Remarks on isomorphisms of simple inductive types

David Chemouil, Sergei Soloviev

▶ To cite this version:

David Chemouil, Sergei Soloviev. Remarks on isomorphisms of simple inductive types. Mathematics, Logic and Computation (Satellite Event of ICALP 2003, in honor of N. G. de Bruijn's 85th birthday), Jun 2003, Eindhoven, Netherlands. 85 (7), pp.106-124, 2003, <10.1016/S1571-0661(04)80760-6>. <hal-00783654>

HAL Id: hal-00783654

https://hal.archives-ouvertes.fr/hal-00783654

Submitted on 18 May 2015

Remarks on Isomorphisms of Simple Inductive Types

David Chemouil, Sergei Soloviev [1,2]

IRIT, Université Paul Sabatier
118, route de Narbonne
31062 Toulouse, France

Abstract

We study isomorphisms of types in the system of simply-typed λ-calculus with inductive types and recursion operators. It is shown that in some cases (multiproducts, copies of types), it is possible to add new reductions in such a way that strong normalisation and confluence of the calculus are preserved, and the isomorphisms may be regarded as intensional w.r.t. a stronger equality relation.

1 Introduction

1.1 Presentation

This work is part of a larger project where we are exploring the possibilities of extensions preserving strong normalisation and confluence of standard reduction systems by new reductions of the form $f' \ (f \ t) \longrightarrow t$ where f' is in some sense an inverse of f.

The way this notion of invertibility may be understood is one of the questions we are investigating. A possibility would be to take the invertibility w.r.t extensional equality of functions between inductive types.

Here, we shall consider the simply-typed λ-calculus, equipped with inductive types ($i.e$ recursive types satisfying a condition of strict positivity) and structural recursion schemes on these types.

In this short paper, we will focus on two particular cases where the usefulness of this extension seems obvious. Namely, we shall study some isomorphisms of products (defined as inductive types) and the notion of *copy* of a type

[1] E-mails: `chemouil@irit.fr`, `soloviev@irit.fr`

[2] Work partly funded by Project ISOT (STIC 21) from Department STIC of CNRS.

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

1.2 Isomorphisms of Types

Let us first recall a few facts and definitions about isomorphisms of types.

Definition 1.1 Consider a typed λ-calculus, equipped with an equivalence relation $\sim$ on terms, a term $\mathrm{id}_A : A \to A$ for any type A and a composition operator $\circ$ (with suitable typing) verifying the following conditions, for any function $f : A \to B$:

$$f \circ \mathrm{id}_A \sim f \qquad\qquad \mathrm{id}_B \circ f \sim f$$

Then, two types A and B are said to be *isomorphic* (written $A \cong B$) if there exist two λ-terms $f : A \to B$ and $g : B \to A$ such that

$$f \circ g \sim \mathrm{id}_B \qquad\qquad g \circ f \sim \mathrm{id}_A$$

In this case, g is often written f^{-1} and called the *inverse* of f.

Until now, isomorphisms of types have mostly been studied in various first- or second-order λ-calculi, where $\sim$ is usually generated by $\beta\eta$-conversion [3], $\mathrm{id}_A \mathrel{\widehat{=}} \lambda x : A \cdot x$ and $\circ \mathrel{\widehat{=}} \lambda g : B \to C \cdot \lambda f : A \to B \cdot \lambda x : A \cdot g\ (f\ x)$ (for any types A, B, and C). As an example, we have the following result:

Proposition 1.2 ([21]; [9,11]) *All isomorphisms holding in $\lambda^1\beta\eta_{\to,\times,1}$, the first-order simply-typed λ-calculus with binary products and unit type (or, equivalently, in cartesian closed categories), are obtainable by finite compositions of the following "base" of seven isomorphisms:*

$$A \times B \cong B \times A \qquad\qquad A \times (B \times C) \cong (A \times B) \times C$$

$$(A \times B) \to C \cong A \to (B \to C) \qquad A \to (B \times C) \cong (A \to B) \times (A \to C)$$

$$A \times 1 \cong A \qquad\qquad A \to 1 \cong 1 \qquad\qquad 1 \to A \cong A$$

1.3 Isomorphisms of Inductive Types

Now, it is our view that, as long as inductive types are concerned, *intensional isomorphisms*, in ordinary sense, lack expressivity. To view this problem in a larger context, one needs a notion of *extensionality*.

Definition 1.3 Two types A and B are *extensionally isomorphic* (written $A \approxeq B$) if there exists two λ-terms $f : A \to B$ and $g : B \to A$ such that

$$\forall\, x : A \cdot g\ (f\ x) \sim x \quad \text{and} \quad \forall\, y : B \cdot f\ (g\ y) \sim y\ .$$

(Note that $\cong$ and $\approxeq$ are both equivalence relations.)

[3] It was shown in [10] that with β-conversion solely, the only invertible term is the identity.

Obviously, we have $A \cong B \Rightarrow A \approxeq B$, but the converse is usually not true. One way to achieve this kind of isomorphisms would be to add extensional reduction rules to the calculi, such as η rules, surjective pairing, etc. However, many calculi don't come equipped with extensional reduction rules, for various reasons (decidability, confluence, etc); though some positive results do exist, e.g [17,14,16]. Hence, in this paper, we will mainly be interested with $\beta\iota$-reduction only (where ι-reduction is the rule associated to structural recursion over inductive types).

Of course, extensional isomorphisms are *provable* by induction, but they are not *computable*, *i.e*, one doesn't have (for example)

$$\lambda x : A \cdot f^{-1}\ (f\ x) \ \longrightarrow_{\beta\iota}\ \lambda x : A \cdot x.$$

Without appealing to full extensionality, we think that, if f and f^{-1} are mutually invertible extensional isomorphisms, it is worth considering the addition of new reduction rules (call them σ-reductions, following [6]) as follows:

$$f\ (f^{-1}\ x) \ \longrightarrow_{\sigma}\ x \quad \text{and} \quad f^{-1}\ (f\ x) \ \longrightarrow_{\sigma}\ x.$$

1.4 Outline of the paper

In Sect. 2, we quickly give essential definitions of a simply-typed λ-calculus with inductive types.

Then, in Sect. 3, we quickly present a small lemma ("Deferment Lemma") that is of interest in the next section.

In Sect. 4, we illustrate the addition of rewrite rules on n-ary products. We show that, for products, strong normalisation and confluence are preserved for a rewrite rule corresponding to commutativity, while it is not the case for associativity, unless we also add surjective pairing.

Finally, in Sect. 5, we study the notion of *isomorphic copy* of a type, and how a rewrite rule corresponding to it may or not be added to the calculus.

2 Simply-Typed λ-Calculus with Inductive Types

We define the simply-typed λ-calculus with inductive types, which may be seen as an extension of Gödel's system $\mathcal{T}$. Some references on λ-calculus and inductive types may be found in [4,20,5,23,19,8]. Furthermore, most of our notations and results concerning rewrite systems are taken from [1]. For a given reduction $\longrightarrow_R$, we write $\longrightarrow_R^+$ for its transitive closure, and $\longrightarrow_R^*$ for its reflexive-transitive closure.

