Texts in Theoretical Computer Science
An EATCS Series

Springer-Verlag Berlin Heidelberg GmbH

Simona Ronchi Della Rocca
Luca Paolini

The Parametric Lambda Calculus

A Metamodel for Computation

Springer

Authors

Prof. Simona Ronchi Della Rocca
Università di Torino
Dipartimento di Informatica
corso Svizzera 185
10149 Torino, Italy
ronchi@di.unito.it
www.di.unito.it/~ronchi

Dr. Luca Paolini
Università di Torino
Dipartimento di Informatica
corso Svizzera 185
10149 Torino, Italy
paolini@di.unito.it
www.di.unito.it/~paolini

Series Editors

Prof. Dr. Wilfried Brauer
Institut für Informatik der TUM
Boltzmannstr. 3, 85748 Garching, Germany
Brauer@informatik.tu-muenchen.de

Prof. Dr. Grzegorz Rozenberg
Leiden Institute of Advanced Computer Science
University of Leiden
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
rozenber@liacs.nl

Prof. Dr. Arto Salomaa
Turku Centre for Computer Science
Lemminkäisenkatu 14 A, 20520 Turku, Finland
asalomaa@utu.fi

Library of Congress Cataloging-in-Publication Data

Ronchi Della Rocca, S. (Simona)
The parametric lambda calculus : A metamodel for computation / Simona Ronchi Della Rocca, Luca Paolini.
p. cm. – (Texts in theoretical computer science)
Includes bibliographical references and index.
ISBN 978-3-642-05746-5 ISBN 978-3-662-10394-4 (eBook)
DOI 10.1007/978-3-662-10394-4
1. Lambda calculus. I. Paolini, Luca, 1970– II. Title. III. Series.
QA9.5.R66 2004 511.3'5–dc22 2003069100

ACM Computing Classification (1998): F.4, F.3, I.2.3, D.2

ISBN 978-3-642-05746-5

Originally published by Springer-Verlag Berlin Heidelberg New York in 2004
Softcover reprint of the hardcover 1st edition 2004

Cover Design: KünkelLopka, Heidelberg
Typesetting: Camera-ready by the authors
Printed on acid-free paper 45/3142/GF - 5 4 3 2 1 0

To Corrado Böhm, from which Simona and –
by transitivity – Luca learned the pleasure
of research and the interest in λ-calculus

Preface

The λ-calculus was invented by Church in the 1930s with the purpose of supplying a logical foundation for logic and mathematics [25]. Its use by Kleene as a coding for computable functions makes it the first programming language, in an abstract sense, exactly as the Turing machine can be considered the first computer machine [57]. The λ-calculus has quite a simple syntax (with just three formation rules for terms) and a simple operational semantics (with just one operation, substitution), and so it is a very basic setting for studying computation properties.

The first contact between λ-calculus and real programming languages was in the years 1956-1960, when McCarthy developed the LISP programming language, inspired from λ-calculus, which is the first "functional" programming language, i.e., where functions are first-class citizens [66]. But the use of λ-calculus as an abstract paradigm for programming languages started later as the work of three important scientists: Strachey, Landin and Böhm. Strachey used the λ-notation as a descriptive tool to represent functional features in programming when he posed the basis for a formal semantics of programming languages [92]. Landin formalized the idea that the semantics of a programming language can be given by translating it into a simpler language that is easier to understand. He identified such a target language in λ-calculus and experimented with this idea by giving a complete translation of ALGOL60 into λ-calculus [64]. Moreover, he declared in [65] that a programming language is nothing more than λ-calculus plus some "syntactic sugar". Böhm was the first to use λ-calculus as an effective programming language, defining, with W. Gross, the CUCH language, which is a mixture of λ-calculus and the Curry combinators language, and showing how to represent in it the most common data structures [19].

But, until the end of the 1960s, λ-calculus suffered from the lack of a formal semantics. In fact, while it was possible to codify in it all the computable functions, the meaning of a generic λ-term not related to this coding was unclear. The attempt to interpret λ-terms as set-theoretic functions failed, since it would have been necessary to interpret it into a set D isomorphic to the set of functions from D to D, which is impossible since the two spaces always have different cardinality. Scott [88, 89] solved the problem by interpreting λ-calculus in a lattice isomorphic to the space of its continuous functions,

thus giving it a clear mathematical interpretation. So the technique of interpretation by translation, first developed by Landin, became a standard tool to study the denotational semantics of programming languages; almost all textbooks in denotational semantics follow this approach [91, 98].

