Simply Typed Lambda-Calculus Modulo Type Isomorphisms *

Alejandro Díaz-Caro[1,2] and Gilles Dowek[2]

[1] Université Paris-Ouest Nanterre La Défense, 200 avenue de la République, 92001 Nanterre, France
[2] INRIA, 23 avenue d'Italie, CS 81321, 75214 Paris Cedex 13, France

In informal mathematics, isomorphic structures are often identified. For instance, the natural numbers and non negative integers are never distinguished, although they formally are different structures.

In typed lambda-calculus, in programming languages, and in proof theory, two types A and B are said to be isomorphic, when there exists two functions ϕ from A to B and ψ from B to A such that $\psi\phi\mathbf{r} = \mathbf{r}$ for all terms $\mathbf{r}$ of type A and $\phi\psi\mathbf{s} = \mathbf{s}$ for all terms $\mathbf{s}$ of type B.

In some cases, isomorphic types are identified. For instance, in Martin-Löf's type theory [16], in the Calculus of Constructions [8], and in Deduction modulo [14,15], definitionally equivalent types are identified. For example, the types $x \subseteq y$, $x \in \mathcal{P}(y)$ and $\forall z\ (z \in x \Rightarrow z \in y)$ may be identified. However, definitional equality does not handle all the isomorphisms and, for example, the isomorphic types $A \wedge B$ and $B \wedge A$ are not usually identified: a term of type $A \wedge B$ does not have type $B \wedge A$.

Not identifying such types has many drawbacks, for instance if a library contains a proof of $B \wedge A$, a request on a proof of $A \wedge B$ fails to find it [18], if $\mathbf{r}$ and $\mathbf{s}$ are proofs of $(A \wedge B) \Rightarrow C$ and $B \wedge A$ respectively, it is not possible to apply $\mathbf{r}$ to $\mathbf{s}$ to get a proof of C, but we need to explicitly apply a function of type $(B \wedge A) \Rightarrow (A \wedge B)$ to $\mathbf{s}$ before we can apply $\mathbf{r}$ to this term. This has lead to several projects aiming at identifying in a way or another isomorphic types in type theory, for instance with the univalence axiom [4]. Identifying types also leads, as we shall see, to interesting calculi.

In [6], Bruce, Di Cosmo and Longo have provided a characterisation of isomorphic types in the simply typed λ-calculus extended with products and a unit type. In this work, we fully defined a simply typed λ-calculus extended with products, where all the isomorphic types are identified, and we provide a proof of strong normalisation for it. All the isomorphisms in such a setting, can be summarised to the following four:

$$A \wedge B \equiv B \wedge A \qquad (1) \qquad\qquad A \Rightarrow (B \wedge C) \equiv (A \Rightarrow B) \wedge (A \Rightarrow C) \qquad (3)$$

$$A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C \qquad (2) \qquad\qquad (A \wedge B) \Rightarrow C \equiv A \Rightarrow B \Rightarrow C \qquad (4)$$

Any other isomorphisms can be obtained by a combination of the previous four. For example, $A \Rightarrow B \Rightarrow C \equiv B \Rightarrow A \Rightarrow C$ is a consequence of isomorphisms (4) and (1).

Identifying types requires to also identify terms. For example, if $\langle \mathbf{r}, \mathbf{s} \rangle$ has type $A \wedge B = B \wedge A$, then it is not clear what the first projection would be. A more elaborated example, if $\mathbf{r}$ is a closed term of type A, then $\lambda x^A.x$ is a term of type $A \Rightarrow A$, and $\langle \lambda x^A.x, \lambda x^A.x \rangle$ is a term of type $(A \Rightarrow A) \wedge (A \Rightarrow A)$, hence, by isomorphism (3), a term of type $A \Rightarrow (A \wedge A)$. Thus the term $\langle \lambda x^A.x, \lambda x^A.x \rangle \mathbf{r}$ is a term of type $A \wedge A$. Although this term contains no redex, we do not want to consider it as normal, in particular because it is not an introduction. So we shall distribute the application over the comma, yielding the term $\langle (\lambda x^A.x)\mathbf{r}, (\lambda x^A.x)\mathbf{r} \rangle$ that finally reduces to $\langle \mathbf{r}, \mathbf{r} \rangle$. Similar considerations lead to introduction of several equivalence rules on terms.

*Full version of this paper available at http://www.diaz-caro.info/stmti.pdf (submitted)

One of the main difficulties in the design of this calculus is the design of the elimination rule for the conjunction. A rule like "if $\mathbf{r} : A \wedge B$ then $\pi_1(\mathbf{r}) : A$", would not be consistent. Indeed, if A and B are two arbitrary types, $\mathbf{s}$ a term of type A and $\mathbf{t}$ a term of type B, then $\langle \mathbf{s}, \mathbf{t} \rangle$ has both types $A \wedge B$ and $B \wedge A$, thus $\pi_1\langle \mathbf{s}, \mathbf{t} \rangle$ would have both type A and type B. The approach we have followed is to consider explicitly typed (Church style) terms, and parametrise the projection by the type: if $\mathbf{r} : A \wedge B$ then $\pi_A(\mathbf{r}) : A$ and the reduction rule is then that $\pi_A\langle \mathbf{s}, \mathbf{t} \rangle$ reduces to $\mathbf{s}$ if $\mathbf{s}$ has type A.

