Progress in Theoretical Computer Science

Roberto Di Cosmo

Isomorphisms of Types:

from λ-calculus to information retrieval and language design

Birkhäuser 1995
Boston • Basel • Berlin

Roberto Di Cosmo
LIENS-DMI
Ecole Normale Supérieure
75230 Paris Cedex 05
France

Library of Congress Cataloging-in-Publication Data

Di Cosmo, Roberto, 1963-
Isomorphisms of types : from [lambda]-calculus to information retrieval and language design / Roberto Di Cosmo.
p. cm. -- (Progress in theoretical computer science)
Includes bibliographical references and index.
ISBN-13: 978-1-4612-7585-5
1. Programming languages (Electronic computers). 2. Type theory. 3. Human-computer interaction. I. Title. II. Series.
QA76.7.D5 1993 94-36650
005.13'1--dc20 CIP

Printed on acid-free paper

Birkhäuser ®

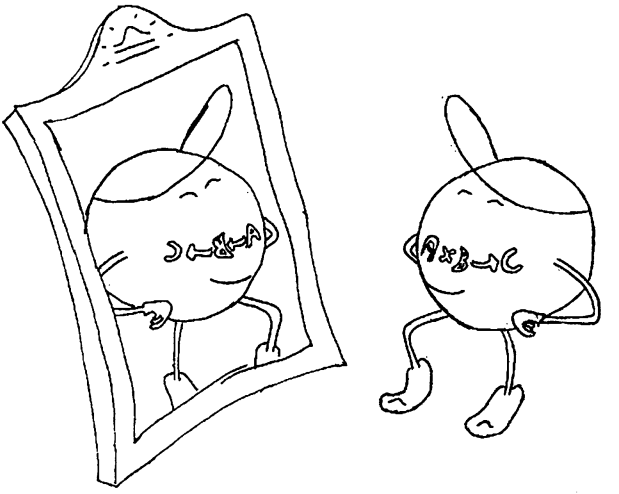
ISBN-13: 978-1-4612-7585-5 e-ISBN-13: 978-1-4612-2572-0
DOI: 10.1007/978-1-4612-2572-0

Typeset by the author.

9 8 7 6 5 4 3 2 1

Contents

Isomorphisms of Types: From λ-calculus to Information Retrieval and Language Design



Acknowledgments

This book is the outcome of several years of interaction with many persons without whom this work would not have seen the light, but in all these years some very special persons have been constantly at my side, and I would like to begin by thanking them. My parents and all my family, who know that there are no words to express my gratitude to them. I had the chance to meet Delia Kesner and to share with her the wonderful experience of building day after day a strong relationship that goes far beyond our scientific cooperation.

I'm greatly indebted to my advisor, Giuseppe Longo, for his continuous encouragement, discussions and insights: he strongly motivated me to carry on all the work that supports the results presented in this book.

I met Pierre-Louis Curien on my arrival in Paris, and I enjoyed with him the wonderful experience of the investigation of $\lambda^2\beta\eta\pi*$, the core of Chapters 2 and 3 of this book. He knows how much I owe him.

Thanks to Roger Hindley and Jerzy Tiuryn who made me the honor of being referees for this book.

I would like to express my gratitude to Mikael Rittri, whose work on library searches provided interesting practical applications to the theoretical results developed in all this work: he has been a constant and copious source of information on the ongoing work in that area. Thanks go also to Gregory Mints for pointing out to us the work of Sergei Soloviev: I met Sergei shortly after I discovered his works, and I would like to thank him for the many fruitful and interesting discussions we had since then.

I wish to thank Hubert Comon and Jean-Pierre Jouannaud, as well as all the working group of the LRI at the University of Orsay, for many stimulating discussions: they helped and pushed me in many essential phases.

The whole group working at INRIA at the CAML system has provided invaluable support in the development of the code that implements the library search system, and I'm particularly grateful to Pierre Weis, whose help was crucial in the integration of this new tool in the CAML system, and to Xavier Leroy, whose impressive programming skill led to a work-

ing experimental implementation of the new type-inference mechanism for CamlLight in less than an afternoon.