2.1 Types

Throughout this paper, we consider an infinite set $\mathcal{S} = \{\alpha, \beta, \ldots\}$ of *type variables.* We also consider an infinite set of variables $\mathcal{V}$ (with $\mathcal{V} \cap \mathcal{S} = \varnothing$), and an infinite set $\mathcal{C}$ of *inductive-type constructors* (or *introduction operators*), with $\mathcal{C} \cap \mathcal{S} = \mathcal{C} \cap \mathcal{V} = \varnothing$.

Moreover, as usual in this sort of presentation, we consider all terms and types up to α-conversion, *i.e* the names of bound variables are irrelevant.

Note 1 *In the following, the sign $\equiv$ will denote syntactic equality, and definitions will be introduced in the calculus with the sign $\widehat{=}$. Furthermore, we will use the common notation* let $x = e_1$ in e_2 *for* $e_2[e_1/x]$.

Definition 2.1 The set of *pre-types* is generated by the following grammar rules:

$$\begin{array}{rl} \mathrm{Ty} ::= & \alpha \quad | \quad (\mathrm{Ty} \to \mathrm{Ty}) \quad | \quad \mathrm{Ind}(\alpha)[\,\mathrm{CS}\,] \\ \mathrm{CS} ::= & \mathrm{CL} \quad | \quad \varepsilon \\ \mathrm{CL} ::= & c : \mathrm{Ty} \quad | \quad c : \mathrm{Ty}\,;\,\mathrm{CL} \end{array}$$

with $c \in \mathcal{C}$ (as usual, ε denotes the empty word). Of course, we require that any constructor belong to only one inductive type.

Note 2 *We consider that $\to$ is right associative, hence $\tau_1 \to (\tau_2 \to \tau_3)$ will be subsequently written $\tau_1 \to \tau_2 \to \tau_3$.*

An inductive type with n constructors $c_1, \ldots, c_n$ in $\mathcal{C}$, each of arity k_i (with $1 \leqslant i \leqslant n$), is then of the form

$$\mathrm{Ind}(\alpha)[\,c_1 : \sigma_1^1 \to \ldots \to \sigma_1^{k_1} \to \alpha\,;\,\ldots\,;\,c_n : \sigma_n^1 \to \ldots \to \sigma_n^{k_n} \to \alpha\,],$$

where the part between brackets is bound by $\mathrm{Ind}(\alpha)$. Moreover, every $\sigma_i \equiv \sigma_i^1 \to \ldots \to \sigma_i^{k_i} \to \alpha$ must verify certain conditions, as explained below.

Definition 2.2 A *strictly positive operator* τ over a type variable α (written τ spos α) is inductively defined by the following rules:

$$\frac{}{\alpha \text{ spos } \alpha} \qquad\qquad \frac{\alpha \notin \mathrm{FV}(\tau_1) \qquad \tau_2 \text{ spos } \alpha}{\tau_1 \to \tau_2 \text{ spos } \alpha}$$

Definition 2.3 An *(inductive) schema* τ over a type variable α (written τ sch α) is inductively defined by the following rules:

$$\frac{}{\alpha \text{ sch } \alpha} \qquad \frac{\alpha \notin \mathrm{FV}(\tau_1) \qquad \tau_2 \text{ sch } \alpha}{\tau_1 \to \tau_2 \text{ sch } \alpha} \qquad \frac{\tau_1 \text{ spos } \alpha \qquad \tau_2 \text{ sch } \alpha}{\tau_1 \to \tau_2 \text{ sch } \alpha}$$

Intuitively, a schema σ is of the form $\sigma^1 \to \ldots \to \sigma^k \to \alpha$, where every σ^j is itself:

- either a type not containing α (we call this σ^j a *non-recursive operator*);
- or a type of the form $\sigma^j \equiv \nu_1 \to \ldots \to \nu_m \to \alpha$ (we call this σ^j a *strictly positive operator*), where α does not appear in any ν_ℓ.

Note 3 *Given a schema $\sigma \equiv \sigma^1 \to \ldots \to \sigma^k \to \alpha$, we will denote by* $\mathrm{SP}_\alpha(\sigma)$ *the set of indices j (with $1 \leqslant j \leqslant k$) such that σ^j is a strictly positive operator over α,* i.e $\mathrm{SP}_\alpha(\sigma) = \{j \mid 1 \leqslant j \leqslant k \wedge \sigma^j \text{ spos } \alpha\}$. *This set will be useful because it corresponds to arguments (of a given constructor) on which a recursive call may be carried out.*

Definition 2.4 A *type* τ (written $\tau : \star$) is inductively defined by the following rules:

$$\frac{\alpha \in \mathcal{S}}{\alpha : \star} \qquad \frac{\tau_1 : \star \qquad \tau_2 : \star}{\tau_1 \to \tau_2 : \star} \qquad \frac{c_i \in \mathcal{C} \qquad \sigma_i : \star \qquad \sigma_i \text{ sch } \alpha \qquad (1 \leqslant i \leqslant n)}{\mathrm{Ind}(\alpha)[\, c_1 : \sigma_1 \,; \ldots ; \, c_n : \sigma_n \,] : \star}$$

Example 2.5 With these rules, it is possible to define the types of natural numbers, of Brouwer's ordinals and of lists of natural numbers (normally, these inductive types should have different constructor names, we used some common names for the sake of readibility):

$$\begin{aligned} \mathrm{Nat} &\mathrel{\widehat{=}} \mathrm{Ind}(\alpha)[\, 0 : \alpha \mid \mathrm{S} : \alpha \to \alpha \,] \\ \mathrm{Ord} &\mathrel{\widehat{=}} \mathrm{Ind}(\alpha)[\, 0 : \alpha \mid \mathrm{S} : \alpha \to \alpha \mid \mathrm{L} : (\mathrm{Nat} \to \alpha) \to \alpha \,] \\ \mathrm{ListNat} &\mathrel{\widehat{=}} \mathrm{Ind}(\alpha)[\, \mathrm{nil} : \alpha \mid \mathrm{cons} : \mathrm{Nat} \to \alpha \to \alpha \,]. \end{aligned}$$

Note that any inductive type τ generates a *recursor* (or *structural-recursion operator*) $\mathcal{R}_{\tau,\kappa}$ to any type κ. This will be further explained in the next section concerned with terms of the language.

2.2 Terms

We will now define the terms of our calculus.

Definition 2.6 The set of *terms* is generated by the following grammar rule:

$$M \ ::= \ c \ \mid \ x \ \mid \ (\lambda x : \tau \cdot M) \ \mid \ (M \ M) \ \mid \ \mathcal{R}_{\tau,\kappa}\,,$$

where $x \in \mathcal{V}$, $c \in \mathcal{C}$ and τ and κ are types.

Note 4 *Application is left-associative, hence $(\ldots(M_1 \ M_2) \ \ldots) \ M_n)$ can be written $M_1 \ldots M_n$. In the same way, abstraction is right-associative, hence $(\lambda x_1 : \tau_1 \cdot (\lambda x_2 : \tau_2 \cdot M))$ can be written $\lambda x_1 : \tau_1 \cdot \lambda x_2 : \tau_2 \cdot M$*

We now define a syntactic operation that will be useful to assert typing rules for terms.