But this rule introduces some non-determinism. Indeed, in the particular case where A happens to be equal to B, then both $\mathbf{s}$ and $\mathbf{t}$ have type A and $\pi_A\langle \mathbf{s}, \mathbf{t} \rangle$ reduces both to $\mathbf{s}$ and to $\mathbf{t}$. Notice that although this reduction rule is non-deterministic, it preserves typing.

Thus, our calculus is one of the many non-deterministic calculi in the line of [5,7,9,10,12,17]. Our pair-construction operator is like the parallel composition operator in a non deterministic calculi. In non-deterministic calculi, the parallel composition is such that if $\mathbf{r}$ and $\mathbf{s}$ are two λ-terms, the term $\langle \mathbf{r}, \mathbf{s} \rangle$ represents the computation that runs either $\mathbf{r}$ or $\mathbf{s}$ non-deterministically, that is such that $\langle \mathbf{r}, \mathbf{s} \rangle \mathbf{t}$ reduces either to $\mathbf{rt}$ or $\mathbf{st}$. In our case, $\pi_B(\langle \mathbf{r}, \mathbf{s} \rangle \mathbf{t})$ first reduces to $\pi_{A \Rightarrow B}\langle \mathbf{r}, \mathbf{s} \rangle \mathbf{t}$ and then to $\mathbf{rt}$ or $\mathbf{st}$.

In [11] we showed that this calculus is also related to the algebraic calculi [1–3,13,19], some of which have been designed to express quantum algorithms. In this case, the pair $\langle \mathbf{s}, \mathbf{t} \rangle$ is not interpreted as a non deterministic choice but as a superposition of two processes running $\mathbf{s}$ and $\mathbf{t}$, and the projection π is the related to the projective measurement, that is the only non deterministic operation.

References

[1] P. Arrighi and A. Díaz-Caro. A System F accounting for scalars. *LMCS*, 8(1:11), 2012.

[2] P. Arrighi, A. Díaz-Caro, and B. Valiron. A type system for the vectorial aspects of the linear-algebraic lambda-calculus. *EPTCS*, 88:1–15, 2012.

[3] P. Arrighi and G. Dowek. Linear-algebraic lambda-calculus: higher-order, encodings, and confluence. *LNCS*, 5117:17–31, 2008.

[4] S. Awodey, A. Pelayo, and M. A. Warren. Voevodsky's univalence axiom in homotopy type theory. *Notices of the AMS*, 60(08):1164–1167, 2013.

[5] G. Boudol. Lambda-calculi for (strict) parallel functions. *IC*, 108(1):51–127, 1994.

[6] K. B. Bruce, R. Di Cosmo, and G. Longo. Provable isomorphisms of types. *MSCS*, 2(2):231–247, 1992.

[7] A. Bucciarelli, T. Ehrhard, and G. Manzonetto. A relational semantics for parallelism and non-determinism in a functional setting. *Annals of Pure and Applied Logic*, 163(7):918–934, 2012.

[8] T. Coquand and G. Huet. The calculus of constructions. *IC*, 76(2–3):95–120, 1988.

[9] U. de'Liguoro and A. Piperno. Non deterministic extensions of untyped λ-calculus. *IC*, 122(2):149–177, 1995.

[10] M. Dezani-Ciancaglini, U. de'Liguoro, and A. Piperno. A filter model for concurrent lambda-calculus. *SIAM Journal of Computing*, 27(5):1376–1419, 1998.

[11] A. Díaz-Caro and G. Dowek. The probability of non-confluent systems. *EPTCS*, 143:1–15, 2013.

[12] A. Díaz-Caro, G. Manzonetto, and M. Pagani. Call-by-value non-determinism in a linear logic type discipline. *LNCS*, 7734:164–178, 2013.

[13] A. Díaz-Caro and B. Petit. Linearity in the non-deterministic call-by-value setting. *LNCS*, 7456:216–231, 2012.

[14] G. Dowek, T. Hardin, and C. Kirchner. Theorem proving modulo. *Journal of Automated Reasoning*, 31(1):33–72, 2003.

[15] G. Dowek and B. Werner. Proof normalization modulo. *The Journal of Symbolic Logic*, 68(4):1289–1316, 2003.

[16] P. Martin-Löf. *Intuitionistic type theory.* Studies in proof theory. Bibliopolis, 1984.

[17] M. Pagani and S. Ronchi della Rocca. Linearity, non-determinism and solvability. *Fundamenta Informaticae*, 103(1-4):173–202, 2010.

[18] M. Rittri. Retrieving library identifiers via equational matching of types. *LNCS*, 449:603–617, 1990.

[19] L. Vaux. The algebraic lambda calculus. *MSCS*, 19(5):1029–1059, 2009.