \beta$. It is easy to show that we can derive the typing: $\{p : \mu, q : \rho\} \vdash_\cap p^n q : \beta$, for all n. To do this, however, for $n > 0$ we have to derive the typing $\{p : \mu, q : \rho\} \vdash_\cap p^{n-1} q : \alpha$ or the typing $\{p : \mu, q : \rho\} \vdash_\cap p^{n-1} q : \gamma$. We can derive the former only if $n \Leftrightarrow 1$ is even and the latter only if $n \Leftrightarrow 1$ is odd. In this example we have

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

Remark 4.8 Let $T_\tau = \{\sigma_1, \ldots, \sigma_t\}$. We can assume w.l.o.g. that $\{\mu_1^i, \ldots, \mu_{k_i}^i, \mu_0^i\} \subseteq T_\tau$, for each $i \in \{1, \ldots, l\}$, otherwise μ_i is not useful in typings of any Church numeral (we never derive the judgment $B \vdash_\cap p : \mu_i$ in a derivation of $\vdash_\cap \overline{n} : \tau$).

Given a type as in Fig. 3, we can characterize the translations in $\lambda_\to$ of any possible derivation $D_{\{p:\mu,q:\rho\} \vdash_\cap p^n q:\sigma}$, for all n and for all $\sigma \in T_\tau$. In the following definition, q^j and p^j are term variables generated by the translation T_D (Def. 3.1). In particular, q^j is assigned type $\widehat{\rho}_j$, while p^j is assigned type $\widehat{\mu}_j$.

Definition 4.9 (Numeral bodies) Let $T_\tau = \{\sigma_1, \ldots, \sigma_t\}$. We define the family of terms $P^{\tau,\sigma_i} = \bigcup_{n \in \mathbf{N}} P_n^{\tau,\sigma_i}$ as follows:

$$P_0^{\tau,\sigma_i} = \{q^j \mid \rho_j = \sigma_i\},$$
$$P_{k+1}^{\tau,\sigma_i} = \{p^j \, Q_1 \ldots Q_{k_j} \mid 1 \le j \le l, \, \sigma_i = \mu_0^j, \quad Q_r \in P_k^{\tau,\mu_r^j}, \, 1 \le r \le k_j\},$$

Proposition 4.10 *For all $n \in \mathbf{N}$, for all $\sigma \in T_\tau$:*

$$\{T_D(D) \mid D_{\{p:\mu,q:\rho\} \vdash_\cap p^n q:\sigma}\} = P_n^{\tau,\sigma}$$

Proof. Induction on n. □

It comes out that the set N_n^τ of translations of possible typings of a Church numeral $\overline{n}$ with $\tau \equiv \mu \to \rho \to \tau'$ is exactly the set of terms $\lambda p_1 \ldots p_l q_1 \ldots q_m . M$, with $M \in P_n^{\tau,\tau'}$. Moreover, any $\hat{n} \in N_n^\tau$ shares with $\overline{n}$ the property of having a Böhm tree of depth n. Hence the set of terms N_n^τ, can be seen as a redundant representation of natural numbers. We will define encoders $E_{[\tau]}$ and decoders $D_{[\tau]}$ such that $E_{[\tau]}\overline{n} = T_D(D_{\vdash_\cap \overline{n}:\tau})$ for some derivation D and $D_{[\tau]}(T_D(D_{\vdash_\cap \overline{n}:\tau})) = \overline{n}$ for all D. A proof of Leivant's conjecture will immediately follow.

4.3. A Proof of Leivant's Conjecture

We first address the problem of finding the decoder $D_{[\tau]}$.

