BOOK REVIEWS

Isomorphisms of Types: from λ-Calculus to Information Retrieval and Language Design

Roberto Di Cosmo
Birkhäuser Verlag AG, Basel, 1994, 243p.
ISBN 3-7643-3763-X

Ileana Valentina Rabega was born in Romania in 1955. She graduated in Mathematics from the University of Bucharest, Faculty of Mathematics, in 1979. Since that year she has been working at the Research Institute for Informatics in Bucharest. Her fields of interest include formal specification / design and correctness proving techniques, CASE systems, artificial intelligence methods applied to software engineering. She is now a senior research worker and her published papers, in Romanian journals or in Proceedings of international symposia, are more than thirty.

As the author says in the preface, this is a book about isomorphisms of types. The fact that theoretical and implementational aspects of the research are as well treated makes the audience of this book become larger. In the reviewer's opinion it is a chance for many readers, that such a book could appear, because it is conceptually very responsive to both researcher's and programmer's needs. In general, many papers on this topic have been written, but what is really needed for is a book putting together all these aspects. As a consequence, there are many potential types of readers: those who want to get an insight into type isomorphism and have a certain background, those who have no background, and try to get familiar with this research topic, practitioners, who could enter now theoretical details, etc. This book has a very attractive style, that makes it interesting for all mentioned readership.

The book is divided into seven chapters that create quite a vast panorama of the research domain.

In the **introduction** the importance of polymorphic systems is explained; it lies in the fact that such systems allow to re-use the same piece of code at different types. For example, in Pascal programming language the type system is strong and explicit; that is, the programmer has to explicitly declare, before using it, the type of each identifier, and once declared an identifier, say, of real type, it must contain a value of this particular type, during execution; it is not possible for it to contain a value of some other type. Functional and logical languages, being more concerned with *what* a program is supposed to do, were gradually endowed with theoretical acquisitions that permitted them to surpass rigid schemes, as those mentioned for Pascal programs.

Concepts such as types, strong, weak, implicit and explicit typing, monomorphic and polymorphic type systems, static and dynamic type checking, how to deal with the issue of program partial and total correctness, equality (isomorphism) of types are both intuitively and formally explained.

The typed calculi used in this book are given in Natural Deduction style formalism, which makes it evident the connection between typed λ-calculus and proofs in Intuitionistic Logic. The most complex calculus introduced in this book is the second-order λ-calculus with pairs and unit type; it is presented by describing types, terms and equality as follows:

- Types are defined by grammar:

$$Type ::= At \mid Var \mid Type \rightarrow Type \mid Type \times Type \mid \forall X.\ Type$$

where *At* are countably many atomic types including a distinguished constant type **T** and *Var* are countably many type variables. Usually $\rightarrow$ is referred as the *arrow* type, and $\times$ as the *product* type.

❑ Terms (M: A means "M is a term of type A")

- the set of terms contains a countable set x, y, ... of term variables for each type and a constant $*$:**T**
- terms are constructed from variables and constants via the following rules (where Γ ranges over environments):

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \rightarrow B} \text{ (lambda abstr.)}$$

$$\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash (MN) : B} \text{ (applic.)}$$

$$\frac{\Gamma \vdash M : A \qquad N : B}{\Gamma \vdash \langle M, N \rangle : A \times B} \text{ (pairing)}$$

$$\frac{\Gamma \vdash M : A \times B}{\Gamma \vdash p_1 M : A} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash p_2 M : B}$$
(projections)

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \lambda X.M : \forall X.A} \text{ (univ. abstr.)}$$

$$\frac{\Gamma \vdash M : \forall X.A}{\Gamma \vdash M [B] : A[B/X]} \text{ (univ. applic.) for any type B.}$$

Pairing and projections are new term formation rules.

- Equality

(β) $(\lambda x.M)N = M[N/x]$
(η) $\lambda x.Mx = M$ if x not free in M
(π) $p_i \langle M_1, M_2 \rangle = M_i$
(**SP**) $\langle p_1 M , p_2 M \rangle = M$
(**top**) $M = *$ if $M : \mathbf{T}$
(β^2) $(\lambda x.M)[A] = M[A/X]$
(η^2) $\lambda x.M[X] = M$, X not free in M.

Equality is generated by β, η, π, **SP**, **top**, β^2 and η^2.
Instead of a reduction rule, the calculi introduced in this book yield equalities between terms, a fact that provides a more abstract and general view.

