The Yoneda Reduction of Polymorphic Types (Abstract)* **

Paolo Pistone[1] and Luca Tranchini[2]

[1] DISI, Università di Bologna, Mura Anteo Zamboni,7, I-40126, Bologna
[2] Wilhelm-Schickard-Institut, Universität Tübingen, Sand 13, D-72076, Tübingen
paolo.pistone2@unibo.it
luca.tranchini@gmail.com

Abstract. We explore a family of type isomorphisms in System F whose validity corresponds, semantically, to some form of the Yoneda isomorphism from category theory. These isomorphisms hold under theories of equivalence stronger than $\beta\eta$-equivalence, like those induced by parametricity and dinaturality. We show that the Yoneda type isomorphisms yield a rewriting over types, that we call Yoneda reduction, which can be used to eliminate quantifiers from a polymorphic type, replacing them with a combination of monomorphic type constructors. We establish some sufficient conditions under which quantifiers can be fully eliminated from a polymorphic type, and we show some application of these conditions to count the inhabitants of a type and to compute program equivalence in some fragments of System F.

Keywords: Type isomorphism · Yoneda Lemma · Propositional quantification.

1 Introduction

The study of type isomorphisms is a fundamental one both in the theory of programming languages and in logic, through the well-known *proofs-as-programs* correspondence: type isomorphisms supply programmers with transformations allowing them to obtain simpler and more optimized code, and offer new insights to understand and refine the syntax of type- and proof-systems.

* This extended abstract is an excerpt from the long version of a paper which the authors have recently published in the proceedings of CSL 21 [21]. The long version of the paper (available at `https://arxiv.org/abs/1907.03481`) contains more detailed proofs and some additional results compared to [21].

** We thank the anonymous reviewers for interesting insights, and in particular for pointing us to a similarity between the type isomorphisms studied in this paper and a result known as Ackermann's Lemma in the field of second-order quantifier elimination (see [11], section 6).

Roughly speaking, two types A, B are isomorphic when one can transform any call by a program to an object of type A into a call to an object of type B, without altering the behavior of the program. Thus, type isomorphisms are tightly related to theories of *program equivalence*, which describe what counts as the observable behavior of a program, so that programs with the same behavior can be considered equivalent.

The connection between type isomorphisms and program equivalence is of special importance for polymorphic type systems like System F (hereafter $\Lambda 2$). In fact, while standard $\beta\eta$-equivalence for $\Lambda 2$ and the related isomorphisms are well-understood [7,8], stronger notions of equivalence (as those based on *parametricity* or *free theorems* [29,16,1]) are often more useful in practice but are generally intractable or difficult to compute, and little is known about the type isomorphisms holding under such theories.

A cornerstone result of category theory, the *Yoneda lemma*, is sometimes invoked [3,13,5,27,14] to justify some type isomorphisms in $\Lambda 2$ like e.g.

$$\forall X.(A \Rightarrow X) \Rightarrow (B \Rightarrow X) \equiv B \Rightarrow A \qquad \forall X.(X \Rightarrow A) \Rightarrow (X \Rightarrow B) \equiv A \Rightarrow B \tag{$\star$}$$

which do not hold under $\beta\eta$-equivalence, but only under stronger equivalences. Such isomorphisms are usually justified by reference to the interpretation of polymorphic programs as *(di)natural transformations* [3], a well-known semantics of $\Lambda 2$ related to both parametricity [23] and free-theorems [28], and yielding a not yet well-understood equational theory over the programs of $\Lambda 2$ [12,17,20], that we call here the *ε-theory*. Other isomorphisms, like those in Fig. 1, can be justified in a similar way as soon as the language of $\Lambda 2$ is enriched with other type constructors like $1, 0, +, \times, \Rightarrow$ and *least/greatest fixpoints* $\mu X.A, \nu X.A$.

All such type isomorphisms have the effect of *eliminating* a quantifier, replacing it with a combination of monomorphic type constructors, and can be used to test if a polymorphic type has a *finite* number of inhabitants (as illustrated in Fig. 2) or, as suggested in [5], to devise *decidable tests* for program equivalence.

In this paper we develop a formal study of the elimination of quantifiers from polymorphic types using a class of type isomorphisms, that we will call *Yoneda type isomorphisms*, which generalize the examples above. Then, we explore the application of such type isomorphisms to establish properties of program equivalence for polymorphic programs.

