Research Statement

Carlos C. Martínez. Versión (2006)

"If you keep proving stuff that others have done, getting confidence, increasing the complexities of your solutions - for the fun of it - then one day you'll turn around and discover that nobody actually did that one! And that's the way to become a computer scientist." — Richard Feynmann

Synopsis

My research concerns the theoretical and algorithmic aspects of *higher order unification and matching problems* in typed lambda calculi *modulo type isomorphisms*. This work has been motivated by the intricacies of trying to automate program transformations viewed as instances of higher order matching problems. My goal has been to find justifications to perform these transformations in a broader and yet meaningful context.

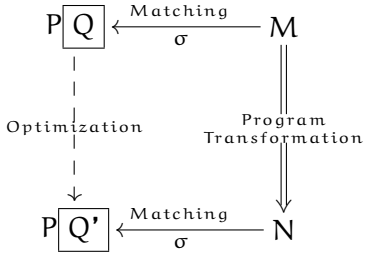
Background

The problem of automating higher order logic was first addressed by Robinson [Rob69]. The complexity of the problem become evident when higher order unification was shown to be undecidable in 1972, independently by Huet [Hue72, Hue73] and Lucchesi [Luc72], a result later sharpened by Goldfarb [Gol81].

(Higher Order Unification) Let M *and* N *be lambda-terms. We say that* M *and* N *are* unifiable *if and only if there exist a substitution* σ *such that* $\sigma(\mathrm{M}) = \sigma(\mathrm{N})$.

Huet[73] proposed to solve a restriction of the general unification problem. His approach consisted in an enumeration of *pre-unifiers*, this is; delaying and assuming solved *flexible-flexible* equations (These are equations where both head-terms are free variables, usually having infinite solutions), and gave a complete set of reduction-transformations for the remaining cases. An other important case of unification is when one of the terms has no free variables, this is known as the *matching problem*, it has shown to be sufficiently powerful for symbolic computation. In this setting, Huet's semi decision algorithm turns out to be decidable, generating a finite set of solutions in the case of second order terms. However, the general matching problem for the full higher order hierarchy has remained open after 30 years of investigation. Further restrictions of the unification problem have shown to be decidable as well, it was possible to combine pre-unification with first order unification, as in Miller [Mil91], where the decidability of higher order pattern unification was also shown.

(Program Transformation Schemes.) We said that a tuple $\mathrm{T} = \langle \mathrm{M}, \mathrm{N} \rangle$ *is a* program transformation scheme *if and only if* M, N *are two program contexts having the same free variables and types. Such program transformation* $\mathrm{M} \Rightarrow_{\mathrm{T}} \mathrm{N}$ *is* applicable *to a program* P *at* Q *if the following condition* $\mathrm{Q} = \sigma(\mathrm{M})$ *holds.*



The second order matching algorithm introduced by Huet and Lang [HL78] in the late 70's was motivated by *program transformation schemes* originally suggested by Burstall and Darlington [BD77], these transformations based on rewriting techniques are easily represented by second order instances of matching problems. The program transformation paradigm provides programmer with clearer coding style, and leads to desirable programming properties, such as; modularity, maintainability and readability, as counterpart of usual programming practices that sometime relies on obscure efficient coding. These ideas have been used by O'de Moor and Sittampalam [OS99] in the design of functionals programming languages compilers as part of an optimization pre-processing.

Avoiding paradoxes has been crucial to the development of foundations of mathematics in the past century, and one can recall two different approaches, one based on sets *(ZF)* and the other based on functions *(Type Theory)* as their primitive concepts. The latter has more relevance in our context. The concept of *types* was introduced to avoid such paradoxes, a notion which will have an essential role in computer science, and became an important tool in the design programming languages, to prove properties of programs, as well as to trace program errors.

(Type Isomorphisms) Two Types A *and* B *are isomorphic* $\mathrm{A} \simeq \mathrm{B}$ *if and only if there exist functions* $\mathrm{M} : \mathrm{A} \to \mathrm{B}$ *and* $\mathrm{N} : \mathrm{B} \to \mathrm{A}$ *such that* $\mathrm{M} \circ \mathrm{N} = \mathrm{Id}_{\mathrm{A}}$ *and* $\mathrm{N} \circ \mathrm{M} = \mathrm{Id}_{\mathrm{B}}$.

