Categories for Software Engineering

José Luiz Fiadeiro

Categories for Software Engineering

Springer

José Luiz Fiadeiro
University of Leicester
Department of Computer Science
University Road
Leicester LE1 7RH
United Kingdom

Library of Congress Control Number: 2004113132

ACM Computing Classification (1998):
D.2.11, F.3.1, D.2.1, D.2.4, D.1.3

ISBN 3-540-20909-3 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

Cover design: KünkelLopka, Heidelberg
Production: LE-TeX Jelonek, Schmidt & Vöckler GbR, Leipzig
Typesetting by the author
Printed on acid-free paper 45/3142/YL - 5 4 3 2 1 0

From a loving father

Preface

Why Another Book on Category Theory?

In the past ten years, several books have been published on category theory either by computer scientists or having computer scientists as a target audience (e.g. [6, 12, 22, 89, 105], to which a precious collection of little gems [90] and the chapter cum book [91] should be added). Isn't the working computer scientist spoilt with choice?

Although each of the above mentioned books presents an approach of its own, there is one aspect in common in their view of computer science: the analogy between arrows (morphisms) and (classes of) computations. This "type-theoretic" or "functional" approach corresponds to a view of computer science as a science of computation, i.e. a discipline concerned with the study of computational phenomena where the focus is on the nature and organisation of computations.

However, there is another view of computer science where the focus is, instead, on the development of computer programs or systems. This is the approach that supports, for instance, software engineering. From this point of view, arrows do not capture computational phenomena, or abstractions thereof, but instead relationships between programs, or abstractions of programs, that arise in the development of computer systems, for instance, refinement of higher-level specifications into executable programs [100, 104], and superposition of new features over existing systems [72].

Not surprisingly, this same difference in the points of view can be found when logic is taken as a mathematical domain for formalising aspects of computer science. The "computations as proofs" paradigm is the one that corresponds to the "classical" application of category theory. Terms of the logic correspond to objects in a category of programs whose morphisms capture (partial) computations. From a logical point of view, the perspective that we take in this book is not centred on terms but on theories as system specifications. Morphisms then capture what in logic is known as "interpretations between theories", the cornerstone for the formalisation of refinement in program development and other operations on specifications and system designs [17, 82, 103, 104].

Category theory can also be presented as the branch of mathematics that, par excellence, addresses "structure". As the introduction will try to

explain, this is because category theory causes structure to emerge from relationships between objects as captured by arrows, and not extensionally as in set theory. Indeed, the term *morphism*, often used for arrow in category theory, has in its etymology the notion of *preservation of form*. What these structures are, or mean, is up to the "user". Hence, in the "classical" approach, we find applications of category theory that address the structure of computations. In the approach that is taken in this book, the structures that are addressed are those that capture modularisation principles in software development, in particular, those that have been used for constructing distributed systems (e.g. as in [81]) and, more recently, emerging in the guise of what has become known as software architectures [50].

The practical difference between the two approaches in what concerns category theory in general, and this book in particular, is that the reader will not find as many references to algebraic topology or related fields of mathematics as applied, for instance, to domain theory. Although this book is still "mathematical", the software engineer will find the mathematics applied to objects of its day-to-day concerns: programs, object classes, specifications, designs, and so on.

This approach can be also situated as belonging to the class of applications of category theory to general systems theory, namely in the tradition initiated in [52, 53, 64], an area of science that, as the name indicates and the introduction elucidates, encompasses more than computational systems in the traditional sense. Through books aimed at wider audiences like [71], a unifying view of complex systems as they arise in disparate areas like physics, biology, social sciences, economics and, yes, informatics, has started to emerge (pun intended), which is a clear indication of new levels of maturity in science in general and informatics in particular. Hence, one of the purposes of this book is to help computing scientists and software engineers acquire formal tools that will enable them to follow and participate in this "new" culture.

A trait that is common to all these areas is a view of complex systems as communities of interacting, simpler, autonomous entities. Whereas, in areas like biology or social sciences, the notion of "community" is intrinsic, its use in areas like software engineering is more artificial and is normally identified with methods and development techniques that, in the past few years, have attempted to tackle complexity by borrowing the organisational principles that can be recognised in such "natural" communities. Object-oriented modelling, agent-based programming and component-based development all make use, in one way or another and with different emphasis, of this analogy. This brings us to the application area covered in the third part of this book.

CommUnity is the name of a language for parallel program design that is similar to Unity [19] but adopts instead an interaction model that places it in the realm of these more general and unifying approaches to systems.

It addresses in particular the most recent trend, *service*-oriented software development, an (r)evolution of the popular object-oriented modelling techniques for the "Internet-age" or what is becoming known as the "real-time" or "now" economy. The distinctive feature of this new trend is in the emphasis that it puts in the externalisation and explicit modelling of interactions as first-class citizens so that systems can be more easily reconfigured, in run-time, and without interruption of vital services. These characteristics match, precisely, features that are intrinsic to category theory, namely those that distinguish it from set theory.

That is why, even if a substantial part of this book is illustrated with examples borrowed from software engineering practice, we decided to devote three chapters to the application of category theory to CommUnity and its relationship to software architectures. This material will provide an opportunity for the reader to see concepts and techniques of category theory applied in an integrated and systematic way. At the same time, the reader will be able to appreciate how far one can go in formalising software development methods and techniques in mathematical frameworks, which is essential for a mature engineering discipline and, in my opinion, is our responsibility as computing scientists.

