On Isomorphism of Dependent Products in a Typed Logical Framework*

Sergei Soloviev[1,2]

1 **IRIT, University of Toulouse**
118 route de Narbonne, 31062 Toulouse, France
`soloviev@irit.fr`
2 **associated researcher at**
ITMO University
St. Petersburg, Russia

Abstract

A complete decision procedure for isomorphism of kinds that contain only dependent product, constant *Type* and variables is obtained. All proofs are done using Z. Luo's typed logical framework. They can be easily transferred to a large class of type theories with dependent product.

1998 ACM Subject Classification F.4.1 Lambda calculus and related systems, D.2.13 Reuse models, D.3.3 Frameworks

Keywords and phrases Isomorphism of types, dependent product, logical framework

Digital Object Identifier 10.4230/LIPIcs.TYPES.2014.274

1 Introduction

Why an axiomatization of the isomorphism relation between types in dependent type systems (based on type rewriting, as in the case of simply-typed λ-calculus or system F, see, *e.g.*, [1, 3, 15]) was never considered? Why no complete decision procedure for this relation was developed there? Isomorphism of dependent types is used to some extent in proof assistants based on dependent type systems, such as Coq (*cf.* [4]). We could find only one paper by D. Delahaye [5] where the author tries to explore type isomorphisms in the Calculus of Constructions along the lines used in the above-mentioned papers. On a theoretical side, isomorphisms play also an important role in the study of Univalent Foundations [9]. There are some studies of isomorphisms of inductive types [2, 6], but little is done on isomorphisms even in the "core" of logical frameworks (including, *e.g.*, dependent product).

In the paper [5] dependent product *and* dependent sum are considered but no complete axiomatisation (suitable for "non-contextual" rewriting) or complete decision procedure is obtained. As Delahaye writes:

- we have developed a theory Th^{ECCE} with "ad hoc" contextual rules, which is sound for $ECCE$;
- we have made contextual restrictions on Th^{ECCE} to build a decision procedure Dec^{Coq} which is sound for Th^{ECCE} and which is an approximation of the contextual part of Th^{ECCE};
- we have implemented Dec^{Coq} in a tool called $SearchIsos$.

* This work was partially supported by Government of the Russian Federation Grant 074-U01.

20th International Conference on Types for Proofs and Programs (TYPES 2014).
Editors: Hugo Herbelin, Pierre Letouzey, and Matthieu Sozeau; pp. 274–287
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

It should be said that some points concerning the notion of isomorphism in *ECCE* used in [5] remain not very clear. In particular, he needs η-rules to justify all axioms he considers. It is known that with Σ-types and cumulative hierarchy, the CR-property does not hold in the presence of η-contraction scheme (and similar scheme for surjective pairing) [11], p.50.

In this paper we are going to obtain a *complete* decision procedure in a "core" system that contains only dependent product, constant $Type$ and variables. We shall consider mostly the logical framework proposed by Z. Luo [11]. It is sufficiently close to the Martin-Löf logical framework or to the Calculus of Constructions, but in difference from Martin-Löf's original system it is typed, and in difference from the Calculus of Constructions it has the explicit equality rules even for standard $\beta\eta$-conversions, and this is more convenient for the study that concerns both isomorphism and equality. In difference from Edinburgh LF it permits to specify other type theories. This system (without Σ-types and cumulative hierarchy) is confluent with respect to $\beta\eta$-reductions.

We shall use the notation from [11]. In particular, $(x : K)K'$ will denote dependent product and $[x : K]P$ abstraction (instead of frequently used $\Pi x : K.K'$ and $\lambda x : K.P$). If $x : K$ does not occur freely in K', it will be written $(K)K'$ (that corresponds to $K \to K'$ in simply typed lambda calculus).

As to the above mentioned "core part", an answer may be that the isomorphism relation between dependent product kinds seems at a first glance too limited to be interesting. Among "basic" isomorphisms, there is only one obvious isomorphism that corresponds to the isomorphism $A \to (B \to C) \sim B \to (A \to C)$ of simply-typed λ-calculus. The corresponding isomorphism in dependent type case is

$$\Gamma \vdash (x : A)(y : B)C \sim (y : B)(x : A)C.$$

Here A, B, C are kinds, $(x : A)D$ means dependent product. In difference from simply-typed calculus we need a context Γ because A, B, C may contain free variables. The variable x must be not free in B and y in A, and $(x : A)(y : B)C$, $(y : B)(x : A)C$ must be well formed kinds in Γ.

Notice that *a priori* it does not exclude the existence of isomorphisms that are *not* generated by this basic isomorphism (*cf.* [7]).

In fact, though, there are some other aspects that make even the isomorphisms in the "core part" of dependent type systems interesting. The role of contexts (variable declarations) is to be taken into account. The equality of kinds is non-trivial and it has an influence on (the defintion of) the isomorphisms: for example, the condition that x is not free in B above may be not satisfied but B may be equal to B_0 that does not contain x free.

In the "core part" itself the role of contexts is rather superficial, but it shows what is to be expected if we consider more sophisticated type theories defined using logical frameworks.

The next aspect is more important. It is illustrated by the following example. Let $\Gamma \vdash A \sim A'$. Consider $\Gamma' \vdash (x : A)B$. Let $\Gamma \vdash P : (x : A)A'$ be the term that represents the isomorphism between A and A' and $\Gamma \vdash P' : (x : A')A$ the term that represents its inverse (in this case x is not free in A' and x' is not free in A). Then, in difference from the simply-typed case where $(x : A)B \sim (x : A')B$, the isomorphism P' appears inside B:

$$\Gamma \vdash (x : A)B \sim (x' : A')[(P'x')/x]B$$

($[(P'x')/x]$ denotes substitution). Notice that there may be many mutually inverse isomorphisms between A and A' (represented by $P_1, P_1', \ldots, P_n, P_n', \ldots$ and the structure of the "target" type $(x : A')[(P_i'x)/x]B$ depends on their choice. (This was noticed already in [5].) Thus, if we see the isomorphic transformation as rewriting, this rewriting is not *local*, and there is little hope that one can describe the isomorphism relation between types using rewriting rules for types[1] as in, *e.g.*, [1, 3, 15].

2 Basic definitions

We consider Z. Luo's typed logical framework LF [11].