Definition 2.7 Let τ be an inductive type, $\sigma \equiv \sigma^1 \to \ldots \to \sigma^k \to \alpha$ a schema over α in τ, and κ a type. Let $\{j_p\}_{p=1,\ell} = \mathrm{SP}_\alpha(\sigma)$. Then, we define

$$\Upsilon_\tau(\sigma, \kappa) \equiv \sigma^1[\tau/\alpha] \to \ldots \to \sigma^k[\tau/\alpha] \to \sigma^{j_1}[\kappa/\alpha] \to \ldots \sigma^{j_\ell}[\kappa/\alpha] \to \kappa.$$

Definition 2.8 We now present the typing rules for the calculus:

$$\frac{}{\Gamma, x : \alpha \vdash x : \alpha}\ (\textsc{ax}) \qquad \frac{\tau \equiv \mathrm{Ind}(\alpha)[\,\ldots; c : \sigma\,; \ldots] \qquad \tau : \star}{\Gamma \vdash c : \sigma[\tau/\alpha]}\ (\textsc{constr})$$

$$\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{(\lambda x : \tau_1 \cdot M) : \tau_1 \to \tau_2}\ (\lambda) \qquad \frac{\Gamma \vdash M : \tau_1 \to \tau_2 \qquad \Gamma \vdash N : \tau_1}{\Gamma \vdash (M\ N) : \tau_2}\ (\textsc{app})$$

$$\frac{\begin{array}{c}\tau \equiv \mathrm{Ind}(\alpha)[\,c_1 : \sigma_1\,; \ldots; c_n : \sigma_n\,] \\ \Gamma \vdash M_i : \Upsilon_\tau(\sigma_i, \kappa) \qquad (1 \leqslant i \leqslant n)\end{array}}{\Gamma \vdash (\mathcal{R}_{\tau,\kappa}\ M_1\ \ldots\ M_n) : \tau \to \kappa}\ (\textsc{elim})$$

2.3 Reduction

Definition 2.9 We define the usual *β-reduction* rule as follows:

$$(\lambda x : \tau \cdot M)\ N \ \longrightarrow_\beta\ M[N/x]\ .$$

Now, we define the ι-reduction. However, to do so, we first need to make a technical definition which will be helpful.

Definition 2.10 Let $\nu \equiv \nu_1 \to \ldots \to \nu_m \to \alpha$ be a strictly positive operator over α. Then, we define

$$\Xi(R, N, \nu) \equiv \lambda z_1 : \nu_1 \cdot \ldots \cdot \lambda z_m : \nu_m \cdot R\ (N\ z_1\ \ldots\ z_m)\ .$$

Of course, in the special case where $m = 0$, we have $\Xi(R, N, \nu) \equiv R\ N$.

Definition 2.11 Now, let $\sigma \equiv \sigma^1 \to \ldots \to \sigma^k \to \alpha$ be a schema over α, and let $\{j_p\}_{p=1,\ell} = \mathrm{SP}_\alpha(\sigma)$. Then, we define *ι-reduction* by

$$\mathcal{R}_{\tau,\kappa}\ M_1\ \ldots\ M_n\ (c_i\ N_1\ \ldots\ N_{k_i}) \longrightarrow_\iota\ M_i\ N_1, \ldots\ N_{k_i}\ N'_{j_1}\ \ldots\ N'_{j_\ell}\,,$$

where $N'_{j_p} \equiv \Xi(\mathcal{R}_{\tau,\kappa}\ M_1\ \ldots\ M_n, N_{j_p}, \sigma_{j_p})$, for all $1 \leqslant p \leqslant \ell$.

Examples of rules for some basic inductive types are given in Figure 1 on the following page.

Proposition 2.12 *For the simply-typed λ-calculus with inductive types, βι-reduction is strongly normalising and confluent.*

See for example [8].

3 A Deferment Lemma

There are many lemmas concerning with strong normalisability of a relation $\longrightarrow_{RS}$ when $\longrightarrow_R$ and $\longrightarrow_S$ are strongly normalising. Though the lemma we

$$\begin{array}{lll}
\mathcal{R}_{\mathrm{Nat},\kappa}\ a\ f\ 0 & \longrightarrow_{\iota} & a \\
\mathcal{R}_{\mathrm{Nat},\kappa}\ a\ f\ (\mathrm{S}\ p) & \longrightarrow_{\iota} & f\ p\ (\mathcal{R}_{\mathrm{Nat},\kappa}\ a\ f\ p) \\
 & & \\
\mathcal{R}_{\mathrm{Ord},\kappa}\ a\ f\ g\ 0 & \longrightarrow_{\iota} & a \\
\mathcal{R}_{\mathrm{Ord},\kappa}\ a\ f\ g\ (\mathrm{S}\ p) & \longrightarrow_{\iota} & f\ p\ (\mathcal{R}_{\mathrm{Ord},\kappa}\ a\ f\ g\ p) \\
\mathcal{R}_{\mathrm{Ord},\kappa}\ a\ f\ g\ (\mathrm{L}\ k) & \longrightarrow_{\iota} & g\ k\ (\lambda z : \mathrm{Nat} \cdot (\mathcal{R}_{\mathrm{Ord},\kappa}\ a\ f\ g\ (k\ z))) \\
 & & \\
\mathcal{R}_{\mathrm{ListNat},\kappa}\ a\ f\ \mathrm{nil} & \longrightarrow_{\iota} & a \\
\mathcal{R}_{\mathrm{ListNat},\kappa}\ a\ f\ (\mathrm{cons}\ h\ t) & \longrightarrow_{\iota} & f\ h\ t\ (\mathcal{R}_{\mathrm{ListNat},\kappa}\ a\ f\ t)
\end{array}$$

Fig. 1. Recursion rules for some basic inductive types

consider below is close to many results in the folklore, we could not find its exact formulation in the literature.

Note also that this lemma is not equivalent to the so-called Postponement Lemma for η-contractions in pure λ-calculus, see e.g [3] p. 386.

Definition 3.1 Let $\longrightarrow_R$ and $\longrightarrow_S$ be two reductions. Then, $\longrightarrow_S$ is *deferable w.r.t* $\longrightarrow_R$ if, for all terms t and u such that $t \longrightarrow_S \longrightarrow_R u$, there is a derivation $t \longrightarrow_R \longrightarrow_{RS}^* u$.

$$\begin{tikzcd}
 & t \arrow[dl, "S"'] \arrow[dr, dotted, "R"] & \\
{} \arrow[dr, "R"'] & & {} \arrow[dl, dotted, "RS"', "*"] \\
 & u &
\end{tikzcd}$$

Lemma 3.2 (Deferment Lemma) *Let* $\longrightarrow_R$ *and* $\longrightarrow_S$ *be two strongly normalising relations. Then, if* $\longrightarrow_S$ *is deferable w.r.t* $\longrightarrow_R$*,* $\longrightarrow_{RS}$ *is strongly normalising.*

Proof. Let $\longrightarrow_R$ and $\longrightarrow_S$ be two strongly normalising relations, such that $\longrightarrow_S$ is deferable w.r.t $\longrightarrow_R$. Let us suppose that $\longrightarrow_{RS}$ is not strongly normalising, and show that it leads to a contradiction.