The Liens and the Dmi, at the Ecole Normale Supérieure in Paris, helped my work by providing a fantastic scientific environment and excellent facilities.

A special thank goes to Robert Constable, for hosting me in Cornell's unique environment for six unforgettable months.

Thanks finally to Franco Barbanera, Emmanuel Chailloux, Pierre Cregut, Mariangiola Dezani-Ciancaglini, Vincent Danos, Furio Honsell, Martin Hyland, Eugenio Moggi, Chetan Murthy, Paliath Narendran, Adolfo Piperno, Andrew Pitts, Laurent Regnier, Simona Ronchi-Della Rocca, Pino Rosolini, André Scedrov, Val Tannen and many others for several interesting discussions on all these matters.

Preface

This is a book about *isomorphisms of types*, a recent difficult research topic in type theory that turned out to be able to have valuable practical applications both for programming language design and for more human-centered information retrieval in software libraries. By means of a deep study of the syntax of the now widely known typed λ-calculus, it is possible to identify some simple equations between types that on one hand allow to improve the design of the ML language, and on the other hand provide the basis for building radically new information retrieval systems for functional software libraries.

We present in this book both the theoretical aspects of these researches and a fully functional implementation of some of their applications in such a way to provide interesting material both for the theoretician looking for proofs and for the practitioner interested in implementation details.

In order to make it possible for these different types of readers to use this book effectively, some special signs are used to designate material that is particularly technical or applied or that represents a digression. When the symbol

appears at the beginning of a section or a subsection, it warns that the material contained in such section is particularly technical with respect to the general level of the chapter or section where it is located. This material is generally reserved to theoreticians and does not need to be read by the casual reader.

The symbol

marks instead those sections, or subsections, that constitute a "detour" in the train of thought. Such material is generally of interest on its own and allows a better understanding of the subject, but it does not need to be read immediately.

Finally, the symbol

marks sections that contain material particularly useful for applied readers interested in implementing new systems based on the theoretical results that they probably do not want to study fully.

Let we just briefly remark that the first chapters contain mostly theoretical sections, while the applied material is mostly to be found in the last Chapters.

Source code

Implementations of the search tools described in this book are available for many different functional programming languages.

Mikael Rittri's retrieval system for Lazy ML is available on request (e-mail: `rittri@cs.chalmers.se`). One version uses unification modulo isomorphisms.

Brian Matthews's retrieval system for Haskell is available on request (e-mail: `bmm@inf.rl.ac.uk`). This system handles Haskell's type-classes.

The Venari project's search tool for SML, with an interface in Gnu Emacs, is available on request from `amy+@cs.cmu.edu`.

Finally, the source code for the complete implementations of the library search systems described in Chapter 6 is available by `anonymous ftp` from `ftp.inria.fr` in

`/lang/caml/V3.1/common.tar.Z` : look at files in `user_lib/FIND_IN_LIB` (equality modulo isomorphisms for CAML), and

`/lang/caml-light/cl6unix.tar.Z` : look at files in `contrib/search_isos` (equality and matching modulo isomorphisms, with NeXT and Gnu Emacs interface, for CamlLight).

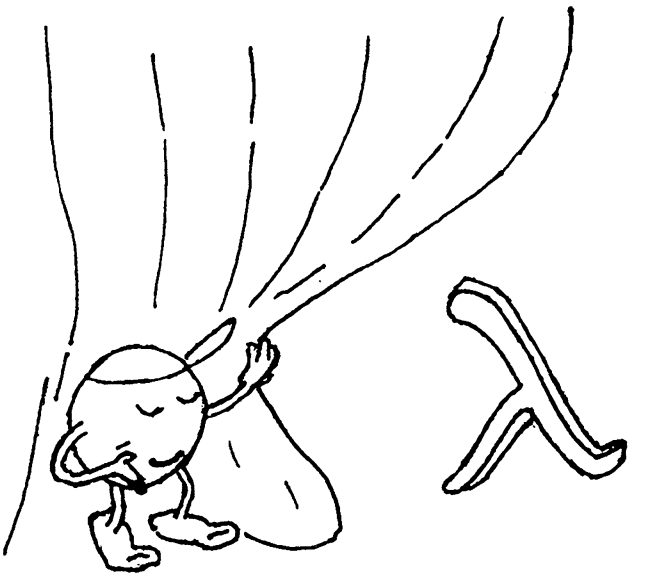
Julien Jalon and Jerôme Vouillon implemented during their *Projet d'informatique* the search tools based on matching.