Because LF is mostly used to specify type theories, types in LF are called kinds (to distinguish them from types in the specified type theories). In LF there are five forms of judgements (below $\Gamma \vdash J$ will be sometimes used as a generic notation for one of these five judgement forms):

- $\Gamma \vdash \mathbf{valid}$ (Γ is a valid context);
- $\Gamma \vdash K\mathbf{kind}$ (K is a kind in the context Γ);
- $\Gamma \vdash k : K$ (k is an object of the kind K);
- $\Gamma \vdash k = k' : K$ (k and k' are equal objects of the kind K);
- $\Gamma \vdash K = K'$ (K and K' are equal kinds in Γ).

There are the following inference rules in LF (we use here an equivalent formulation which is more convenient proof-theoretically, *cf.* [14]):

Contexts and assumptions

$$(1.1)\ \frac{}{<>\vdash \mathbf{valid}} \qquad (1.2)\ \frac{\Gamma \vdash K\mathbf{kind} \quad x \notin FV(\Gamma)}{\Gamma, x:\ K \vdash \mathbf{valid}} \qquad (1.3)\frac{\Gamma, x:\ K, \Gamma' \vdash \mathbf{valid}}{\Gamma, x:\ K, \Gamma' \vdash x:\ K}$$

$$\frac{\Gamma_1, \Gamma_2 \vdash J \quad \Gamma_1, \Gamma_3 \vdash \mathbf{valid}}{\Gamma_1, \Gamma_3, \Gamma_2 \vdash J}(wkn)$$

$$(\text{where } FV(\Gamma_2) \cap FV(\Gamma_3) = \emptyset).$$

General equality rules

$$(2.1)\ \frac{\Gamma \vdash K\mathbf{kind}}{\Gamma \vdash K = K} \qquad (2.2)\ \frac{\Gamma \vdash K = K'}{\Gamma \vdash K' = K} \qquad (2.3)\ \frac{\Gamma \vdash K = K' \quad \Gamma \vdash K' = K''}{\Gamma \vdash K = K''}$$

$$(2.4)\ \frac{\Gamma \vdash k : K}{\Gamma \vdash k = k : K} \qquad (2.5)\ \frac{\Gamma \vdash k = k' : K}{\Gamma \vdash k' = k :\ K} \qquad (2.6)\frac{\Gamma \vdash k = k' : K \quad \Gamma \vdash k' = k'' : K}{\Gamma \vdash k = k'' :\ K}$$

Retyping rules

$$(3.1)\frac{\Gamma \vdash k:\ K \quad \Gamma \vdash K = K'}{\Gamma \vdash k:\ K'} \qquad (3.2)\frac{\Gamma \vdash k = k':\ K \quad \Gamma \vdash K = K'}{\Gamma \vdash k = k':\ K'}$$

$$(3.3)\frac{\Gamma, x : K, \Gamma' \vdash J \quad \Gamma \vdash K = K'}{\Gamma, x : K', \Gamma' \vdash J}$$

---

[1] Maybe, it is better to say "non-contextual" instead of "local". But what is needed here is more than dependency on context of the applicability of a rewriting rule. In fact all occurrences of x (indefinitely many) must be simultaneously replaced by $P'x'$. The inclusion of an explicit substitution rule as a part of rewriting process may have its own drawbacks. The rewriting rules considered in [5] that take into account this observation are called there "contextual", but to us this terminology does not seem perfect. Indeed, the "context" has to be changed simultaneously, otherwise at some point the expression will not be well typed. There is also some confusion of the rewriting "context" in this sense, and the usual type-theoretical contexts of variable declarations.

The kind Type

$$(4.1)\frac{\Gamma \vdash \textbf{valid}}{\Gamma \vdash \mathit{Type}\textbf{kind}} \quad (4.2)\frac{\Gamma \vdash A : \mathit{Type}}{\Gamma \vdash \mathit{El}(A)\textbf{kind}} \quad (4.3)\frac{\Gamma \vdash A = B : \mathit{Type}}{\Gamma \vdash \mathit{El}(A) = \mathit{El}(B)}$$

Dependent product (kinds and terms)[2]

$$(5.1)\frac{\Gamma, x : K \vdash K'\textbf{kind}}{\Gamma \vdash (x : K)K'\textbf{kind}} \qquad (5.2)\frac{\Gamma, x : K_1 \vdash K_1' = K_2' \quad \Gamma \vdash K_1 = K_2}{\Gamma \vdash (x : K_1)K_1' = (x : K_2)K_2'}$$

$$(5.3)\frac{\Gamma, x : K \vdash k : K'}{\Gamma \vdash [x : K]k : (x : K)K'} \qquad (5.4)\frac{\Gamma, x : K_1 \vdash k_1 = k_2 : K \quad \Gamma \vdash K_1 = K_2}{\Gamma \vdash [x : K_1]k_1 = [x : K_2]k_2 : (x : K_1)K}$$

$$(5.5)\frac{\Gamma \vdash f : (x : K)K' \quad \Gamma \vdash k : K}{\Gamma \vdash f(k) : [k/x]K'} \quad (5.6)\frac{\Gamma \vdash f = f' : (x : K)K' \quad \Gamma \vdash k_1 = k_2 : K}{\Gamma \vdash f(k_1) = f'(k_2) : [k_1/x]K'}$$

$$(5.7)\frac{\Gamma, x : K \vdash k' : K' \quad \Gamma \vdash k : K}{\Gamma \vdash ([x : K]k')k = [k/x]k' : [k/x]K'} \quad (5.8)\frac{\Gamma \vdash f : (x : K)K' \quad x \notin FV(\Gamma)}{\Gamma \vdash [x : K]f(x) = f : (x : K)K'}$$

Substitution rules

$$(6.1)\frac{\Gamma, x : K, \Gamma' \vdash \textbf{valid} \quad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash \textbf{valid}} \qquad (6.2)\frac{\Gamma, x : K, \Gamma' \vdash K'\textbf{kind} \quad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K'\textbf{kind}}$$

$$(6.3)\frac{\Gamma, x : K, \Gamma' \vdash k' : K' \quad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]k' : [k/x]K'} \qquad (6.4)\ \frac{\Gamma, x : K, \Gamma' \vdash K' = K'' \quad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K' = [k/x]K''}$$

$$(6.5)\frac{\Gamma, x : K, \Gamma' \vdash k' = k'' : K' \quad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]k' = [k/x]k'' : K'}$$

$$(6.6)\ \frac{\Gamma, x : K, \Gamma' \vdash K'\textbf{kind} \quad \Gamma \vdash k_1 = k_2 : K}{\Gamma, [k_1/x]\Gamma' \vdash [k_1/x]K' = [k_2/x]K'}$$

$$(6.7)\ \frac{\Gamma, x : K, \Gamma' \vdash k' : K' \quad \Gamma \vdash k_1 = k_2 : K}{\Gamma, [k/x]\Gamma' \vdash [k_1/x]k' = [k_1/x]k' : [k/x]K'}$$

In the syntax of LF $(x : K)K'$ denotes dependent product, and $[x : K]k$ denotes abstraction, x is considered as bound in K' and k respectively. In case when x is not free in K' we shall write $(K)K'$ instead of $(x : K)K'$. We shall use $\equiv$ for syntactic identity.