There has been a great deal interest over the years in constructing models of lambda calculi which satisfy certain equations between *(isomorphisms)* types; specifically, recursive definitions of data-types are often interpreted as equations to be solved over mathematical structures. The existence of non-trivial isomorphisms has been shown in the simply lambda calculus by Dezani [Dez76], second order lambda calculus by Bruce, Di Cosmo and Longo [BDL91] and [DiC95], and recursive type systems Fiore, Di Cosmo and Balat [FCB02]; providing an interesting interplay of the close relationship between these calculi and their relevant categorical structures, proof systems [Sol83, Sol93], and applications to retrieving software components as introduced by Rittri [Rit91],[Rit93].

The realization of type isomorphisms relies on the existence of invertible terms, so it is important to have a characterization of such terms. Dezani[Dez76] has described the *finite hereditary permutation* terms as the only invertible terms in the simply typed case. It is rather easy to see, that given two isomorphic simple types, there are a finite number of invertible terms up to α-equivalence witnessing this. Di Cosmo [DiC95] has extended Dezani's invertible terms characterization to the first order type system in the presence of products and unit types, and he has also studied a second order type system that approaches the full type system of the ML functional programming languages. These works provide us with the content to explore the scope of our ideas.

Thesis Work

As we mentioned above matching is a plausible frame for program transformations, and our aim has been to strengthen this approach by providing a more general setting. It is not hard to see that *matching* can be sometimes restrictive for program transformation purposes, since matching respect types. We have shown that this constraint rules out some instances having *mild* differences. We have proposed that it is possible to overcome these difficulties in a way the program transformation can be performed, and we have suggested that the difference in reordering programs by invertible ones is mild, thereby supporting our objective of enriching matching under isomorphism of types.

(Program Isomorphisms) Two programs $\mathrm{P} : \mathrm{A}$ *and* $\mathrm{Q} : \mathrm{B}$ *are program isomorphic* $\mathrm{P} \simeq \mathrm{Q}$ *if and only if* $\mathrm{A} \simeq \mathrm{B}$ *by an invertible program* $\mathrm{F} : \mathrm{A} \to \mathrm{B}$ *such that* $\mathrm{FP} = \mathrm{Q}$.

This definition lead us to our new matching definition.

(Matching Programs Modulo Type Isomorphism) Let P *and* Q *be programs. We say that* P *is* matchable *to* Q *if and only if there exist inputs* $\mathrm{in}_1, ..., \mathrm{in}_n$ *such that* $\mathrm{Pin}_1 ... \mathrm{in}_n \simeq \mathrm{Q}$.

Some of the most interesting work concerns polymorphic type disciplines, where the reuse of existing code becomes natural. This being our goal, at this point we can report progress in preliminary states restricting ourselves to the case of simply typed lambda calculus, where the setting of this project is well understood.

Enumeration of invertible terms leads to a naive decision procedure for unification modulo type isomorphisms, obtained by exhaustively seeking for solutions of the usual unification problems generated by the finitely many invertible terms. As we believe this approach is rather inefficient and it becomes natural to expect refinements. We have shown that it is possible to address the problem of finding an invertible terms and solving the unification problem in the usual setting of solving it by transformations. The novel idea here is to introduce new variables witnessing the invertible terms that allow us to split those tasks by adding an extra transformation to the standard set of transformations of the underling decidable restriction of the unification problem.

Ongoing Work

Can we achieve more efficiency? It has been shown by Zibin, Gil and Considine [ZGC03] that testing type isomorphism can be done efficiently in the presence of functions, products, and unit types, establishing a polynomial upper bound on the size of the input types. So we might expect improvements by combining successfully these algorithms with a matching algorithm. A first challenge has been introduced and remains to be addressed; ZGC's decision algorithm has been derived from classical bottom up tree isomorphism algorithms, and unification algorithms are usually top down reduction.