Notation index

Notation	comment	page
$s.n.f.(A)$	split normal form of type A	175
pts	principal type schema	193

Chapter 1

Introduction



In modern programming languages the notion of type has become so natural and widespread that it is easy to forget about its origins and its theoretical relevance. A *type* is simply seen as a useful means of classification of the objects that a program manipulates: types help to understand better what a program does, and they also provide a valuable firewall against many common programming errors.

But the theory of types is also a well-established and very active field of research in mathematical logic and theoretical computer science, one which can often generate interesting practical fallout. In this book we give a detailed account of one such successful story of nice theory that produced valuable practical fallout: the study of *isomorphic* types, originally started from very theoretical motivations in category theory and in the theory of λ-calculus, turned out to provide a firm basis both for the design of new programming languages and for the development of natural tools for information retrieval in software libraries.

The first sections up to 1.6 of this introduction provide a gentle and general introduction to the notion of type. We start from its origins in mathematics, and then we focus on its modern use in programming languages and type theory, showing how the typed λ-calculus can be of great help to understand many key features of the type systems available in modern programming languages.

Finally, in Section 1.7, we give a general overview of the reasons why types are good candidates to classify software components and how they can be used as retrieval tools, with examples and motivations coming from different programming languages. This will bring up the question of when two types contain the same information and must then be identified; this is addressed in the Section 1.8, with broader references to proof theory, category theory and lambda calculus.

1.1 What is a type?

The modern notion of *type* makes its first appearance in the field of mathematical logic, where it was introduced long before the appearance of digital computers and modern programming languages. As we will briefly recall, two different approaches to the foundation of mathematical logic - one based on sets and the other on functions - had to resort to a notion of *type* in order to avoid paradoxes, and both were soon abandoned as logical formalisms. But the notion of type survived and found a fundamental place in computer science, where it is now an essential tool to design usable programming languages, to prove properties of programs, to ensure data encapsulation and hiding, and to retrieve software components as well as toward many more

applications.

We will now first recall how types developed from set theory, then look at how they evolved in imperative programming languages, and finally follow the smooth evolution of types in the theory of the λ-calculus, which showed up as an essential connection tool between logic and functional programming.

1.2 Types in mathematical logic

At the end of the 19th century, Frege made a revolutionary effort towards a global formalization of mathematics. It is contained in many now famous works (like [Fre79, Fre93, Fre03]), where a first formal theory of set is presented as a basis for the formalization of all the mathematical activity.

In Frege's system, sets are taken as the basic building blocks for a complete formalization of mathematics and are very liberally built out of logical formulae by means of what in modern terms is called the axiom of comprehension. In very simple terms, this means that whenever you are able in Frege's formalism to write down a particular property (like, for example, the property of being an even number), then you can immediately form the set of all objects having such a property (in the example, the set of all even numbers).

Such a liberal system, as discovered in 1902 by Russel, allows us to define paradoxical objects like the set of the sets that are not elements of themselves, $R = \{x | x \notin x\}$ in modern terms. Indeed, "not being an element of itself" is a property that can be formalized in Frege's system, and then R must be a set. But then, is R an element of itself? Either answer to this question is in contradiction with the definition of R, and this fact goes now under the name of Russel's paradox. Due to this inconsistency, Frege's original system allows to prove any proposition, and so has no logical interest.

In order to avoid the paradoxes of Frege's system, Russel and Whitehead introduced in [RW10] a classification of the sets using *types*: a kind of tag that is attached to every set. By using these types, they could properly restrict the set-forming operations so that, for example, one is no longer able to build a set like R. The price to pay is an explicit classification of the sets. This can be considered as the origin of the notion of type, but it was just the beginning of a long and fruitful line of research that shifted over the years from the foundation of mathematics to theoretical computer science, where is now best studied. The interested reader will find a nice historical survey of types in mathematical logic in [CF58, CHS72], while we will focus now on the use of types in computer science.