One of the fundamental properties of derivations in LF is that the inferences of substitutions, *wkn* and context-retyping 3.3 that create problems with structural induction on derivations can be eliminated, *i.e.*, a judgement is derivable iff it has a substitution, context-retyping and *wkn*-free derivation ([14], Theorem 3.1, [12], Definition 3.12 and Algorithm 3.13).

In [12] such derivations are called canonical (Definition 3.12). The following technical lemmas are easily proved by induction on the size of canonical derivation in LF.

▶ **Lemma 1.** *Let $\Gamma, \Gamma', \Gamma'' \vdash J$ be any judgement derivable in LF. If the variables from Γ' do not occur into Γ'' and J, then $\Gamma, \Gamma'' \vdash J$ is derivable.*

[2] To facilitate reading, let us notice that the syntax of raw kinds and terms is very simple: $K ::= \mathit{Type} \,|\, \mathit{El}(P) \,|\, (x : K)K', \quad P ::= x \,|\, (PQ) \,|\, [x : K]P$.

Since in LF types (*kinds*) of variables may depend on terms (other variables) the variables cannot any more be freely permuted. Let us formulate some statements (without detailed proofs) that we shall use below.

Let us consider the list of variables with *kinds*, $u_1 : Q_1, \ldots, u_k : Q_k$. Let $u_i \lhd u_j$ denote that u_i occurs in the kind Q_j of u_j. The same applies to prefixes like $(u_1 : Q_1) \ldots (u_k : Q_k)Q$ and $[u_1 : Q_1] \ldots [u_k : Q_k]Q$.

Let $u_1 : Q_1, \ldots, u_k : Q_k$ be part of a valid context (respectively $(u_1 : Q_1) \ldots (u_k : Q_k)Q$, $[u_1 : Q_1] \ldots [u_k : Q_k]Q$ be part of derivable *kind* or term). In this case the relation $\lhd$ generates a partial order on indexes $1, \ldots, k$ which we shall denote by $\lhd*$.

▶ **Lemma 2.** *Consider the judgements*

$$\Gamma, x_1 : K_1, \ldots, x_n : K_n, \Gamma' \vdash \textbf{valid},$$

$$\Gamma \vdash (x_1 : K_1) \ldots (x_n : K_n)K_0\textbf{kind},$$

$$\Gamma \vdash [x_1 : K_1] \ldots [x_n : K_n]P : (x_1 : K_1) \ldots (x_n : K_n)K_0$$

in LF. For any permutation σ that respects the order $\lhd*$,

$$\Gamma, x_{\sigma_1} : K_{\sigma_1}, \ldots, x_{\sigma_n} : K_{\sigma_n}, \Gamma' \vdash \textbf{valid},$$

$$\Gamma \vdash (x_{\sigma_1} : K_{\sigma_1}) \ldots (x_{\sigma_n} : K_{\sigma_n})K_0\textbf{kind},$$

$$\Gamma \vdash [x_{\sigma_1} : K_{\sigma_1}] \ldots [x_{\sigma_n} : K_{\sigma_n}]P : (x_{\sigma_1} : K_{\sigma_1}) \ldots (x_{\sigma_n} : K_{\sigma_n})K_0$$

are derivable in LF.

Besides standard equality rules, equality in LF is defined by the rules (5.7, 5.8). Obviously, it is based on β and η conversions (incorporated explicitly using 5.7 and 5.8). This permits to define conversions in a more familiar way.

▶ **Proposition 3.**

1. *Let J be an LF-judgement (of any of the five forms described above) and v an occurrence of an expression either of the form $([x : K]P)S$ or of the form $[x : K](Px)$ with x not free in P. Let J' be obtained by replacement of v by the occurrence of $[S/x]P$ or P respectively. Then J is derivable in LF iff J' is derivable. (We shall say that one is obtained from another by β, respectively η conversion.)*
2. *Let J be of the form $\Gamma \vdash K\textbf{kind}$ or $\Gamma \vdash P : K$ respectively and v belong to K (respectively, to P). Let K', respectively, P' be obtained from K (k') as in 1. If J is derivable, the equality $\Gamma \vdash K = K'$, respectively, $\Gamma \vdash P = P' : K$ is derivable.*

Proof.

1. By induction on the length of a canonical derivation of J.
2. By induction on the length of a canonical derivation of J and (for one of implications) on the length of series of conversions. ◀

We shall use the fact that LF is strongly normalizing and has the Church-Rosser property with respect to β- and η-reductions, see H. Goguen's thesis [8], and also [13]. H. Goguen applied typed operational semantics to LF and its extension UTT to prove these results; L. Marie-Magdeleine in [13] applied Goguen's method to UTT with certain additional equality rules. We do not always need SN and CR in our proofs, but since we want to concentrate our attention on isomorphisms, the use of SN and CR permits to make some shortcuts.

In simply-typed λ-calculus the definition of invertibility of terms may use contexts (that include free term variables). This is relevant for the study of retractions [16]. However, type variables and terms variables are completely separated, and to describe the isomorphisms of types it is enough to consider closed terms [3]. Equality of types coincides with identity. One says that the types A, B are isomorphic iff there exist terms $P : A \to B$ and $P' : B \to A$ such that $P' \circ P =_{\beta\eta} \lambda x : A.x$ and $P \circ P' =_{\beta\eta} \lambda x : B.x$.

In dependent type systems the equality of types (kinds) depends on equality of terms. Some variables from the context may occur in kinds whose isomorphism we want to check. This motivates the following definition.