But there was a gap between λ-calculus and the real functional programming languages. The majority of real functional languages have a "call-by-value" parameter passing policy, i.e., parameters are evaluated before being passed to a function, while the reduction rule of λ-calculus reflects a "call-by-name" policy, i.e., a policy where parameters are passed without being evaluated. In the folklore there was the idea that a call-by-value behaviour could be mimicked in λ-calculus just by defining a suitable reduction strategy. Plotkin proved that this intuition was wrong and that λ-calculus is intrinsically call-by-name [78]. So, in order to describe the call-by-value evaluation, he proposed a different calculus, which has the same syntax as λ-calculus, but a different reduction rule.

The aim of this book is to introduce both the call-by-name and the call-by-value λ-calculi and to study their syntactical and semantical properties, on which their status of paradigmatic programming languages is based. In order to study them in a uniform way we present a new calculus, the $\lambda\Delta$-calculus, whose reduction rule is parametric with respect to a subset Δ of terms (called the set of input values) that enjoy some suitable conditions. Different choices of Δ allow us to define different languages, in particular the two λ-calculus variants we are speaking about. The most interesting feature of $\lambda\Delta$-calculus is that it is possible to prove important properties (like confluence) for a large class of languages in just one step. We think that $\lambda\Delta$-calculus can be seen as the foundation of functional programming.

Organization of the Book

The book is divided into four parts, each one composed of different chapters. The first part is devoted to the study of the syntax of $\lambda\Delta$-calculus. Some syntactical properties, like confluence and standardization, can be studied for the whole Δ class. Other properties, like solvability and separability, cannot be treated in a uniform way, and they are therefore introduced separately for different instances of Δ.

In the second part the operational semantics of $\lambda\Delta$-calculus is studied. The notion of operational semantics can be given in a parametric way, by supplying not only a set of input values but also a set of output values Θ, enjoying some very natural properties. A universal reduction machine is defined, parametric into both Δ and Θ, enjoying a sort of correctness property in the sense that, if a term can be reduced to an output value, then the machine stops, returning a term operationally equivalent to it. Then four particular reduction machines are presented, three for the call-by-name λ-calculus and one for the call-by-value λ-calculus, thereby presenting four operational behaviours that

are particularly interesting for modeling programming languages. Moreover, the notion of extensionality is revised, giving a new parametric definition that depends on the operational semantics we want to consider.

The third part is devoted to denotational semantics. The general notion of a model of $\lambda\Delta$-calculus is defined, and then the more restrictive and useful notion of a filter model, based on intersection types, is given. Then four particular filter models are presented, each one correct with respect to one of the operational semantics studied in the previous part. For two of them completeness is also proved. The other two models are incomplete: we prove that there are no filter models enjoying the completeness property with respect to given operational semantics, and we build two complete models by using a technique based on intersection types. Moreover, the relation between the filter models and Scott's models is given.

The fourth part deals with the computational power of $\lambda\Delta$-calculus. It is well known that λ-calculus is Turing complete, in both its call-by-name and call-by-value variants, i.e. it has the power of the computable functions. Here we prove something more, namely that each one of the reduction machines we present in the third part of this book can be used for computing all the computable functions.

Use of the Book

This book is dedicated to researchers, and it can be used as a textbook for master's or PhD courses in Foundations of Computer Science. Moreover, we wish to advise the reader that its aim is not to cover all possible topics concerning λ-calculus, but just those syntactical and semantics properties which can be used as tools for the foundation of programming languages. The reader interested in studying λ-calculus in itself can use the classical textbook by Barendregt [9], or other more descriptive ones such as [51] or [60]. The reader interested in a typed approach can read Mitchell's text [69] for an introduction, in which two chapters are dedicated to simply typed λ-calculus and its model, and the book of Hindley for a complete development of the topic [49].

Acknowledgement. Both authors would like to thank all the people of the "lambda-group" at the Dipartimento di Informatica of the Università di Torino for their support and collaboration. Moreover they are grateful to Roger Hindley and Elaine Pimentel for pointing out some inaccuracies. Luca Paolini thanks Pino Rosolini for the useful and interesting discussions about the topics of this book. Simona Ronchi Della Rocca did the final revision of the book during a sabbatical period. Some friends offered her hospitality and a stimulating scientific environment: Betti Venneri, Gigi Liquori, Rocco De Nicola, Pierre Lescanne and Philippe De Groote. To all of them she wants to

express her gratitude. Last but not least, both the authors thank the publisher Ingeborg Mayer, whose patient assistance made possible the publication of this book.

Torino, May 2004

Simona Ronchi Della Rocca
Luca Paolini

Contents

Part III. Denotational Semantics