If $\longrightarrow_{RS}$ is not strongly normalising, then $\longrightarrow_{RS}^*$ consists of an infinite alternation of $\longrightarrow_R^*$ and $\longrightarrow_S^*$. Then, one can inductively "lift" $\longrightarrow_R$-reductions by deferring every $\longrightarrow_S$-reduction followed by an $\longrightarrow_R$-reduction, thus building an infinite derivation of $\longrightarrow_R$ steps. This contradicts the fact that $\longrightarrow_R$ is strongly normalising. □

In fact, we can prove a slightly more powerful lemma whose premises occur however less in practice.

Definition 3.3 Let $\longrightarrow_R$ and $\longrightarrow_S$ be two reductions. Then, $\longrightarrow_S$ is *0-deferable w.r.t* $\longrightarrow_R$ if, for all terms t and u such that $t \longrightarrow_S u$, there is a derivation $t \longrightarrow_R \longrightarrow^*_{RS} u$.

$$\begin{tikzcd}
t \arrow[dd, "S"'] \arrow[dr, dotted, "R"] & \\
& {} \arrow[dl, dotted, "RS", "*"'] \\
u &
\end{tikzcd}$$

Lemma 3.4 (0-Deferment Lemma) *Let $\longrightarrow_R$ and $\longrightarrow_S$ be two strongly normalising relations. Then, if $\longrightarrow_S$ is 0-deferable w.r.t $\longrightarrow_R$, $\longrightarrow_{RS}$ is strongly normalising.*

Proof. Immediate, because 0-deferment implies deferment. □

Remark 3.5 Since the submission of this paper, we found some references about what we call Deferment Lemma (cf. [2,15] and most notably [13]). While we shall keep calling this property "deferment" in the current paper, we intend to use the preferable term "adjournement" afterwards, following Delia Kesner (private communication).

4 Multiproducts

Let us define a schema of inductive types representing n-ary products:

$$\Pi_n\ A_1 \ldots A_n \mathrel{\widehat{=}} \mathrm{Ind}(\alpha)[\ \langle\cdot\rangle_n : A_1 \to \ldots \to An \to \alpha\ ]\ ,$$

with recursion operator $(\!|\cdot|\!)_n$ defined by

$$\begin{aligned}
&(\!|\cdot|\!)_n : (A_1 \to \ldots \to A_n \to B) \to (\Pi_n\ A_1 \ldots A_n \to B)\\
&(\!|f|\!)_n\ \langle a_1 \ldots a_n\rangle_n \longrightarrow_\iota f\ a_1 \ldots a_n\ .
\end{aligned}$$

The projections p^n_k are defined as $(\!|\lambda x_1 : A_1 \cdot \ldots \cdot \lambda x_n : A_n \cdot x_k|\!)_n$.

Remark 4.1 One may note that the product of morphisms $f_i : C \to A_i$ (with $1 \leqslant i \leqslant n$) is definable, without the elimination operator, by

$$\mathrm{prod}_n\ f_1 \ldots f_n \mathrel{\widehat{=}} \lambda z : C \cdot \langle f_1\ z, \ldots, f_n\ z\rangle_n\ .$$

However, many familiar properties of product and projections do not hold intensionally. For example, we have $\langle p^2_1\ x, p^2_2\ x\rangle_2 \neq_{\beta\iota} x$ for $x : \Pi_2\ A\ B$. In fact, this property, usually known as *surjective pairing*, stipulates that the product is unique.

4.1 Commutativity of Products

Now, let ϱ be a permutation of $\{1, \ldots, n\}$. The permutation of $\Pi_n\ A_1 \ldots A_n$ induced by ϱ is denoted $\overline{\varrho}$, and defined as $(\!|\lambda x_1 : A_1 \cdot \ldots \cdot \lambda x_n : A_n \cdot \langle x_{\varrho(1)}, \ldots, x_{\varrho(n)}\rangle_n|\!)_n$.

Proposition 4.2 *For any term $t : \Pi_n\ A_1 \ldots A_n$ and permutations ϱ and ω defined on $\{1, \ldots, n\}$, the equality $\overline{\varrho \circ \omega}\ t =_{\beta\iota} \overline{\varrho}\ (\overline{\omega}\ t)$ is provable.*

Still, while we can prove this proposition by induction, it is important to note that the equality is not computable for an arbitrary t, but just when $t \equiv \langle t_1, \ldots, t_n \rangle_n$ for some n (cf. Sect. 1.3 on page 3). Note also that for mutually inverse permutations ϱ and ϱ^{-1}, $\overline{\varrho}$ and $\overline{\varrho^{-1}}$ are mutually inverse extensional isomorphisms.

Now, for given mutually inverse permutations ϱ and ϱ^{-1}, let us add the following rewrite rules to the system of $\beta\iota$-reductions:

$$\overline{\varrho}\ (\overline{\varrho^{-1}}\ x) \longrightarrow_\sigma x \qquad\qquad \overline{\varrho^{-1}}\ (\overline{\varrho}\ x) \longrightarrow_\sigma x\ .$$

(Note that $\overline{\varrho}$ and $\overline{\varrho^{-1}}$ are *concrete*, *i.e* constant, terms of the calculus.)

Remark 4.3 To lighten the notation, let us write π and π' for $\overline{\varrho}$ and $\overline{\varrho^{-1}}$. We will also make use of diagrams, as is usually done for this kind of proof.

Lemma 4.4 *σ-reduction is strongly normalising.*

Proof. Take the length of terms as an ordering. □

Theorem 4.5 *$\beta\iota\sigma$-reduction is strongly normalising.*

Proof. We show that σ-reduction is deferable w.r.t β-reduction (case i) and w.r.t ι-reduction (case ii).

(i) For β-reduction. The crucial case is when the σ-redex occurs inside a β-redex.

i.1. As a first possibility, we may have $t \equiv t'[(\lambda x : A \cdot p[\pi\ (\pi'\ s)])\ q]$. Note that π and π' do not contain variables.

$$\begin{tikzcd}
& t \equiv t'[(\lambda x : A \cdot p[\pi\ (\pi'\ s)])\ q] \arrow[dl, "\sigma"'] \arrow[dr, dotted, "\beta"] & \\
t'[(\lambda x : A \cdot p[s])\ q] \arrow[dr, "\beta"'] & & t'[(p[\pi\ (\pi'\ s)])[q/x]] \arrow[dl, dotted, "\sigma"] \\
& t'[(p[s])[q/x]] &
\end{tikzcd}$$

i.2. We may also have $t \equiv t'[(\lambda x : A \cdot p)\ (q[\pi\ (\pi'\ s)])]$, in which case the term p may contain many (or zero) occurrences of x, which requires to carry as many σ-reductions.

$$\begin{tikzcd}
& t \equiv t'[(\lambda x : A \cdot p)\ (q[\pi\ (\pi'\ s)])] \arrow[dl, "\sigma"'] \arrow[dr, dotted, "\beta"] & \\
t'[(\lambda x : A \cdot p)\ (q[s])] \arrow[dr, "\beta"'] & & t'[p[q[\pi\ (\pi'\ s)] \,/\, x]] \arrow[dl, dotted, "\sigma", "*"' near end] \\
& t'[p[q[s]/x]] &
\end{tikzcd}$$

(ii) For ι-reduction.

ii.1. The crucial case occurs when a ι-redex may interact with π and π', hence we must have $t \equiv t'[\pi\ (\pi'\ \langle s_1, \ldots, s_n\rangle_n)]$. But then, it is immediate to see that $t \longrightarrow_\sigma t'[\langle s_1, \ldots, s_n\rangle_n]$ can also be performed by the derivation: $t'[\pi\ (\pi'\ \langle s_1, \ldots, s_n\rangle_n)] \longrightarrow_\iota \longrightarrow^+_{\beta\iota} t'[\langle s_1, \ldots, s_n\rangle_n]$. This is a trivial case of 0-deferment.

ii.2. In other cases, the ι-redex doesn't interfere with σ-reduction, therefore deferment is obviously possible.