2 A Type-Rewriting Theory of Polymorphic Types

In the first part of the paper we investigate the type-rewriting arising from Yoneda type isomorphisms and its connection with proof-theoretic techniques to count type inhabitants.

2.1 Counting type inhabitants with type isomorphisms.

Examples like the one in Fig. 2 suggest that, while arising from a categorical reading of polymorphic programs, Yoneda Type Isomorphisms have a proof-theoretic

$$\forall X.X \Rightarrow X \Rightarrow A \equiv A[X \mapsto 1+1] \qquad (*)$$
$$\forall X.(A \Rightarrow X) \Rightarrow (B \Rightarrow X) \Rightarrow C \equiv C[X \mapsto \mu X.A+B] \qquad (**)$$
$$\forall X.(X \Rightarrow A) \Rightarrow (X \Rightarrow B) \Rightarrow D \equiv D[X \mapsto \nu X.A \times B] \qquad (***)$$

Fig. 1: Other examples of Yoneda type isomorphisms, where X only occurs positively in A, B, C and only occurs negatively in D.

flavor. To demonstrate this, we show that the use of such isomorphisms to count type inhabitants can be compared with some well-known proof-theoretic *sufficient conditions* for a *simple* type A to have a unique or finitely many inhabitants [2,6], namely the *balancedness*, *negatively-non duplicated* and *positively-non duplicated* conditions. We show that whenever an inhabited simple type A satisfies any of the first two conditions (resp. a special case of the third), then its universal closure $\forall X_1 \dots \forall X_n.A$ can be converted to 1 by applying Yoneda type isomorphisms and usual $\beta\eta$-isomorphisms, and when A satisfies a special case of the third, then it can be converted to $1 + \dots + 1$. Furthermore, we suggest that the appearance of the fixpoints in isomorphisms like $(**)$ can be related to a well-known approach to counting type inhabitants by solving systems of *polynomial fixpoint equations* [30].

2.2 Eliminating Quantifiers with Yoneda Reduction

We then turn to investigate in a more systematic way the quantifier-eliminating rewriting over types arising from the left-to-right orientation of Yoneda type isomorphisms. A major obstacle here is that the rewriting must take into account possible applications of $\beta\eta$-isomorphisms, whose axiomatization is well-known to be challenging in presence of the constructors $+, 0$ [10,15] (not to say of μ, ν). For this reason we introduce a family of rewrite rules, that we call *Yoneda reduction*, defined not directly over types but over a class of finite trees which represent the types of $\mathsf{\Lambda 2}$ (but crucially *not* those made with $0, +, \dots$) up to $\beta\eta$-isomorphism.

Using this rewriting we establish some sufficient conditions for eliminating quantifiers, based on elementary graph-theoretic properties of such trees, which in turn provide some *new* sufficient conditions for the finiteness of type inhabitants of polymorphic types. First, we prove quantifier-elimination for the types satisfying a certain *coherence* condition which can be seen as an instance of the 2-SAT problem. We then introduce a more refined condition by associating each polymorphic type A with a value $\kappa(A) \in \{0, 1, \infty\}$, that we call the *characteristic of* A, so that whenever $\kappa(A) \neq \infty$, A rewrites into a monomorphic type, and when furthermore $\kappa(A) = 0$, A converges to a finite type. In the last case our method provides an effective way to count the inhabitants of A.

The computation of $\kappa(A)$ is somehow reminiscent of linear logic *proof-nets*, as it is obtained by inspecting the existence of cyclic paths in a graph obtained by adding some "axiom-links" to the tree-representation of A.

$$\forall X.X \Rightarrow X \Rightarrow \forall Y.(\forall Z.(Z \Rightarrow X) \Rightarrow (\forall W.(W \Rightarrow Z) \Rightarrow W \Rightarrow X) \Rightarrow Z \Rightarrow Y) \Rightarrow (X \Rightarrow Y) \Rightarrow Y$$
$$\overset{(*)}{\equiv} \forall Y.(\forall Z.(Z \Rightarrow 1+1) \Rightarrow (\forall W.(W \Rightarrow Z) \Rightarrow W \Rightarrow 1+1) \Rightarrow Z \Rightarrow Y) \Rightarrow (1+1 \Rightarrow Y) \Rightarrow Y$$
$$\overset{(\star)}{\equiv} \forall Y.(\forall Z.(Z \Rightarrow 1+1) \Rightarrow (Z \Rightarrow 1+1) \Rightarrow Z \Rightarrow Y) \Rightarrow (1+1 \Rightarrow Y) \Rightarrow Y$$
$$\overset{(***)}{\equiv} \forall Y.((\nu Z.(1+1) \times (1+1)) \Rightarrow Y) \Rightarrow (1+1 \Rightarrow Y) \Rightarrow Y$$
$$\overset{(**)}{\equiv} \mu Y.(\nu Z.(1+1) \times (1+1)) + (1+1) \equiv 1+1+1+1+1+1$$

Fig. 2: Short proof that a $\Lambda 2$-type has 6 inhabitants, using type isomorphisms.