▶ **Definition 4.** Let $\Gamma \vdash K\mathbf{kind}$ and $\Gamma \vdash K'\mathbf{kind}$. We shall say that K, K' are isomorphic in Γ iff there exist terms $\Gamma \vdash P : (K)K'$, $\Gamma \vdash P' : (K')K$ such that

$$(*)\ \ \Gamma \vdash P' \circ P = [x : K]x : (K)K, \quad \Gamma \vdash P \circ P' = [x : K']x : (K')K'.$$

▶ Remark 5. Equal kinds may contain different free variables, and it has to be taken into account. If we consider $\beta\eta$-normal forms of K and K', they may not contain some free variables that are present in K and K'. The normal forms may be well-formed in a more narrow context Γ_0. Still, the isomorphism of K and K' will not hold in Γ_0 because Γ is necessary to prove the equality of kinds to their normal forms. In case of $\beta\eta$-equality one may try to define some sort of "minimal" context, but when the extensions of LF are more "exotic", this may be not possible (at the moment, we study one such extension, a generalization to dependent type systems of axiom C [10]).

3 Isomorphism of Kinds in LF

At a first glance, the theory of isomorphisms in LF cannot be very interesting. Indeed, with respect to the LF-equality there exists only one "basic" isomorphism, but as it turns out even this basic isomorphism generates in LF much more intricate isomorphism relation than in the simply-typed case.

▶ **Example 6.** Let $\Gamma \vdash (x : K_1)(y : K_2)K\mathbf{kind}$ be derivable in LF and $x \notin FV(K_2)$. Then $\Gamma \vdash (y : K_2)(x : K_1)K\mathbf{kind}$ is derivable and $(x : K_1)(y : K_2)K \sim (y : K_2)(x : K_1)K$ in Γ. The terms

$$[z : (x : K_1)(y : K_2)K][y : K_2][x : K_1](zxy),$$

$$[z : (y : K_2)(x : K_1)K][x : K_1][y : K_2](zyx)$$

are mutually inverse isomorphisms between these *kinds*.

This example corresponds directly to the well known example of isomorphism in simply-typed lambda calculus. In difference from simply-typed λ-calculus, there are some technical points that have to be proved, such as the fact that the derivability of $\Gamma \vdash (y : K_2)(x : K_1)K\mathbf{kind}$ follows from the derivability of $\Gamma \vdash (x : K_1)(y : K_2)K\mathbf{kind}$. For example, the derivability of $\Gamma \vdash (x : K_1)(y : K_2)K\mathbf{kind}$ implies the derivability of $\Gamma, x : K_1, y : K_2 \vdash K\mathbf{kind}$ and this implies the derivability of $\Gamma, y : K_2, x : K_1 \vdash K\mathbf{kind}$ because $x \notin FV(K_2)$ (cf. Lemma 2).

▶ **Example 7.** Let in the previous example $\Gamma \vdash K_1 = K_2$. Then there exists at least two isomorphisms between $(x : K_1)(y : K_2)K$ and $(y : K_2)(x : K_1)K$ in Γ. Indeed, one isomorphism is the identity isomorphism, and another is the isomorphism considered in the previous example.

The following example shows the "non-locality" of syntactic rewriting relation associated with isomorphisms in LF.

▶ **Example 8.** Let $\Gamma \vdash (x : K_1)K\mathbf{kind}$ be derivable in LF. Let $K_1 \sim K_2$ in Γ, and $\Gamma \vdash P : (K_1)K_2$, $\Gamma \vdash P' : (K_2)K_1$ be mutually inverse isomorphisms. Then $(x : K_1)K \sim (x : K_2)[(P'x)/x]K$ in Γ. The isomorphism from the first to the second kind is given by the following term:

$$\Gamma \vdash [z : (x : K_1)K][x : K_2](z(P'x)) : ((x : K_1)K)(x : K_2)[(P'x)/x]K,$$

and its inverse by

$$\Gamma \vdash [z : (x : K_2)[(P'x)/x]K][x : K_1](z(Px)) : ((x : K_2)[(P'x)/x]K)(x : K_1)K.$$

Notice that after second substitution (generated by the application of z in the second line) P' and P being mutually inverse isomorphisms will cancel each other. Notice also that since P and P' may be not a unique pair of isomorphisms between K_1 and K_2, the replacement of K_1 by K_2 does not uniquely determine the "target" kind. We cannot merely replace K_1 by K_2 (without introducing P' in K) because the correct kinding inside K may be lost.

Let $\Gamma \vdash P : El(A)$ be provable in LF. Then P is either a variable or an application. Formal proof can be done by induction on the length of derivation of $\Gamma \vdash P : El(A)$.

▶ **Lemma 9.** *Let* $\Gamma \vdash El(A)\mathbf{kind}$ *and* $\Gamma \vdash El(B)\mathbf{kind}$*. Then* $\Gamma \vdash El(A) \sim El(B)$ *iff* $\Gamma \vdash El(A) = El(B)$*. The isomorphism between* $El(A)$ *and* $El(B)$ *is unique up to equality in LF and is represented by the term* $\Gamma \vdash [x : El(A)]x : (El(A))El(B)$*.*

Proof. Consider the non-trivial "if". Assume there exist mutually inverse isomorphisms $\Gamma \vdash P : (El(A))El(B)$ and $\Gamma \vdash P' : (El(B))El(A)$. That is, the compositions of P and P' are equal to identities:

$$\Gamma \vdash [x : El(A)](P'(Px)) = [x : El(A)]x : (El(A))El(A),$$

$$\Gamma \vdash [x : El(B)](P(P'x)) = [x : El(B)]x : (El(B))El(B)$$

(with x fresh).

Without loss of generality we may assume that each of P, P' is normal. Consider, e.g., P'. It may have either the form $[y : El(B)](zk_1 \dots k_n)$ or the form $zk_1 \dots k_n$ (z being a variable). It is easily seen that in the second case the whole cannot normalize to $[x : El(A)]x$. In the first case, if it normalizes to $[x : El(A)]x$, n must be 1 and $[(Px)/y]k_1$ must normalize to x. Similar analysis of the form of P leads to the conclusion of the lemma. ◀

▶ **Theorem 10.** *Let* $\Gamma \vdash K\mathbf{kind}$ *in LF. Then:*

1. *the number of kinds (considered up to equality) that are isomorphic to* K *in LF in the context* Γ *is finite;*
2. *for every kind* $\Gamma \vdash K'\mathbf{kind}$ *such that* $K \sim K'$ *in* Γ*, the number of isomorphisms between* K *and* K' *in* Γ *is finite;*
3. *there exists an algorithm that lists all these isomorphisms (and kinds).*

First, let us notice that if $\Gamma \vdash K\mathbf{kind}$ is derivable in LF then K has either the form $(x_1 : K_1)\dots(x_n : K_n)El(A)$ or the form $(x_1 : K_1)\dots(x_n : K_n)\mathbf{Type}$. This can be easily proved by induction on the derivation of $\Gamma \vdash K\mathbf{kind}$ in LF.