The **second Chapter** puts forward some results concerning confluence and normalization as general properties for reduction. A reduction system is given by means of reduction rules. In general, two views of a calculus are considered within the reduction strategy of a programming language:

- an *operational* view, where terms are reduced in order to get their values, whatever they are defined to be;
- a *logical* view, where the associated equality of terms is of interest, that is, the conditions when two terms are equal are sought for, irrespective of these two terms meanings.

Confluence says that any two reduction sequences, starting from the same term, have a common reduct, that is, they can be prolonged in a way that allows to reach the same term.

Normalization treats the possibility of reaching a normal form in a finite number of steps.

A basic theory of reduction for the typed calculi that form the basis of the study of isomorphic types in λ-calculus, is developed. A confluent notion of reduction is provided, that permits the work on terms in normal form, being aware of the fact that such a normal form is unique.

The **third Chapter** provides two proofs of strong normalization for theorems introduced in the second Chapter.

The **fourth Chapter** characterizes the isomorphisms which hold in all models of first-order λ-calculus with pairs and unit type i.e. the typed λ-calculus with surjective pairing and

"terminal object". It is also proved that it is decidable whether two types, built from type variables, are isomorphic in all the models of this calculus (which are, in fact, the Cartesian Closed Categories - ccc).

In fact, this chapter and the fifth Chapter provide a finite, complete and decidable axiomatization of the types isomorphic in every model of all interesting subsystems (first- and second-order) of λ-calculus with pairs and unit type, by means of purely syntactical proof theoretic methods is provided. The first -order case could easily be handled due to the reduced interaction between types and terms that characterizes the monomorphic systems. Within the second-order case, the difficulties posed by polymorphic types had to be overcome.

The **fifth Chapter** is dedicated to the proof of completeness of the theory of isomorphisms for isomorphisms in the second-order λ-calculus with unit type. This represents by far the most complex proof in the book, because in the second-order case the problem of invertibility of terms has to be faced . The main ideas underlying this proof are:

❶ Consider a restriction of the class of isomorphisms to isomorphisms of a particular form and a special class of invertible terms, for which a syntactic characterization is given:

- Two relevant classes of types are identified, types not containing products, or terminal objects in the category, which are called *simple types*, and products of simple types, which are called *regular types*. It is proven that the theory of isomorphisms for isomorphisms in second-order λ-calculus with unit type is complete for isomorphisms of types if and only if it is complete for isomorphisms between *regular types*. **That means, a reduction to a subclass of types is obtained.**
- Any isomorphism between regular types can be proved by invertible terms, whose free variables have simple types, called canonical invertible terms, thus obtaining a **reduction to a subclass of terms**.
- The problem of completeness of the theory of isomorphisms for isomorphisms in the second-order λ-calculus with unit type is reduced to the problem of completeness of isomorphisms between *regular types* proven by *canonical* invertible terms.

❷ Characterize canonical bijections of second-order λ-calculus with pairs and unit type under invertible terms of the second order λ-calculus, which benefits from a syntactic characterization in the literature.

❸ Show that the theory of isomorphisms for isomorphisms in the second-order λ-calculus with unit type is complete for the definable isomorphisms showing that the theory of isomorphisms for isomorphisms in the second-order λ-calculus with unit type is complete for isomorphisms between regular types.

An immediate consequence of the main theorem in this chapter is that, given the two types, it is decidable whether they are isomorphic in all models of second-order λ-calculus with pairs and unit type.

These results lay the necessary theoretical basis to the development of both library search tools based on the type as specification paradigm and extensions of the usual type-checking algorithms for strongly typed functional languages.

The **sixth Chapter** provides a first formal foundation for the theory of isomorphisms of types regarding type-assignment calculi, similar to Milner's ML. An adequate new notion of isomorphism is introduced, which allows the design of a search algorithm. This algorithm is more efficient than the previously proposed ones.

These results have led to a complete practical implementation of a library search system for both the CAML and CamlLight functional languages based on equality of types modulo isomorphisms. The algorithm is fully described for the case of the CAML system, but it lends to re-writing for other dialects of ML or different functional languages. On the other side, the new

notion of isomorphism suggests its scope as far as the traditional ML type inference algorithm.

The **seventh Chapter** is dedicated to a comprehensive definition of the advanced work related to the isomorphism of types. Some hints at possible future applications , not yet focussed on, are also made.

Suggestive program examples are given in all chapters of the book. An extensive bibliography, a citation index and a subject index, and extremely suggestive and original drawings, that introduce each chapter, round up the book.

Ileana Valentina Rabega