3 Program Equivalence in System F with Finite Characteristic

In the second part of the paper we direct our attention to programs rather than types, and we exploit our results on type isomorphisms to establish some non-trivial properties of program equivalence for polymorphic programs in some suitable fragments of $\Lambda 2$.

3.1 Computing equivalence with type isomorphisms

Computing program equivalence under the ε-theory can be a challenging task, as this theory involves global permutations of rules which are difficult to detect and apply [17,20,26,22,28]. Things are even worse at the semantic level, since computing with dinatural transformations can be rather cumbersome, due to the well-known fact that such transformations need not compose [3,18].

Nevertheless, our approach to quantifier-elimination based on the notion of characteristic provides a way to compute program equivalence *without* the appeal to ε-rules, free theorems and parametricity, since all polymorphic programs having types of finite characteristic can be embedded inside well-known monomorphic systems. To demonstrate this fact, we introduce two fragments $\Lambda 2^{\kappa \leqslant 0}$ and $\Lambda 2^{\kappa \leqslant 1}$ of $\Lambda 2$ in which types have a fixed finite characteristic, and we prove that these are equivalent, under the ε-theory, respectively, to the simply typed λ-calculus with finite products and co-products (or, equivalently, to the *free bicartesian closed category* $\mathbb{B}$), and to its extension with μ, ν-types (that is, to the *free cartesian closed μ-bicomplete category* $\mu\mathbb{B}$ [24,4]). Using well-known facts about $\mathbb{B}$ and $\mu\mathbb{B}$ [25,4,19], we finally establish that the ε-theory is decidable in $\Lambda 2^{\kappa \leqslant 0}$ and undecidable in $\Lambda 2^{\kappa \leqslant 1}$.

3.2 Program equivalence and predicativity

We provide an example of how the correspondence between polymorphic types of finite characteristic and μ, ν-types can be used to prove non-trivial properties of program equivalence. A main source of difficulty with $\Lambda 2$ is that polymorphic programs are *impredicative*, that is, a program of universal type $\forall X.A$ can be instantiated at *any* type B, yielding a program of type $A[B/X]$. It is thus useful to be able to predict when a complex type instantiation can be replaced by a one of smaller complexity, without altering the program behavior.

Using the fact that a universal type $\forall X.A$ of finite characteristic in which A is of the form $A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow X$ is isomorphic to the *initial algebra* $\mu X.T$ of some appropriate functor T, we establish a sufficient condition under which a program containing an instantiation of $\forall X.A$ as $A[B/X]$ can be transformed into one with instantiations of types strictly less complex than B.

We finally use this condition to provide a simpler proof of a result from [22], showing that all programs in a certain fragment of $\Lambda 2^{\kappa \leqslant 0}$ (the fragment freely generated by the embedding of finite sums and products) can be transformed into predicative programs only containing *atomic* type instantiations, a result related to some recent investigations on atomic polymorphism [9].