Once efficiency is achieved, it would be desirable to derive complexity results on decidable cases such as matching at low order and pattern unification; and it would also be interesting to generate an implementation to test these ideas in the context of program transformation.

Referencias

[BDL91] Bruce K., Di Cosmo R., Longo G., Provable Isomorphism of Types, *Mathematical Structures in Computer Science*, 2:231–247, 1991.

[BD77] Burstall R., Darlington, J. Some program transformations for developing recursive programs *In Proceedings International Conference on Reliable Software,* Los Angeles, 1975, and also *Journal Assoc. Comput. Mach.*, 24:44-67, 1977.

[Ch40] Church A., A formulation of the simple theory of types, *Journal of Symbolic logic*, 5:56–68, 1940.

[Dez76] Dezani-Ciancaglini M., Characterization of Normal Forms Possessing Inverse in the $\lambda_{\beta\eta}$-Calculus, *Theoretical Computer Science*, 2:323–337, 1996.

[DiC95] Di Cosmo R., Isomorphism of Types: from λ-calculus to information retrieval and language design, Birkhäuser, (1995). *Progress in Theoretical Computer Science.* Birkhauser, 1995. ISBN-0-8176-3763-X.

[DM04] Dougherty D. and Martínez C., Unification and Matching Modulo Type Isomorphism, *In Proceedings of II International Workshop of Higher Order Rewriting*, Aachen, Germany, June 2004. Abstract in Technical report of the Computer Science Department of RWTH Aachen University.

[FCB02] Fiore M., Di Cosmo R. and Balat V., Remarks on isomorphisms in typed lambda calculi with empty and sum types, In *Logic in Computer Science*, pp. 147–156, Los Alamitos, CA, USA, July 22–25 2002. IEEE Computer Society.

[Gol81] Goldfarb W., The undecidability of the second order unification problem, *Journal of Theoretical Computer Science.* 13:225–230, 1981.

[Hue72] Huet G., Constrained resolution: A complete method for higher order logic. *Ph.D. dissertation*, Case Western Reserve University, Cleveland, Ohio, 1972.

[Hue73] Huet G., The Undecidability of Unification in Third Order Logics, *Information and Control*, 22:257–267, 1973.

[HL78] Huet G. and Lang B., Proving and Applying Program Transformations Expressed with Second Order Patterns, *Acta Informatica*, 11:31–55, 1978.

[Luc72] Lucchesi C.L., the Undecidability of the Unification Problem for Third Order languages, *Report CSRR 2059, Dept of Applied Analysis and Computer Science, University Waterloo*, 1972.

[Mil91] Miller D., A logic programming language with lambda-abstraction, function variables, and simple unification, *Journal of Logic and Computation*, 1(4):497–536, 1991.

[OS99] de Moor O. and Sittampalam G., Generic Program Transformation, Proceedings of the 3rd International Summer School on Advanced Functional Programming. *Springer Lecture Notes in Computer Science*, 1608:116–149, 1999.

[Rob69] Robinson J.A., New directions in mechanical theorem proving, In A.J.H. Morell, editor *International Federation for Information Processing Congress 1968* 63–67, North Holland, 1969.

[Rit91] Rittri M., Using types as search keys in function libraries, *Journal of Functional Programming*, 1(1):71–89, 1991.

[Rit93] Rittri M., Retrieving Library Functions by Unifying Types Modulo Linear Type Isomorphism, *Theoretical Informatics and Applications*, 27:71–89, 1993.

[Sol83] Soloviev S., The Category of Finite Sets and Cartesian Closed Categories, *Journal of Soviet Mathematics*, 22(3):1387–1400, 1983.

[Sol93] Soloviev S., A Complete axiom system for Isomorphism of Types in Closed Categories, *Logic Programming and Automated Reasoning, 4th International Conference*, LNCS 698: 360–371, 1993.

[ZGC03] Zibin Y., Gil Y. and Considine J., Efficient algorithms for isomorphisms of simple types, In *Proceedings of the 30th ACM Symposium on Principles of Programming Languages (POPL 2003)*, pp. 160–171. ACM Press, 2003.