Proof of Theorem 10. The proof will proceed by induction on rank of K which is defined as follows.

▶ **Definition 11.** If $K \equiv \mathbf{Type}$ or $K \equiv El(A)$ then $rank(K) = 0$. If $K \equiv (x : K_1)K_2$ then $rank(K) = max(rank(K_1), rank(K_2)) + 1$.

▶ Remark 12. The $rank(K)$ is not changed by β- and η-reductions inside K and by substitution: $rank(K) = rank([k/x]K)$.

The *base case* of induction is assured by Lemma 9.

To proceed, we shall use type erasure and Dezani's theorem about invertible terms in untyped λ-calculus[3] as in [1, 3]. Of course, some modifications to take into account dependent types will be necessary. Before we continue with the proof of the theorem, several definitions and auxiliary statements are needed.

▶ **Definition 13.** Let $\Gamma \vdash P : K$ in LF. By $e(P)$ we shall denote the λ-term obtained by erasure of all kind-labels in P (and replacement of all expressions $[x]$ by λx). We shall call term variables of P all variables that occur in $e(P)$.

The following definition is a refined (equivalent) reformulation of definition 1.9.2 of [3].

▶ **Definition 14.** An untyped λ-term M with one free variable x is a finite hereditary permutation (f.h.p.) iff

- $M \equiv x$, or
- there exists a permutation $\sigma : n \to n$ such that $M \equiv \lambda x_{\sigma_1} \ldots .x_{\sigma_n}.xP_1 \ldots P_n$ (the only free variable of M is x, and its unique occurrence is explicitly shown) where the only free variable of P_i is x_i and P_i is a finite hereditary permutation (for all $1 \le i \le n$).
- If M is a f.h.p. then the term $\lambda x.M$ will be called its closure. We shall also say that it is closed finite hereditary permutation (c.f.h.p.). The notion of c.f.h.p. corresponds to f.h.p. of [3].

▶ Remark 15. In "standard" cases the passage from the term P such that $e(P)$ is a f.h.p. to the term whose erasure is a c.f.h.p. is done by a single abstraction:

$$\frac{\Gamma, z : K \vdash P : K'}{\Gamma \vdash [z : K]P : (z : K)K'}.$$

We do not "abstract" the "head variable" of a f.h.p. because sometimes we want to by-step the problem of permutability of variables in LF if the head variable is not the rightmost variable of a context.

The result similar to simply-typed λ-calculus holds.

▶ **Proposition 16** (cf. Theorem 1.9.5 of [3]). *If $\Gamma \vdash P : (K)K'$ and $\Gamma \vdash P' : (K')K$ are mutually inverse terms in LF then $e(P)$ and $e(P')$ are c.f.h.p.*

If $e(P)$ is a c.f.h.p. then P has the structure

$$[z : (x_1 : K_1) \ldots (x_n : K_n)K_0][x'_{\sigma_1} : K'_{\sigma_1}] \ldots [x'_{\sigma_n} : K'_{\sigma_n}](zP_1 \ldots P_n).$$

When we consider invertible terms, we always can assume that they are normal and in such a form.

[3] Probably the "simply-typed erasure": to replace all occurrences of $El(A)$ by El (considered as another constant *kind*) will work as well, but it seems that a fully justified application of this method may need as much technical lemmas as the proof that we propose below.

In difference from simply-typed λ-calculus, there are additional constraints on the possible permutations in LF, because some of the types K_j may depend on variables $x_i, i < j$, and in this case permutation of x_i and x_j destroys typability. As a consequence, if $e(P)$ is a f.h.p. then the original term is not necessarily well typed.

▶ **Example 17.** The term

$$P \equiv [z : (x : K_1)(y : K_2(x))K][y : K_2(x)][x : K_1](zxy)$$

is not well typed in LF, but $e(P)$ is a c.f.h.p.

Let us prove some lemmas concerning properties of well typed terms P such that $e(P)$ is a f.h.p. (they can be easily reformulated for c.f.h.p.)

▶ **Lemma 18.** *Let $\Gamma \vdash P : K'$ be derivable in LF, and $e(P)$ be a f.h.p. with head variable $z : K$, $K \equiv (x_1 : K_1) \dots (x_n : K_n)K_0$. Let*

$$P \equiv [x'_{\sigma_1} : K'_{\sigma_1}] \dots [x'_{\sigma_n} : K'_{\sigma_n}](zP_1 \dots P_n).$$

If x is a free variable that occurs in $P_1, \dots, P_n$ then it occurs in the kind of z.

Proof. There is no occurrence of x as term variable of f.h.p. because of the properties of f.h.p. Let there be an occurrence of x into *kinds* of variables in P_i. Notice that since P is well typed, $P_1 : K_1$ (in appropriate context), $P_2 : [P_1/x_1]K_2, \dots, P_n : [P_{n-1}/x_{n-1}](\dots[P_1/x_1]K_n$, $zP_1 \dots P_n : [P_n/x_n](\dots[P_1/x_1]K_0 = K'_0$.

Consider the Böhm-tree of $e(P)$ [1, 3]. We order the paths (from the root to nodes, not necessarily to leaves) lexicographically, in such a way that the path "more to the left" is less that the paths "more to the right". Now, we may find the smallest path such that the variable in some P_i that corresponds to the occurrence at its end contains x in its *kind*.

If the length of the path is 1, the occurrence lies in the prefix of P_i, and a matching occurrence of x into $[P_{i-1}/x_{i-1}](\dots[P_1/x_1]K_i$ must exist. Indeed, it cannot come from $P_j, j < i$ due to the choice of the path, so it comes from K_i.