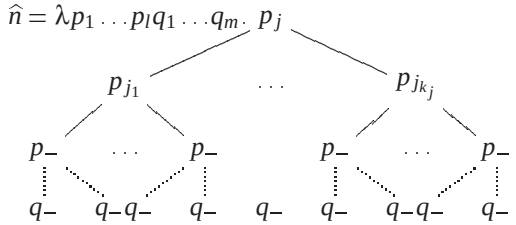

Figure 4. Structure of pseudonumerals

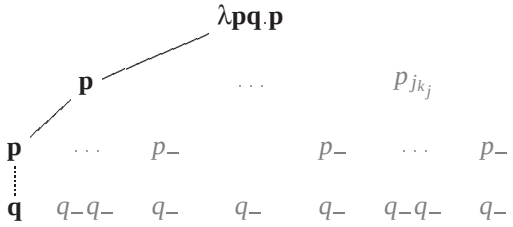

Figure 5. Extracting a Church numeral

Lemma 4.11 *Let τ be as in Fig. 3 and let $\widehat{n} = T_D(D_{\vdash_\cap \overline{n}:\tau})$, for some typing derivation D. There exists a Curry-typeable term $D_{[\tau]}$ and a basis $B_{[\tau]}$ such that $B_{[\tau]} \vdash_\to D_{[\tau]} : \widehat{\tau} \to ((o \to o) \to o \to o)$ and $D_{[\tau]}\widehat{n}_{[\tau]}$ reduces to $\overline{n}$, for any $\widehat{n}_{[\tau]}$.*

Proof. (Sketch) The idea is to prune the tree of the pseudonumeral $\widehat{n}_{[\tau]}$, whose shape is shown in Figure 4, keeping its leftmost branch and collapsing the non–leaf variables of this branch into a single one, hence reconstructing a Church numeral, see Figure 5. First, we observe that w.l.o.g. a term can be typed using only one type variable, say o. We consider the following terms, for $1 \leq i \leq l, 1 \leq j \leq m$:

$$P_i = \lambda x_1 \ldots x_{k_i} s_1 \ldots s_{a_i} . p(x_1 v_1^i \ldots v_{b_i}^i)$$

and

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

Observe that P_i has type $\widehat{\mu_i}$, where

- $\widehat{\mu_0^i} = \phi_1 \to \cdots \to \phi_{a_i} \to o$,
- $\widehat{\mu_1^i} = \psi_1 \to \cdots \to \psi_{b_i} \to o$,

for suitable $\phi_1, \ldots, \phi_{a_i}, \psi_1, \ldots, \psi_{b_i}$. Moreover, Q_j has type $\widehat{\rho_j} \equiv \chi_1 \to \cdots \to \chi_{r_j} \to o$, for suitable $\chi_1, \ldots, \chi_{r_j}$. Finally, we define:

$$D_{[\tau]} = \lambda x p q . x P_1 \ldots P_l Q_1 \ldots Q_m v_1 \ldots v_r,$$

where $\widehat{\tau'} \equiv \xi_1 \to \cdots \to \xi_r \to o$, for suitable $\xi_1, \ldots, \xi_r$. .

The new (free) variables are just used to allow a uniform typing for p and q with $o \to o$ and o, respectively. □

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

Lemma 4.12 *Let τ be such that for every n, the judgement $\vdash_\cap \overline{n} : \tau$ is derivable. Then there exists a Curry typeable term $E_{[\tau]}$, such that for any n, $E_{[\tau]}\overline{n} \Leftrightarrow_\beta^* \widehat{n}_{[\tau]}$.*

Proof. (Sketch) The idea is to construct a term that, given a numeral $\overline{n}$ as input, iteratively generates a tuple of pseudonumerals. Let τ be a type as in Fig. 3 and let $t = |T_\tau|$. At the k-th step of the iteration, the mentioned t-tuple has the shape

$$< \widehat{k}_{[\delta_1]}, \ldots, \widehat{k}_{[\delta_t]} >,$$

where $\delta_i = \mu \to \rho \to \sigma_i$, for all $i \in \{1, \ldots, t\}$ (we remark that $\tau = \delta_i$ for some i). We cannot restrict ourselves to the construction of pseudonumerals of type $\widehat{\tau}$ only, because the body of a pseudonumeral $\widehat{n}_{[\delta_i]}$ has as subterms, in general, some bodies of pseudonumerals $\widehat{m}_{[\delta_j]}$, with $m < n$ and $i \neq j$.