□

Theorem 4.6 *$\beta\iota\sigma$-reduction is confluent.*

Proof. First, as $\beta\iota\sigma$-reduction is strongly normalising, it is enough to show local confluence (by Newman's Lemma), *i.e* for all terms t, w, w' such that $t \longrightarrow_{\beta\iota\sigma} w$ and $t \longrightarrow_{\beta\iota\sigma} w'$, there exists a term u such that $w \longrightarrow^*_{\beta\iota\sigma} u$ and $w' \longrightarrow^*_{\beta\iota\sigma} u$.

By Lemma 2.12, $\beta\iota$-reduction is confluent. For σ-reductions alone, by Newman's Lemma it is enough to show local confluence. The critical pairs induced by σ-reduction are joinable; hence by the Critical Pair Theorem, σ-reduction is locally confluent. Therefore, for $\beta\iota\sigma$-reductions there are only two extra cases to be considered depending on whether one carries a β- or ι-reduction (combined with σ-) as a first step.

(i) If it is a ι-reduction, then $t \equiv t'[\pi\ (\pi'\ s)]$, and there are 4 possible cases: the ι-redex is in s, the ι-redex has no intersection with $\pi\ (\pi'\ s)$, the ι-redex contains $\pi\ (\pi'\ s)$, or the ι-redex is in $\pi\ (\pi'\ s)$ and intersects with π or π'.

i.1. We have $t \equiv t'[\pi\ (\pi'\ (s'[r]))]$, r being a ι-redex. Then, if t ι-reduces to $t'[\pi\ (\pi'\ s'[r'])]$ and σ-reduces to $t'[s'[r]]$, it is possible to "close" the fork by $t'[\pi\ (\pi'\ s'[r'])] \longrightarrow_\sigma t'[s'[r']]$ and $t'[\pi\ (\pi'\ s'[r'])] \longrightarrow_\iota t'[s'[r']]$.

i.2. Once more, the order is indifferent.

i.3. One has $t' \equiv t''[r[\pi\ (\pi'\ s)]]$. The upper-left ι-reduction cannot affect $\pi\ (\pi'\ s)$ since this part doesn't begin with an introduction operator. (In general, the lower-left reduction would possibly be $\longrightarrow^*_\sigma$ since the number of σ-redexes may change when ι-reduction is applied, but it is not the case for products.)

$$\begin{tikzcd}
& t' \equiv t''[r[\pi\ (\pi'\ s)]] \arrow[dl, "\iota"'] \arrow[dr, "\sigma"] & \\
t''[r'[\pi\ (\pi'\ s)]] \arrow[dr, dotted, "\sigma"'] & & t''[r[s]] \arrow[dl, dotted, "\iota"] \\
& t''[r'[s]] &
\end{tikzcd}$$

i.4. In fact, the ι-redex should coincide with $(\pi'\ s)$, since $(\pi'\ s)$ doesn't begin with an introduction operator, so it cannot be $\pi\ (\pi'\ s)$ (here, we use the concrete definition of π and π'). Thus, s must be of the form $\langle s_1, \ldots, s_n\rangle_n$. But, for all elements of this form, we have $\pi\ (\pi'\ \langle s_1, \ldots, s_n\rangle_n) \longrightarrow_\iota \longrightarrow^+_{\beta\iota} \langle s_1, \ldots, s_n\rangle_n$, hence local confluence holds trivially in this case.

(ii) For β-reduction, cases ii.1 and ii.2 are similar to cases i.1 and i.2, thus treated as above.

ii.3 If $t \equiv t'[(\lambda x : A \cdot p[\pi\ (\pi'\ s)])\ q]$, and $t \longrightarrow_\beta t'[(p[\pi\ (\pi'\ s)])[q/x]]$ and $t \longrightarrow_\sigma t'[(\lambda x : A \cdot p[s])\ q]$, closing the "fork" is straightforward by observing that both terms σ- and β-reduce respectively in one step to $t'[(p[s])[q/x]]$. (Note that this situation appears because π and π' are closed terms.)

ii.4 In the last case, where $t \equiv t'[(\lambda x : A \cdot p)\ (q[\pi\ (\pi'\ s)])]$, the number of occurrences of x in p may influence the number of σ-reductions to perform to close the diagram. Thus, if $t \longrightarrow_\beta t'[p[q[\pi\ (\pi'\ s)]\,/x]]$ and $t \longrightarrow_\sigma t'[(\lambda x : A \cdot p)\ (q[s])]$, we may need a sequence of reductions $t'[p[q[\pi\ (\pi'\ s)]\,/x]] \longrightarrow^*_\sigma t'[p[q[s]/x]]$ while a one-step β-reduction only would be necessary on the other term: $t'[(\lambda x : A \cdot p)\ (q[s])] \longrightarrow_\beta t'[p[q[s]/x]]$.

□

4.2 Associativity of Products

As just seen, products enjoy the commutativity property. However, the associativity does not hold in general, *i.e*, it is not the case that, for example, $\Pi_2\ (\Pi_2\ A\ B)\ C \approxeq \Pi_2\ A\ (\Pi_2\ B\ C)$. This is so because there is an occurence of $\Pi_2\ A\ B$ (or $\Pi_2\ B\ C$) *inside* another Π_2. Thus, the "isomorphisms" g and g' would be defined in the following way:

$$\begin{aligned} g &: \Pi_2\ (\Pi_2\ A\ B)\ C \to \Pi_2\ A\ (\Pi_2\ B\ C) \\ &\mathrel{\widehat{=}} \llparenthesis \lambda p : \Pi_2\ A\ B \cdot \lambda c : C \cdot \langle p^2_1\ p, \langle p^2_2\ p, c\rangle_2\rangle_2 \rrparenthesis \end{aligned}$$

and

$$\begin{aligned} g' &: \Pi_2\ A\ (\Pi_2\ B\ C) \to \Pi_2\ (\Pi_2\ A\ B)\ C \\ &\mathrel{\widehat{=}} (\!| \lambda a : A \cdot \lambda q : \Pi_2\ B\ C \cdot \langle \langle a, p_1^2\ q\rangle_2, p_2^2\ q\rangle_2 |\!) \enspace . \end{aligned}$$

Then, for a term $\langle p, c\rangle_2$, with $p : \Pi_2\ A\ B$ and $c : C$, one has:

$$\begin{aligned} g'\ (g\ \langle p, c\rangle_2) &\longrightarrow_\iota \longrightarrow_\beta g'\ \langle p_1^2\ p, \langle p_2^2\ p, c\rangle_2\rangle_2 \\ &\longrightarrow_\iota \longrightarrow_\beta \langle \langle p_1^2\ p, p_2^2\ p\rangle_2, c\rangle_2 \neq_{\beta\iota} \langle p, c\rangle_2 \end{aligned}$$

because of the lack of surjective pairing. It is interesting to note that, even with extensionality on canonical elements, the isomorphism establishing associativity of binary product does not hold.