Now, assume that the path is longer. Then we obtain a contradiction. Indeed, x must occur into the "abstracted" prefix of some subterm of some P_i, *i.e.*, it lies within an occurrence of the form $yQ_1 \dots Q_k$, and belongs to the prefix of some of Q_i. As above, we arrive at the conclusion that a matching occurrence into *kind* of y must exist, but y belongs to the abstracted prefix at the previous node of the same path. ◀

▶ **Corollary 19.** *Let $\Gamma \vdash P : K'$ be derivable in LF, $e(P)$ be a f.h.p. and z occur as the "head variable" of P. Then there is no other occurrences of z into P, even in the kinds of other variables.*

Proof. Since $e(P)$ is a f.h.p. z could occur (except the "head") only into kinds of variables in P. But then by the previous lemma it must occur into its own *kind* and this is impossible. ◀

▶ **Lemma 20.** *Let (as above) $\Gamma \vdash P : K'$ be derivable in LF, and $e(P)$ be a f.h.p. The free variable x occurs in K' iff it occurs in the kind of the head variable of P.*

Proof. As above, $P \equiv [x'_{\sigma_1} : K'_{\sigma_1}] \dots [x'_{\sigma_n} : K'_{\sigma_n}](zP_1 \dots P_n)$, where $z : (x_1 : K_1) \dots (x_n : K_n)K_0$. The variable x'_i is the head variable of P_i (by properties of f.h.p.).

We proceed by induction on the depth of the Böhm-tree. If the depth is 1, $K' = (x_{\sigma_1} : K_{\sigma_1}) \dots (x_{\sigma_n} : K_{\sigma_n})K_0$ and the lemma is obvious (P_i are variables and P is just a permutation).

Let x occur in the *kind* $K' \equiv (x'_{\sigma_1} : K'_{\sigma_1}) \dots (x'_{\sigma_n} : K'_{\sigma_n})K'_0$. Does it imply that it occurs in the *kind* of z? As above,

- $P_1 : K_1$ (in appropriate context),
- $P_2 : [P_1/x_1]K_2$,
- $\dots$,
- $P_n : [P_{n-1}/x_{n-1}](\dots [P_1/x_1]K_n)$,
- $zP_1 \dots P_n : [P_n/x_n](\dots [P_1/x_1]K_0) = K'_0$.

There are three possibilities: (i) x occurs in the *kind* of z (and we are done); (ii) x occurs in one of K'_{σ_i}; (iii) x occurs in one of the P_i and into K'_0 (via substitution).

In case (ii) x occurs in *kind* of the head variable of P_{σ_i}. We may apply I.H. (for implication in opposite direction) and deduce that x occurs also in the *kind* of P_{σ_i}. We always may choose the leftmost of such P_{σ_j}, and conclude that a matching occurrence of x must exist in the *kind* of z.

In case (iii) we use Lemma 18 and arrive to the previous case.

Now, let us consider the opposite implication for P. Let x occur in the *kind* of z. Either it lies in K_0 (and then will occur in the *kind* of P as well) or it must be matched by the *kind* of one of P_i. Then by I.H. it occurs also in the *kind* of its head variable and into the prefix of P, and thus into K'. ◀

▶ **Corollary 21.** *If $\Gamma \vdash P : (K)K'$ is an isomorphism in LF (all is in normal form) the same free variables occur into K and K'.*

Proof. The term $e(P)$ has to be a c.f.h.p., so

$$P \equiv [z : (x_1 : K_1) \dots (x_n : K_n)K_0][x'_{\sigma_1} : K'_{\sigma_1}] \dots [x'_{\sigma_n} : K'_{\sigma_n}](zP_1 \dots P_n).$$

We apply previous lemma to $[x'_{\sigma_1} : K'_{\sigma_1}] \dots [x'_{\sigma_n} : K'_{\sigma_n}](zP_1 \dots P_n)$ in context $\Gamma, z : (x_1 : K_1) \dots (x_n : K_n)K_0$. ◀

Below we consider some properties of dependency of variables (relations $\lhd$ and $\lhd*$) that we shall use in our study of isomorphism.

▶ **Lemma 22.** *Let us consider $\Gamma \vdash P : (K)K'$, $\Gamma \vdash P' : (K')K$ such that*

- *$\Gamma \vdash P : (K)K'$, $\Gamma \vdash P' : (K')K$ are derivable in LF,*
- *$P \equiv [z : (x_1 : K_1) \dots (x_n : K_n)K_0][x'_{\sigma_1} : K'_{\sigma_1}] \dots [x'_{\sigma_n} : K'_{\sigma_n}](zP_1 \dots P_n)$,*
- *$P' \equiv [z' : (x_{\sigma_1} : K_{\sigma_1}) \dots (x_{\sigma_n} : K_{\sigma_n})K'_0][x_1 : K_1] \dots [x_n : K_n](z'P'_{\sigma_1} \dots P'_{\sigma_n})$,*
- *and $e(P)$ and $e(P')$ are mutually inverse c.f.h.p.*

Then $x_i \lhd x_j$ iff $x'_i \lhd x'_j$.

Proof. Without loss of generality, we may consider also the terms $[x'_{\sigma_1} : K'_{\sigma_1}] \dots [x'_{\sigma_n} : K'_{\sigma_n}](zP_1 \dots P_n)$ in the context $\Gamma, z : (x_{\sigma_1} : K_{\sigma_1}) \dots (x_{\sigma_n} : K_{\sigma_n})K'_0$ and $x_1 : K_1] \dots [x_n : K_n](z'P'_{\sigma_1} \dots P'_{\sigma_n})$ in the context $\Gamma, z' : (x'_{\sigma_1} : K'_{\sigma_1}) \dots (x'_{\sigma_n} : K'_{\sigma_n})K'_0$. Let us prove that $x_i \lhd x_j \Rightarrow x'_i \lhd x'_j$[4].

Because $e(P)$ is a c.f.h.p., the head variables of P_i are x'_i $(1 \le i \le n)$. Since $x_i \lhd x_j$, x_i occurs in K_j, the type of P_j (in appropriate context) is

$$[P_{j-1}/x_{j-1}](\dots [P_1/x_1]K_j),$$

[4] Let us emphasize that here the dependency between x_i and x_j, respectively x'_i and x'_j should be considered, not between x'_{σ_i} and x'_{σ_j}.

thus x'_i does occur in the *kind* of P_j. By Lemma 20 it occurs also into the *kind* of x'_j, and $x'_i \lhd x'_j$.

For opposite implication, we consider P'. ◀

▶ **Corollary 23.** *The partial order $\lhd*$ generated by $\lhd$ on $x_1, \dots, x_n$ coincides with $\lhd*$ generated by $\lhd$ on $x'_1, \dots, x'_n$.*

One more lemma:

▶ **Lemma 24.** *Let $P \equiv [x'_{\sigma_1} : K'_{\sigma_1}] \dots [x'_{\sigma_n} : K'_{\sigma_n}](zP_1 \dots P_n)$ as above. Then x'_i, the head variable of P_i, does not occur in $P_j, j < i$. Similar result holds for P'.*

Proof. The proof is based on the same idea as in Lemma 19. We consider the Böhm-tree of $e(P)$ and the ordering of the paths as above. We assume that x'_j does occur in (the *kind* of) some variable in P_j Then there is a smallest path leading to corresponding occurrence of an abstracted variable in the tree.