We show how to construct encoders under the hypothesis that for each $\sigma_i \in T_\tau$ and for each n, the judgement: $\vdash_\cap \overline{n} : \delta_i$ is derivable. Then we argue about how to approach the general case. Under this hypothesis, $T_\tau = \{\rho_1, \ldots, \rho_m\}$ and hence $t = m$, since types in T_τ are assignable to the body of $\overline{0}$. Moreover we can choose, for each $\sigma_j \in T_\tau$, a μ_i such that $\mu_i = \mu_1^i \cap \ldots \cap \mu_{k_i}^i \to \mu_0^i$, $\mu_0^i = \sigma_j$, and all μ_j^i are in T_τ. Such type does exist under the above hypothesis, since for all n, $T_\tau^n = T_\tau$. We rearrange indexes of types in T_τ in such a way that $i = j$ and we define, for each $1 \leq i \leq t$ an index function $g_i : \{1, \ldots, k_i\} \to \{1, \ldots, t\}$ such that $\sigma_{g_i(j)} = \mu_j^i$. We also use the convention that q^i has type $\widehat{\rho_i}$ and p^i has type $\widehat{\mu_1^i} \to \ldots \to \widehat{\mu_{k_i}^i} \to \widehat{\mu_0^i}$.

We now describe the behaviour of two terms Q and P (fully defined later), that will serve as arguments of $\overline{n}$ in the iteration outlined above.

1. Q is the t–tuple

   $$\lambda z . z \widehat{0}_{[\delta_1]} \ldots \widehat{0}_{[\delta_t]},$$

   which can be typed with

   $$\phi \equiv (\widehat{\delta}_1 \to \cdots \to \widehat{\delta}_t \to \gamma) \to \gamma,$$

   for an arbitrary type γ.

2. P is a term that takes as input a t–tuple of the form

   $$\lambda z . z \widehat{k}_{[\delta_1]} \ldots \widehat{k}_{[\delta_t]}$$

and gives as output the t-tuple:

$$\lambda z.z(\widehat{k+1})_{[\delta_1]}\ldots(\widehat{k+1})_{[\delta_t]}$$

We will see that P can be typed in $\lambda_{\to}$ with $\phi\to\phi$.

Let us consider the following terms ($1\le i\le t$):

Zeroes $Z_i=\lambda p^1\ldots p^l q^1\ldots q^m.q^i=\widehat{0}_{[\delta_i]}$

Successors

$$S_i=\lambda x_1\ldots x_{k_i}p^1\ldots p^l q^1\ldots q^m.p^i(x_1p^1\ldots p^lq^1\ldots q^m)\ldots(x_{k_i}p^1\ldots p^lq^1\ldots q^m)\ .$$

S_i is typeable with $\widehat{\delta}_{g_i(1)}\to\cdots\to\widehat{\delta}_{g_i(k_i)}\to\widehat{\delta}_i$ and it yields $(\widehat{k+1})_{[\delta_i]}$ when applied to $\widehat{k}_{[\delta_{g_i(1)}]},\ldots,\widehat{k}_{[\delta_{g_i(k_i)}]}$

We can now define Q and P as follows:

$$Q=\lambda z.zZ_1\ldots Z_t,$$
$$P=\lambda zw.z(\lambda x_1\ldots x_t.w(S_1x_{g_1(1)}\ldots x_{g_1(k_1)})\ldots(S_tx_{g_t(1)}\ldots x_{g_t(k_t)}))\ .$$

Typing:

- x_i with $\widehat{\delta}_i$,
- w with $\widehat{\delta}_1\to\ldots\to\widehat{\delta}_t\to\gamma$,
- z with $(\widehat{\delta}_1\to\ldots\to\widehat{\delta}_t\to\gamma)\to\gamma$,

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

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 judgement $\vdash_\cap\overline{n}\ :\ \mu\to\rho\to\sigma$ 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 appliable at later stages. Very roughly, this difficulty can be overcome by introducing boolean values representing the existence of pseudonumerals. When successors are looked-up in order to perform an iteration step, the existence of their arguments is checked and the first appliable successor is picked up. It is worth stressing that in this case we have to consider, for a given σ_i in T_τ, all the successors constructing a numeral of type $\widehat{\delta}_i$, whereas in the simple case the arbitrary choice of one of these was sufficient.