4.3 Retractions

Now, let us consider some correspondances between n-products for different n, for example $\Pi_3\ A\ B\ C$ and $\Pi_2\ (\Pi_2\ A\ B)\ C$. Define

$$\begin{aligned} f &: \Pi_2\ (\Pi_2\ A\ B)\ C \to \Pi_3\ A\ B\ C \\ &\mathrel{\widehat{=}} (\!| \lambda y : \Pi_2\ A\ B \cdot \lambda z : C \cdot \langle p_1^2\ y, p_2^2\ y, z\rangle_3 |\!)_2 \end{aligned}$$

and

$$\begin{aligned} f' &: \Pi_3\ A\ B\ C \to \Pi_2\ (\Pi_2\ A\ B)\ C \\ &\mathrel{\widehat{=}} (\!| \lambda x : A \cdot \lambda y : B \cdot \lambda z : C \cdot \langle \langle x, y\rangle_2, z\rangle_2 |\!)_3 \enspace . \end{aligned}$$

For $\langle t, u, v\rangle_3 : \Pi_3\ A\ B\ C$, we have:

$$f\ (f'\ \langle t, u, v\rangle_3) \longrightarrow_\iota \longrightarrow_\beta f\ \langle \langle t, u\rangle_2, v\rangle_2 \longrightarrow_\iota \longrightarrow_\beta \langle t, u, v\rangle_3 \enspace .$$

However, for $\langle y, z\rangle_2 : \Pi_2\ (\Pi_2\ A\ B)\ C$, we have:

$$\begin{aligned} f'\ (f\ \langle y, z\rangle_2) &\longrightarrow_\iota \longrightarrow_\beta f'\ \langle p_1^2\ y, p_2^2\ y, z\rangle_3 \\ &\longrightarrow_\iota \longrightarrow_\beta \langle \langle p_1^2\ y, p_2^2\ y\rangle_2, z\rangle_2 \neq_{\beta\iota} \langle y, z\rangle_2 \enspace , \end{aligned}$$

once again because the type $\Pi_2\ A\ B$ doesn't enjoy surjective pairing. This means that even in an extensional sense (on canonical elements), f is only a *retraction*, and not an isomorphism. Of course, the same situation will appear if we consider the product of n elements expressed with Π_n, and using a superposition of Π_k for $k < n$. While we will not consider deeply the case of retractions in this paper, we think they deserve attention for further studies: this example suggests that Π_3 might be considered as the "canonical" representation of triples, for being the retract of all representations of triples. One may note that this observation demonstrates the usefulness of adding new reductions gradually. The correspondence between products of different arity described above would remain hidden if surjective pairing was already present.

4.4 Surjective Pairing

Let us add the rule $\langle p^2_1\ x, p^2_2\ x\rangle_2 \longrightarrow_{SP} x$ (if x is of product type) to the system with $\beta\iota$-reductions. We will now show that the Deferment Lemma may also be applied to prove strong normalisation of a system of $\beta\iota$SP-reductions.

Consider a *SP*-reduction followed by some β- or ι-reduction.

$$t[\langle p^2_1\ s, p^2_2\ s\rangle_2] \longrightarrow_{SP} t[s] \longrightarrow_{\iota} t^*[s^*]\ \ .$$

If s does not have the form $\langle s_1, s_2\rangle_2$ or it does but the reduction does not use this occurrence of $\langle\cdot,\cdot\rangle_2$ then deferment is obviously possible.

Suppose the reduction that follows *SP* is ι, then t should be a term of the form $t[\langle p^2_1\ s, p^2_2\ s\rangle_2] \equiv t'[(\!| f |\!)_2\ \langle p^2_1\ s, p^2_2\ s\rangle_2]$ where $s : \Pi_2\ A\ B$, $s_1 : A$, $s_2 : B$, $f : A \to B \to C$ and we have

$$t'[(\!| f |\!)_2\ \langle p^2_1\ s, p^2_2\ s\rangle_2] \longrightarrow_{SP} t'[(\!| f |\!)_2\ \langle s_1, s_2\rangle_2] \longrightarrow_{\iota} t'[f\ s_1\ s_2]\ \ .$$

This can be replaced by

$$\begin{aligned} t'[(\!| f |\!)_2\ \langle p^2_1\ s, p^2_2\ s\rangle_2] &\longrightarrow_{\iota} t'[f\ (p^2_1\ s)\ (p^2_2\ s)] \\ &\longrightarrow_{\iota}\longrightarrow_{\beta} t'[f\ s_1\ (p^2_2\ s)] \longrightarrow_{\iota}\longrightarrow_{\beta} t'[f\ s_1\ s_2] \end{aligned}$$

(a trivial case of deferment). It is easy to see that local confluence will hold as well.

5 Isomorphic Copies of (Non-)Algebraic Types

The notion of the copy of a type is a very important one, and occurs quite often in many developments. For example, such operations are frequently used in tree-processing programs such as compilers. In this section, we study how isomorphisms may be used to devise an extended notion of copy, namely the *isomorphic copy* (for want of a better name).

Let us consider two extensionally isomorphic types A and B with isomorphisms $f : A \to B$ and $f^{-1} : B \to A$, and a type

$$C \equiv \mathrm{Ind}(\alpha)[\, c_1 : \sigma^1_1 \to \ldots \to \sigma^{k_1}_1 \to \alpha\,;\, \ldots\,;\, c_n : \sigma^1_n \to \ldots \to \sigma^{k_n}_n \to \alpha\,]\ \ ,$$

possibly containing occurrences of A. An *isomorphic copy* C' of C differs by names of introduction operators, e.g $c'_1, ..., c'_n$, and by the fact that each "atomic" occurrence of A in C is replaced by an occurrence of B in C' (that is to say: A will be replaced by B only if it occurs either as a non-recursive operator, or as the premise —*i.e*, the type of an argument— of a strictly positive operator).

The reader who prefers a less abstract setting may suppose the isomorphisms between A and B belong to the class studied in section 4. It can be also intensional isomorphism, e.g., permutation of premisses of a functional type.

The definitions below also may be modified in such a way that only some selected occurrences of A are considered.