If it belongs to the prefix of P_j (the path has the length 1) then x'_j must belong to K_j in the type of z (because of minimality of the path it cannot come from substitution of P_l with $l < j$ into K_j), and we obtain a contradiction, because there is no occurrences of x'_j into *kind* of z (z lies more to the left in the context).

If the smallest path is longer, similar contradiction appears because we can show that an occurrence of x'_j must appear in the *kind* in the node that immediately precedes the end of this smallest path. ◀

The following lemma prepares the inductive step of our main theorem.

▶ **Lemma 25** (Decomposition)**.** *Let $\Gamma \vdash P : (K)K'$, $\Gamma \vdash P' : (K')K$ be as in previous lemma. Let us consider the term*

$$R \equiv [z'' : (x''_1 : K''_1) \dots (x''_n : K''_n)K''_0][x''_{\sigma_1} : K''_{\sigma_1}] \dots [x''_{\sigma_n} : K''_{\sigma_n}](z''x''_1 \dots x''_n).$$

Here R represents permutation. In particular, $K''_1 \equiv K'_1, K'' \equiv [x''_1/x'_1]K'_1, \dots, K''_n \equiv [x''_{n-1}/x'_{n-1}] \dots [x''_1/x'_1]K'_n$, $K''_0 \equiv [x''_{n-1}/x'_{n-1}] \dots [x''_1/x'_1]K'_0$

Consider also

$$P_0 \equiv [z : (x_1 : K_1) \dots (x_n : K_n)K_0][x'_1 : K'_1] \dots [x'_n : K'_n](zP_1 \dots P_n)$$

and

$$P'_0 \equiv [z' : (x'_1 : K'_1) \dots (x'_n : K'_n)K'_0][x_1 : K_1] \dots [x_n : K_n](z'P'_1 \dots P'_n).$$

Then $\Gamma \vdash R : (K'')K'$, $\Gamma \vdash P_0 : (K)K''$, $\Gamma \vdash P'_0 : (K'')K$ are derivable in LF and the following decompositions hold:

- $\Gamma \vdash P = R \circ P_0 : (K)K'$,
- $\Gamma \vdash P' \circ R = P'_0 : (K'')K$.

Proof. The derivability of all these terms relies on Lemma 22 and its Corollary. Verification of equalities uses standard reductions. For example, let us consider $\Gamma \vdash P = R \circ P_0 : (K)K'$.

Composition of two terms is defined as usual: $R \circ P_0 \equiv [z : K](R(P_0 z))$. By two β-reductions we obtain

$$[z : K][x''_{\sigma_1} : K''_{\sigma_1}] \dots [x''_{\sigma_n} : K''_{\sigma_n}]([x_1 : K_1] \dots [x_n : K_n](zP_1 \dots P_n)x''_1 \dots x''_n).$$

After that follows the series of β-reductions and renaming of bound variables ($x'' \to x'$) that gives P.

Verification for another equality is similar. ◀

Proof of the Theorem 10 (continuation).

Let $\Gamma \vdash P : (K)K'$ be an isomorphism. Then there exists its inverse $\Gamma \vdash P'(K')K$. In particular, $\Gamma \vdash P' \circ P = [z : K]z : (K)K$.

Using the Decomposition Lemma, we may write

$$\Gamma \vdash P' \circ (R \circ P_0) = (P' \circ R) \circ P_0 = P'_0 \circ P_0 = [z : K]z : (K)K.$$

Similar fact holds for $P \circ P'$.

Via two standard β-reductions we obtain

$$P'_0 \circ P_0 = [z : K][x_1 : K_1] \dots [x_n : K_n]([x'_1 : K'_1] \dots [x'_n : K'_n](zP_1 \dots P_n)P'_1 \dots P'_n).$$

Before we continue with reductions, let us see what can be established about contexts and *kinding* of $P_1, \dots, P_n$ and $P'_1, \dots, P'_n$[5].

Consider now the typing of $P_1, \dots, P_n$ and $P'_1, \dots, P'_n$. A straightforward use of properties of LF-derivations gives us:

$$\Gamma, z : K, x'_1 : K'_1, \dots, x'_n : K'_n \vdash P_1 : K_1,$$

$$\Gamma, z : K, x'_1 : K'_1, \dots, x'_n : K'_n \vdash P_2 : [P_1/x_1]K_2,$$

$$\dots$$

$$\Gamma, z : K, x'_1 : K'_1, \dots, x'_n : K'_n \vdash P_n : [P_{n-1}/x_{n-1}](\dots [P_1/x_1]K_n),$$

respectively,

$$\Gamma, z' : K', x_1 : K_1, \dots, x_n : K_n \vdash P'_1 : K'_1,$$

$$\Gamma, z' : K', x_1 : K_1, \dots, x_n : K_n \vdash P'_2 : [P'_1/x_1]K'_2,$$

$$\dots$$

$$\Gamma, z' : K', x_1 : K_1, \dots, x_n : K_n \vdash P'_n : [P'_{n-1}/x_{n-1}](\dots [P'_1/x_1]K'_n).$$

The derivability of these judgements is obtained using the known properties of LF-derivations (see [12, 14]).