References

1. Ahmed, A., Jamner, D., Siek, J.G., Wadler, P.: Theorems for free for free: parametricity, with and without types. In: Proceedings of the ACM on Programming Languages. ICFP, vol. 1, p. Article No. 39. New York (2017)
2. Aoto, T.: Uniqueness of normal proofs in implicational intuitionistic logic. Journal of Logic, Language and Information **8**, 217–242 (1999)
3. Bainbridge, E., Freyd, P.J., Scedrov, A., Scott, P.J.: Functorial polymorphism. Theoretical Computer Science **70**, 35–64 (1990)
4. Basold, H., Hansen, H.H.: Well-definedness and observational equivalence for inductive-coinductive programs. Journal of Logic and Computation **exw091** (2016)
5. Bernardy, J.P., Jansson, P., Claessen, K.: Testing Polymorphic Properties. In: ESOP 2010: Programming Languages and Systems. Lecture Notes in Computer Science, vol. 6012, pp. 125–144. Springer Berlin Heidelberg (2010)
6. Broda, S., Damas, L.: On long normal inhabitants of a type. Journal of Logic and Computation **15**, 353–390 (2005)
7. Bruce, K.B., Di Cosmo, R., Longo, G.: Provable isomorphisms of types. Mathematical Structures in Computer Science **1**, 1–20 (1991)
8. Di Cosmo, R.: A short survey of isomorphisms of types. Mathematical Structures in Computer Science **15**, 825–838 (2005)
9. Ferreira, F., Ferreira, G.: Atomic polymorphism. Journal of Symbolic Logic **78**(1), 260–274 (2013)
10. Fiore, Di Cosmo, R., Balat, V.: Remarks on isomorphisms in typed lambda calculi with empty and sum types. Annals of Pure and Applied Logic **141**(1–2), 35–50 (2006)
11. Gabbay, D.M., Schmidt, R., Szalas, A.: Second-order Quantifier Elimination. College publications (2008)
12. Girard, J.Y., Scedrov, A., Scott, P.J.: Normal forms and cut-free proofs as natural transformations. In: Moschovakis, Y. (ed.) Logic from Computer Science, vol. 21, pp. 217–241. Springer-Verlag (1992)
13. Hasegawa, R.: Categorical data types in parametric polymorphism. Mathematical Structures in Computer Science **4**(1), 71–109 (1994)
14. Hinze, R., James, D.W.: Reason isomorphically! In: WGP 2010. pp. 85–96 (2010)
15. Ilik, D.: Axioms and decidability for type isomorphism in the presence of sums. In: Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (2014)

16. Johann, P.: On proving the correctness of program transformation based on free theorems for higher-order polymorphic calculi. Mathematical Structures in Computer Science **15**, 201–229 (2005)
17. de Lataillade, J.: Dinatural terms in System F. In: Proceedings of the Twenty-Fourth Annual IEEE Symposium on Logic in Computer Science (LICS 2009). pp. 267–276. IEEE Computer Society Press, Los Angeles, California, USA (2009)
18. McCuscker, G., Santamaria, A.: On compositionality of dinatural transformations. In: CSL 2018. LIPIcs, vol. 119, pp. 33:1–33:22. Schloss Daghstuhl–Leibinz-Zentrum fuer Informatik, Dagstuhl, Germany (2018)
19. Okada, M., Scott, P.: A note on rewriting theory for uniqueness of iteration. Theory and Applications of Categories **6**, 47–64 (1999)
20. Pistone, P.: On dinaturality, typability and $\beta\eta$-stable models. In: fuer Informatik, S.D.L.Z. (ed.) 2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 84, pp. 29:1–29:17. Dagstuhl, Germany (2017)
21. Pistone, P., Tranchini, L.: The Yoneda Reduction of Polymorphic Types. In: Computer Science Logic 2021 (CSL 2021). LIPIcs–Leibniz International Proceedings in Informatics, vol. 183, pp. 35:1–35:22 (2021)
22. Pistone, P., Tranchini, L., Petrolo, M.: The naturality of natural deduction (II). On atomic polymorphism and generalized propositional connectives. Studia Logica (2021). https://doi.org/10.1007/s11225-021-09964-z
23. Plotkin, G., Abadi, M.: A logic for parametric polymorphism. In: TLCA '93, International Conference on Typed Lambda Calculi and Applications. Lecture Notes in Computer Science, vol. 664, pp. 361–375. Springer Berlin Heidelberg (1993)
24. Santocanale, L.: Free μ-lattices. Journal of Pure and Applied Algebra **9**, 166–197 (2002)
25. Scherer, G.: Deciding equivalence with sums and the empty type. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages. pp. 374–386. POPL 2017, ACM, New York, NY, USA (2017)
26. Tranchini, L., Pistone, P., Petrolo, M.: The naturality of natural deduction. Studia Logica **107**(1), 195–231 (2019)
27. Uustalu, T., Vene, V.: The Recursion Scheme from the Cofree Recursive Comonad. Electronic Notes in Theoretical Computer Science **229**(5), 135–157 (2011)
28. Voigtländer, J.: Free theorems simply, via dinaturality. In: Declarative Programming and Knowledge Management. pp. 247–267. Springer International Publishing, Cham (2020)
29. Wadler, P.: Theorems for free! In: Proceedings of the fourth international conference on functional programming languages and computer architecture - FPCA '89 (1989)
30. Zaoinc, M.: Fixpoint technique for counting terms in typed lambda-calculus. Tech. rep., State University of New York (1995)