Finally, we can extract, from the constructed t–tuple, the pseudonumeral of type $\widehat{\tau}$, since we know its position in the t–tuple, say r, applying the term:

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

Observe that, by the hypothesis that, for every n, the judgement $\vdash_\cap\overline{n}:\tau$ is derivable, the pseudonumeral $\widehat{k}_{[\tau]}$ appears in the t–tuple at each stage of the iteration. Choosing $\gamma=\widehat{\tau}$, we can type in $\lambda_{\to}$ the encoder

$$E_{[\tau]}=\lambda x.xPQN$$

□

By properties of the translation function, the Curry typeable term $\widehat{M}$ takes as input $k\ge 1$ pseudonumerals of types $\widehat{\tau}_1,\ldots\widehat{\tau}_k$. 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. We thus use an encoder which constructs in parallel pseudonumerals of type $\widehat{\tau}_1,\ldots\widehat{\tau}_k$ and finally puts them into a k–tuple.

Lemma 4.13 *Let M be a term typeable in $\lambda^S_\cap$, which uniformly represents a numeric unary function φ. Let $\widehat{M}=T_D(D_{\vdash_\cap M:\tau_1\cap\ldots\cap\tau_k\to\tau})$. Then there exists a term $E_{[\tau_1,\ldots,\tau_k,\tau]}$, such that $E_{[\tau_1,\ldots,\tau_k,\tau]}\widehat{M}\overline{n}$ is a Curry typeable term which reduces to $\widehat{M}\widehat{n}_{[\tau_1]}\ldots\widehat{n}_{[\tau_k]}$.*

Proof. We use the construction of Lemma 4.12 taking the set of types $T=\biguplus_{i=1}^k T_{\tau_i}$ instead of a single T_τ. We extract, from the t–tuple generated by the term $\overline{n}PQ$, the k pseudonumerals of types $\widehat{\tau}_1,\ldots,\widehat{\tau}_k$, knowing their positions, say $r_1,\ldots,r_k$, in the t–tuple, using the term:

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

Choosing $\gamma=(\widehat{\tau}_1\to\ldots\to\widehat{\tau}_k\to\widehat{\tau})\to\widehat{\tau}$, we type N with

$$(\widehat{\delta}_1\to\ldots\to\widehat{\delta}_t\to\gamma),$$

so that the term $(\overline{n}PQ)N$ has type γ, typing $\overline{n}$ with $(\phi\to\phi)\to\phi\to\phi$. Since $\widehat{M}$ has type $\widehat{\tau}_1\to\ldots\to\widehat{\tau}_k\to\widehat{\tau}$ (from translation), the term $(\overline{n}PQ)N\widehat{M}$ is typeable in $\lambda_{\to}$ with $\widehat{\tau}$ and it reduces to $\widehat{M}\widehat{n}_{[\tau_1]}\ldots\widehat{n}_{[\tau_k]}$. Hence the term

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

satisfies the statement. □

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

Theorem 4.14 *Every function $\varphi:\mathbf{N}\to\mathbf{N}$, uniformly representable in $\lambda^S_\cap$, is uniformly representable in $\lambda_{\to}$.*

Proof. If M represents a function in $\lambda^S_\cap$, let $\widehat{M}$ be the term produced by the translation function. Using Lemmas 4.11 and 4.13, we set

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

which proves the theorem. □

5. Concluding remarks

A new technique has been proposed to compare computational aspects of typed λ-calculi. The presented (syntactical) technique has been successfully applied to the analysis of strong normalization and to the characterization of definable functions in the strict intersection type assignment system. Several directions for future work are suggested by the new approach. First, it would be interesting to investigate the possibility of defining a translation function for the whole intersection type assignment system, eliminating the restriction to strict types, which on its side allows for an easier, syntax directed system. Observe that Leivant's conjecture would be false in the full system only in the case that there exists a function with a uniform typing which is no longer uniform in the strict system.