Now, let us define a function icopy : $C \to C'$ which converts canonical objects from one type to the other. Formally, icopy is of the form $\mathcal{R}_{C,C'}\ M_1 \ldots M_n$. For every constructor $c_i : \sigma_i^1 \to \ldots \to \sigma_i^{k_i} \to C$, let $\{j_p\}_{p=1,\ell} = \mathrm{SP}_\alpha(\sigma)$ and let us denote every strictly positive operator $\sigma_i^{j_p}$ by $\nu_{i,j,1} \to \ldots \nu_{i,j,p_{i,j}} \to \alpha$. Then, we have

$$\begin{aligned} M_i \equiv \lambda x_1 : \sigma_i^1[C/\alpha] \cdot \ldots \cdot \lambda x_{k_i} : \sigma_i^{k_i}[C/\alpha] \cdot \\ \lambda w_{j_1} : \sigma_i^{j_1}[C'/\alpha] \cdot \ldots \cdot \lambda w_{j_\ell} : \sigma_i^{j_\ell}[C'/\alpha] \cdot c_i'\ \delta_1 \ldots \delta_{k_i} \end{aligned}$$

where

$$\delta_m \equiv \begin{cases} \text{(a)} \quad \lambda z_1 : \nu'_{i,m,1} \cdot \ldots \cdot \lambda z_p : \nu'_{i,m,p_{i,m}} \cdot w_m\ z_1' \ldots z_p' & \text{if } m \in j_1, \ldots, j_\ell; \\ \text{(b)} \quad f\ x_m & \text{if } \sigma_i^m \equiv A; \\ \text{(c)} \quad x_m & \text{otherwise;} \end{cases}$$

and, for $1 \leqslant r \leqslant p_{i,m}$:

- $\nu'_{i,m,r} \equiv B$ and $z_r' \equiv f^{-1}\ z_r$ if $\nu_r \equiv A$;
- $\nu'_{i,m,r} \equiv \nu_{i,m,r}$ and $z_r' \equiv z_r$ otherwise.

The function icopy^{-1} : $C' \to C$ is defined similarly.

We may now consider the behaviour of icopy and icopy^{-1} w.r.t introduction operators, assuming that the new σ-reductions icopy^{-1} (icopy x) $\longrightarrow_\sigma x$ *and* $f^{-1}\ (f\ x) \longrightarrow_\sigma x$ are added. The main observation is that

$$\text{icopy}^{-1}\ (\text{icopy}\ (c_i\ t_1 \ldots t_{k_i})) \longrightarrow^+_{\beta\iota} c_i\ t_1' \ldots t_{k_i}'$$

where t_j':

- is t_j in case (c);
- is $f^{-1}\ (f\ t_j)$ in case (b);
- and is of the form $\lambda z_1 : \nu_{i,j,1} \cdot \ldots \cdot \lambda z_p : \nu_{i,i,p_{i,j}} \cdot \text{icopy}^{-1}\ (\text{icopy}\ (t_j\ z_1' \ldots z_p'))$ where $z_r' \equiv f^{-1}\ (f\ z_r)$ if $\nu_r \equiv A$, $z_r' \equiv z_r$ otherwise, in case (a).

Now, suppose we have a term of the form $q[\text{icopy}^{-1}\ (\text{icopy}\ (c_i\ t_1 \ldots t_{k_i}))]$.

Then, by one single σ-reduction, we have

$$q[\text{icopy}^{-1}\ (\text{icopy}\ (c_i\ t_1 \ldots t_{k_i}))] \longrightarrow_\sigma q[c_i\ t_1 \ldots t_{k_i}]\ .$$

But we may try to defer this σ-reduction. First, we have

$$q[\text{icopy}^{-1}\ (\text{icopy}\ (c_i\ t_1 \ldots t_{k_i}))] \longrightarrow_{\beta\iota}^{+} q[c_i\ t'_1 \ldots t'_{k_i}]\ .$$

Now, the deferment will depend on which cases the t'_j are in. In case (c), we have $t'_j \equiv t_j$, so no more reduction is to be done to close the diagram. If case (b) happens, some σ-reductions will be needed:

$$q[c_i\ t'_1 \ldots t'_{k_i}] \longrightarrow_\sigma^{+} q[c_i\ t_1 \ldots t_{k_i}]\ .$$

Finally, if case (a) happens, carrying some σ-reductions may lead to an unclosed diagram:

$$q[c_i\ t'_1 \ldots t'_{k_i}] \longrightarrow_\sigma^{+} q[c_i\ t''_1 \ldots t''_{k_i}]\ ,$$

where t''_j may begin by some abstractions. This situation will not happen only in the specific case, similar in result to case (b), where σ_i^j is a strictly positive operator over α of null arity, *i.e* $\sigma_i^j \equiv \alpha$. For example, this is the case for the 'S' constructor of ordinals. In the general case however (*i.e* with σ_i^j being a strictly positive operator over α of non-null arity), the only way to close the diagram seems to add further η-expansions in the following way:

$$q[c_i\ t_1 \ldots t_{k_i}] \longrightarrow_\eta^{+} q[c_i\ t''_1 \ldots t''_{k_i}]\ .$$

As an example, we have, for the 'L' constructor of ordinals the following reduction graph:

$$\begin{tikzcd}
 & q[\text{icopy}^{-1}\ (\text{icopy}\ (L\ k))] \arrow[dl, "\sigma"'] \arrow[dr, "\beta\iota", "*"{near end, swap}] & \\
q[L\ k] \arrow[dr, "\eta"'] & & q[L\ (\lambda z_1 : N \cdot \text{icopy}^{-1}\ (\text{icopy}\ (k\ (f^{-1}\ (f\ z_1)))))] \arrow[dl, "\sigma", "2"{near end, swap}] \\
 & q[L\ (\lambda z_1 : N \cdot k\ z_1)] &
\end{tikzcd}$$

As a conclusion, if we only meet cases (c) and (b), and case (a) with only null-arity strictly positive operators, it is always possible to (0-)defer σ-reductions in the calculus. Thus $\beta\iota\sigma$-reduction is strongly normalising for "algebraic" types. Confluence follows easily, with a similar proof as for Theorem 4.6 on page 10.

As we briefly discussed above, our "strategy" is to add new reductions one by one. Thus, even the result for algebraic types only opens a large field of applications for icopy, generated by isomorphisms of parameters introduced previously.

The difficult case is when “non-algebraic” types occur. Recently we obtained a proof for this case and the system with η-expansion.

Definition 5.1 We define η-expansion as follows:

$$M \longrightarrow_{\eta} \lambda x : A \cdot M\ x \quad \text{if} \quad \begin{cases} M \text{ is of function type } A \to B \\ M \text{ is neither an abstraction nor applied.} \end{cases}$$

In detailed form the proof is too long to be presented here and we shall only give an outline.

The main observation used in this proof is that if the terms $t_1, ..., t_{k_i}$ above are in η-expanded form then

$$q[c_i\ t_1 \ldots t_{k_i}]\ \overset{+}{\underset{\beta}{\longleftarrow}}\ q[c_i\ t_1'' \ldots t_{k_i}''] \ .$$

E.g., the diagram for ’L’ constructor may be closed differently:

$$\begin{tikzcd}
 & q[\text{icopy}^{-1}\ (\text{icopy}\ (L\ k))] \arrow[dl, "\sigma"'] \arrow[dr, "\beta\iota", "*" near end] & \\
q[L\ k] & & q[L\ (\lambda z_1 : N \cdot \text{icopy}^{-1}\ (\text{icopy}\ (k\ (f^{-1}\ (f\ z_1)))))] \arrow[dl, "\sigma"', "2" near end] \\
 & q[L\ (\lambda z_1 : N \cdot k\ z_1)] \arrow[ul, "\beta"] &
\end{tikzcd}$$

Since we consider the system with η-expansions, we need a proof that the system with $\beta\iota$ and η-expansions is strongly normalising and confluent (we currently have a sketch of this proof).

To prove strong normalisation of the system extended not only by η but by σ-reductions related to icopy we assume that there is an infinite reduction sequence including σ reductions.

To use the observation above we need a lemma that shows that this reduction sequence will remain infinite if we insert appropriate η-expansions (to make the terms t in case (a) η-expanded).