Using Corollary 19, Lemma 24, and then applying Lemma 1 (strengthening), we can make the contexts considerably smaller:

$$\Gamma, x'_1 : K'_1 \vdash P_1 : K_1,$$

$$\Gamma, x'_1 : K'_1, x'_2 : K'_2 \vdash P_2 : [P_1/x_1]K_2,$$

$$\dots$$

$$\Gamma, x'_1 : K'_1, \dots, x'_n : K'_n \vdash P_n : [P_{n-1}/x_{n-1}](\dots [P_1/x_1]K_n),$$

respectively,

$$\Gamma, x_1 : K_1 \vdash P'_1 : K'_1,$$

$$\Gamma, x_1 : K_1, x_2 : K_2 \vdash P'_2 : [P'_1/x_1]K'_2,$$

$$\dots$$

$$\Gamma, x_1 : K_1, \dots, x_n : K_n \vdash P'_n : [P'_{n-1}/x_{n-1}](\dots [P'_1/x_1]K'_n).$$

[5] This is important, because as the Example 8 shows x'_1 may very well occur into $P_2, \dots, P_n$, x'_2 occur into $P_3, \dots, P_n$, *etc.*

It is easily verified that the assumption that P_0 and P'_0 are mutually inverse implies that P_j and P'_j are mutually inverse (in the above contexts). Notice that the rank is not changed by substitution (Remark 12), so the ranks of *kinds* of $[x'_1 : K'_1]P_1, \ldots, [x'_n : K'_n]P_n$, $[x_1 : K_1]P'_1, \ldots, [x_n : K_n]P'_n$ are strictly smaller than the ranks of $(K)K'$ and $(K')K$. We may apply inductive hypothesis to the pairs $P_1, P'_1, \ldots, P_n, P'_n$[6]. So, the number of the isomorphisms of the form P_0, P'_0 is finite and we may list them using isomorphisms corresponding to kinds of smaller rank, obtained by inductive hypothesis.

To pass to the general case, we have to include permutations represented by R. Their number is also finite. The upper bound is given by the number of permutations on $\{1, \ldots, n\}$ and actual number may be less due to the constraints imposed by the relation $\lhd *$ of variable dependency. Of course they all can be listed constructively, and so all the isomorphisms for a given K may be listed. ◀

▶ **Corollary 26.** *The relation of isomorphism of kinds in LF is decidable.*

Proof. An algorithm (not very efficient) works as follows. Let $\Gamma \vdash K\mathbf{kind}, \Gamma \vdash K'\mathbf{kind}$. Using main theorem, we create the list of all *kinds* that are isomorphic to K and verify whether any of them is equal to K' (*e.g.*, reducing to normal form). ◀

▶ Remark 27. When $P : (K)K'$ is an isomorphism, the $rank(K)$ permits to obtain an upper bound to the depth of the Böhm's tree of the f.h.p. $e(P)$ and this in its turn may be used to obtain an upper bound on the number of isomorphisms in the theorem.

4 Conclusion

The "core system" that we studied, Z. Luo's LF with variables, *kind* **Type** and dependent product (and definitional equality) is relatively limited, but the limited character of this system permits to obtain a complete deciding algorithm for isomorphism relation between *kinds* in spite of the fact that local (or non-contextual) rewriting does not work. The limited character of this system permits also to describe completely the structure of isomorphisms. Whether the term P is an isomorphism turns out to be decidable as well.

The restrictions on isomorphisms imposed by type-dependencies allow more (not less) "fine-tuning" than in the case of simply-typed λ-calculus. Isomorphisms of kinds to themselves are called *automorphisms*. We have a sketch of a proof (work in progress) that every finite group is isomorphic to the group of automorphisms of some kind in LF. The groups of automorphisms of simple types correspond to automorphisms of finite trees. An arbitrary finite group can not be represented in this way.

The main use of LF is to specify other type theories. To do this, LF is extended by new constants, rules for these constants, *etc.* For example the Second Order Logic SOL and Universal Type Theory UTT are defined in [11]. There are other possibilities to build type theories using LF, *e.g.*, on may add new equality rules, like the analog of the axiom C from [10]. Another modification of equality (for the whole UTT) was studied in [13].

The isomorphisms in LF described above will remain isomorphisms in these type theories but, of course, other isomorphisms may appear. The study of these isomorphisms remains an open problem.

[6] In this order, because the isomorphisms obtained by inductive hypothesis are substituted into *kinds* more to the right.

We did not yet study (and try to improve) the complexity of decision procedures for isomorphism relation between *kinds* and for the property of a term P to be an isomorphism.

All this is left for study in the near future.

Acknowledgements. The author would like to express his thanks to the organizers of the Univalent Foundations Year at IAS, Princeton (September 2012 – August 2013), the organizers of the Trimester on Semantics of Proofs and Certified Mathematics at the Institute Henri Poincaré (Paris, April–July 2014) who partly supported his participation in these research programs and created excellent work environment, and to anonymous referees for their valuable remarks.

References

1 K. Bruce, R. Di Cosmo, and G. Longo. Provable isomorphisms of types. Math. Str. in Comp. Science, 2(2):231–247, 1992.
2 D. Chemouil.Isomorphisms of simple inductive types through extensional rewriting. Math. Str. in Comp. Science, 15(5):875–915, 2005.
3 R. Di Cosmo. Isomorphisms of types: from lambda-calculus to information retrieval and language design. Birkhäuser, 1995.
4 The Coq Reference Manual, `http://coq.inria.fr/coq/refman/`.
5 D. Delahaye. Information Retrieval in a Coq Proof Library Using Type Isomorphisms. TYPES 1999: 131-147
6 M. Fiore. Isomorphisms of generic recursive polynomial types. In: POPL'04, pp. 77–88, New York, NY, USA, 2004. ACM
7 M. Fiore, R. Di Cosmo and V. Balat. Remarks on isomorphisms in typed lambda calculi with empty and sum types. Annals of Pure and Applied Logic, 141(1):35–50, 2006.
8 H. Goguen. A Typed Operational Semantics for Type Theory. PhD thesis, University of Edinburgh, 1994.
9 The HoTT Book. Homotopy Type Theory: Univalent Foundations of Mathematics. IAS Princeton, 2013.
10 G. Longo, K. Milsted and S. Soloviev. The Genericity Theorem and effective Parametricity in Polymorphic lambda-calculus. Theor. Comp. Science, 121(1993), pp. 323–349.
11 Z. Luo. Computation and Reasoning. A Type Theory for Computer Science. Oxford, 1994.
12 Z. Luo, S. Soloviev, T. Xue. Coercive subtyping: Theory and implementation. Informaion and Computation, 223 (2013), pp. 18–42.
13 L. Marie-Magdeleine. Sous-typage coercitif en présence de réductions non-standards dans un système aux types dépendants. Thèse de doctorat. Université Toulouse 3 Paul Sabatier, 2009.
14 S. Soloviev, Z. Luo. Coercion completion and conservativity in coercive subtyping. Annals of Pure and Applied Logic, 113(1–3):297–322, 2002.
15 S. V. Solov'ev. The category of finite sets and cartesian closed categories. J. of Soviet Mathematics, 22(3):1387–1400, 1983.
16 C. Stirling. Proof Systems for Retracts in Simply Typed Lambda Calculus. ICALP (2) 2013, pp. 398–409.