Also, it seems interesting to investigate the algebraic structure of the redundant representation of numbers, which naturally comes out of our translation function.

References

[1] S. van Bakel, *Complete Restrictions of the Intersection Type Discipline*, Theoretical Computer Science 102 (1992) 135-163.

[2] S. van Bakel, *Intersection Type Disciplines in Lambda Calculus and Applicative Term Rewriting Systems*, PhD Dissertation, Amsterdam (1993).

[3] H. Barendregt, *The Lambda Calculus: its Syntax and Semantics*, North-Holland, Amsterdam, revised edition (1984).

[4] H. Barendregt, *Lambda Calculus with Types*, in Handbook of Logic in Computer Science, Volume II, Oxford University Press (1991).

[5] H. Barendregt, M. Coppo and M. Dezani-Ciancaglini, *A Filter Lambda Model and the Completeness of Type Assignment*, The Journal of Symbolic Logic 48(4) (1983) 931-940.

[6] M. Coppo and M. Dezani-Ciancaglini, *An Extension of the Basic Functionality Theory for the λ-Calculus*, Notre Dame Journal of Formal Logic, 21(4) (1980) 685-693.

[7] M. Coppo, M. Dezani-Ciancaglini and B. Venneri, *Functional Character of Solvable Terms*, Zeitschr. f. math. Logik und Grundlagen d. Math 27 (1981) 45-48.

[8] H.B. Curry, *Functionality in Combinatory Logic*, Proc. Nat. Acad. Sci. U.S.A. 20 (1934) 371-401.

[9] S. Fortune, D. Leivant and M. O'Donnell, *The Expressiveness of Simple and Second Order Type Structures*, Journal of the ACM 30 (1983) 151-185.

[10] R.O. Gandy, *Proofs of strong normalization*, in [13], 1980.

[11] R.O. Gandy, *An early proof of normalization by A.M.Turing*, in [13], 1980.

[12] J.-Y. Girard, *Une extension de l'interprétation de Gödel à l'analyse, et son application à l'elimination des coupures dans l'analyse et la thèorie des types*, Procs of 2nd Scandinavian Logic Symposium, North-Holland, 1971.

[13] *To H.B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, Hindley, J.R., Seldin, J.P., eds, Academic Press, 1980.

[14] A.J. Kfoury and J.B. Wells, *New Notions of Reduction and Non-Semantic Proofs of β-Strong Normalization in Typed λ-calculi*, Symposium on Logic in Computer Science (LICS'95), IEEE Computer Society Press, 1995.

[15] Z. Khasidashvili and A. Piperno, *Normalization of typable terms by superevelopments*, Computer Science Logic'98, LNCS 1584, 1999.

[16] D. Leivant, *Discrete Polymorphism*, ACM conference on Lisp and Functional Programming (1990) 288-297.

[17] D. Leivant, *Functions over free algebras definable in the simple typed lambda calculus*, Theoretical Computer Science 121 (1993) 309-321.

[18] J. Reynolds, *Toward a theory of type structures*, in J. Loeckx (ed.), Conference on Programming, LNCS 19 (1974) 408-425.

[19] R. Statman, *The typed λ-calculus is not elementary recursive*, Theoretical Computer Science 9 (1979) 73-81.

[20] W.W. Tait, *Intensional interpretation of functionals of finite type I*, J. Symbolic Logic 32 (1967) 198-212.

[21] M. Zaionc, *λ-definability on free algebras*, Annals of Pure and Applied Logic 51 (1991) 279-300.