After that, using a modification of deferment (to take into account the condition that the terms t are η-expanded) we show that it would be possible to obtain an infinite sequence consisting of $\beta\eta\iota$ only and this contradiction shows that the system with σ is SN.

The proof is completed by verification of confluence.

Acknowledgement

We would like to thank Roberto Di Cosmo for helpful discussions, and Freiric Barral for his help and proofreading.

6 Conclusion

The systems based on intensional equality (e.g., many proof assistants) often puzzle mathematically-oriented users because some familiar functional equalities (such as equalities related to commutativity and associativity of product) are no more viewed as computational and their use may require additional and heavy proof development. The arguments in favor of the equality based only on $\beta\iota$-reduction (or even $\beta\eta\iota$) may look nice from the foundational point of view but, pragmatically speaking, there is no harm if an extension of a reduction system doesn't destroy properties such as strong normalisation and confluence.

In this short paper, we studied two cases that seem of interest: extensions of reduction systems related to products and also to *"isomorphic" copies* of a type.

As for products, using the Deferment Lemma, we were able to prove that adding a rewriting rule corresponding to commutativity of products keeps the calculus strongly normalising and confluent. The same lemma also enabled us to show that adding surjective pairing to the system of $\beta\iota$-reductions does not break normalisation and confluence properties.

Secondly the notion of *isomorphic copy*, is useful for a clean distinction between the multiple uses of the type itself and of its copies. E.g., in proof assistants, the type of Even numbers is often defined as a copy of type Nat together with an appropriate coercion Even → Nat. Combining this coercion with the isomorphism copy defined above, we may obtain representations of classes of numbers modulo 2^n. Furthermore, isomorphic copies of non-algebraic types may require a notion of η-expansion, and hence to show that $\beta\eta\iota\sigma$-reduction is strongly normalising and confluent.

There are several recent works where normalisation in extended reduction systems is considered (e.g., [22] or [7,8]). This makes the perspective seem quite optimistic.

The calculus we considered here (the simply-typed λ-calculus with inductive types) is a compromise between the richness provided by inductive constructions and the relative simplicity of simply-typed systems. In the case of dependent types, one will meet more difficulties because new reductions will influence type-equality as well.

The subject needs more investigation but appropriate methods (e.g., a modification of H. Goguen's Typed Operational Semantics, see [18]) will probably lead to useful results of the same type as presented here.

References

[1] Baader, F. and T. Nipkow, “Term Rewriting and All That,” Cambridge University Press, New York, 1998.

[2] Bachmair, L. and N. Dershowitz, *Commutation, transformation, and termination*, in: J. H. Siekmann, editor, *Proceedings of the Eighth International Conference on Automated Deduction (Oxford, England)*, Lecture Notes in Computer Science **230** (1986), pp. 5–20.

[3] Barendregt, H. P., “The Lambda Calculus - Its Syntax and Semantics,” North-Holland, Amsterdam, 1984.

[4] Barendregt, H. P., *Lambda calculi with types*, in: D. M. Gabbai, S. Abramsky and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, Oxford University Press, Oxford, 1992 .

[5] Barras, B., S. Boutin, C. Cornes, J. Courant, J.-C. Filliatre, E. Gimenez, H. Herbelin, G. Huet, C. Munoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saibi and B. Werner, *The Coq proof assistant reference manual : Version 6.1*, Technical Report RT-0203, Inria (Institut National de Recherche en Informatique et en Automatique), France (1997).

[6] Barthe, G. and O. Pons, *Type isomorphisms and proof reuse in dependent type theory*, in: F. Honsell and M. Miculan, editors, *Proceedings 4th Int. Conf. on Found. of Software Science and Computation Structures, FoSSaCS'01, Genova, Italy, 2–6 Apr. 2001*, Lecture Notes in Computer Science **2030**, Springer-Verlag, Berlin, 2001 pp. 57–71.

[7] Blanqui, F., J.-P. Jouannaud and M. Okada, *The calculus of algebraic constructions*, in: P. Narendran and M. Rusinowitch, editors, *Proceedings of the 10th International Conference on Rewriting Techniques and Applications (RTA-99)* (1999), pp. 301–316.

[8] Blanqui, F., J.-P. Jouannaud and M. Okada, *Inductive-data-type systems*, Theoretical Computer Science **272** (2002), pp. 41–68.

[9] Bruce, K., R. Di Cosmo and G. Longo, *Provable isomorphisms of types*, Technical Report 90-14, LIENS, École Normale Supérieure, Paris (1990).

[10] Dezani-Ciancaglini, M., *Characterization of normal forms possessing inverse in the $\lambda-\beta-\eta$-calculus*, Theoretical Computer Science **2** (1976), pp. 323–337.

[11] Di Cosmo, R., “Isomorphisms of Types: From λ-Calculus to Information Retrieval and Language Design,” Progress in Theoretical Computer Science, Birkhäuser, Boston, MA, 1995.

[12] Di Cosmo, R. and D. Kesner, *Combining algebraic rewriting, extensional lambda calculi, and fixpoints*, Theoretical Computer Science **169** (1996), pp. 201–220.

[13] Doornbos, H. and B. von Karger, *On the union of well-founded relations*, Logic Journal of the IGPL **6** (1998), pp. 195–201.

[14] Dowek, G., G. Huet and B. Werner, *On the definition of the eta-long normal form in type systems of the cube*, in: H. Geuvers, editor, *Informal Proceedings of the Workshop on Types for Proofs and Programs*, Nijmegen, The Netherlands, 1993.

[15] Geser, A., "Relative Termination," Ph.D. thesis, Universität Passau, Passau, Germany (1990).

[16] Geuvers, H., *The Church-Rosser property for $\beta\eta$-reduction in typed λ-calculi*, in: *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society Press, Santa Cruz, California, 1992, pp. 453–460.

[17] Geuvers, H., "Logics and Type Systems," Ph.D. thesis, Computer Science Institute, Katholieke Universiteit Nijmegen (1993).

[18] Goguen, H., *A typed operational semantics for type theory*, LFCS report ECS-LFCS-94-304, University of Edinburgh, Department of Computer Science (1994).

[19] Luo, Z., "Computation and Reasoning: A Type Theory for Computer Science," Number 11 in International Series of Monographs on Computer Science, Oxford University Press, 1994.

[20] Paulin-Mohring, C., *Inductive definitions in the system Coq. Rules and properties*, in: M. Bezem and J. F. Groote, editors, *Proceedings of the* 1^{st} *International Conference on Typed Lambda Calculi and Applications, TCLA'93,* Utrecht, The Netherlands, Lecture Notes in Computer Science **664** (1993), pp. 328–345.

[21] Soloviev, S. V., *The category of finite sets and Cartesian closed categories*, in: *Theoretical Applications of Methods of Mathematical Logic III*, Zapiski Nauchnykh Seminarov LOMI **105**, Nauka, Leningrad, 1981 pp. 174–194, english translation in *Journal of Soviet Mathematics*, 22(3) (1983), 1387–1400.

[22] Walukiewicz, D., *Termination of rewriting in the calculus of constructions*, Lecture Notes in Computer Science **1443** (1998).

[23] Werner, B., *Méta-théorie du Calcul des Constructions Inductives*, Thèse Univ. Paris VII, France (1994).