This Book and Its Many Authors

Mentioning the connections between category theory and general systems theory is a good opportunity to give due credit to Joseph Goguen for the profound inspiration that his work has instilled, a sentiment that I know is shared by many other researchers in computing science. He has expressed his own views on the applications of category theory to computing in several publications, most notably in [57], which include detailed summaries of technical results that we all have found very useful when categorical approaches were still regarded, at best, as "exotic" [60, 61, 66, 102]. All of us regret that this material has never found its way to a textbook. Because it is not our aim to fill this gap, the reader is strongly encouraged to consult this rich legacy at his or her own pace, bearing in mind that the list of references that is provided at the end is far from being complete.

Completeness is, in fact, a concern that has remained largely alien to my research agenda. (This observation is intended to make some readers smile, but you can take it literally.) This book is more about a personal experience than the output of a rational process of identifying "the" or "a" complete categorical kernel that software engineers can use as a toolbox. The only justification for the inclusion of many concepts and constructions is that they were of help to me, either technically or aesthetically, making it likely that they will be directly useful for other people "like me". The

exclusion of many other, even very basic concepts,[1] can be justified by the officious disclaimer that "the line has to be drawn somewhere", but, most of the time, the reason is that I never stumbled upon them in my daily routine or simply that I have not developed an understanding about them that is deep enough to add any value to what can be found in other books.

This personal experience has gone through well identifiable periods, each of which is associated with a different focus of interest in computing and a group of people with whom I worked directly and whose contributions I would like to acknowledge. My first contact with category theory was when I was studying mathematics as an undergraduate at the University of Lisbon, and Prof. Furtado Coelho challenged the wrath of my fellow students, and his fellow staff, by including this most exotic, difficult and useless of subjects in the curriculum of Algebra II. Applications to computing science came a year later through the study of Goguen and Burstall's Theory of Institutions as a means of formalising conceptual modelling and knowledge representation approaches, under the supervision and in collaboration with Amílcar and Cristina Sernadas [46]. This is when things started to get serious.

In 1988, I started what has been a very rewarding collaboration with Tom Maibaum. During the three years I spent at Imperial College, we developed a categorical approach to object-oriented development based on temporal logic specifications [39], a marriage between my previous work with institutions and the ideas of Tom Maibaum and Paulo Veloso on the nature of specifications in system development [103]. Their contribution permeates the material that is exposed in a way that cannot be referenced in the same way as a technical result. I have been very fortunate to be able to keep exchanging ideas and experiences with them; there are always hidden subtleties that only come to the surface when you are challenged by people like them and required to scratch the innermost levels of your understanding to satisfy their curiosity.

During this same time, Félix Costa explored the categorical semantics of objects from the point of view of algebraic models of concurrency [21]. In 1992, we brought it all together [32]! My collaboration with Félix provided much of the inspiration that led to my own understanding of the application of category theory to systems modelling. Although specific contributions are acknowledged with references to his work, it would be unfair to reduce his contribution to this book to those occasions.

The next phase is devoted to the (then) emerging field of software architecture. It is centred on a language – CommUnity – that I developed together with Georg Reichwein and Tom Maibaum in an initial period, and later on with my students Antónia Lopes and Michel Wermelinger. It

[1] Yes, I know that I will not be forgiven for having left out "must-haves" such as the Yoneda lemma, Cartesian-closed categories, topoi, monads, and so on.

started as a proof of concept, showing that Goguen's categorical approach could be applied to parallel program design in the style of Unity [19] and Interacting Processes [48]. Later, it evolved into a prototype language for architectural modelling [34], a process that led me to understand many concepts that, until then, were blurred by the poor expressive power of the formalisms with which we had been working: non-determinism versus under-specification [77], refinement versus composition [78], and the role of "signatures" in separating computation and coordination [35]. Some of this is revealed in Part III of this book, but you will have to wait for another book to have the full story!

Although this "architectural" period is still very much alive (which does not mean that the others are already dead), another step in this evolution process has occurred: the realization that category theory provides a perfect fit to support service-oriented software development, for instance in the sense of Web-services. But this step is so recent that, in fairness, I cannot acknowledge/blame anybody in particular for it. Nevertheless, it is unlikely that it would have happened so soon, or at all, if I had not accepted the challenge that Luís Andrade presented me with for working with ATX Software SA in putting these "theories" into "practice". This has been a very rewarding process that has given me the opportunity to understand the implications of many of the structures and mechanisms that are intrinsic to category theory. I hope that I have managed to permeate this understanding in the way the material is exposed in the book.

This is probably why this book is being finalised now and only now: during each of the periods I mentioned, a book was planned and parts were sketched. It is only now that the work of so many people has contributed to the contents that I can safely write it on my own without feeling guilty for excluding anybody in particular from coauthoring it.

It so happens that the last thrust in writing this book was made during my first year at the University of Leicester, a renowned address for research in category theory and its applications to computer science. Although I can honestly assure the reader that the decision to join Leicester was not for the advantages of promoting this book, it is certainly a privilege for the book to bear this affiliation!

Finally, I should thank all the colleagues and students who have trod with me the paths that you can choose to follow in this book. The opportunity to discuss and lecture on many of the topics that are covered contributed decisively in helping me reach the level of maturity that made me decide that this book could be written. The feedback I received from tutorials presented at events such as ECOOP, ETAPS, FME, OOPSLA and TOOLS also helped me decide that this book should be written. The interest and encouragement of people like Ira Forman and Desmond D'Souza have also reassured me that the message could perfectly go beyond the walls of academia.

Special Acknowledgements

Although the previous paragraphs have given me the opportunity to acknowledge the contributions of a number of people and institutions, there are some specific colleagues to whom I would like to express my deepest gratitude for direct contributions to this book:

- *Félix Costa*: A significant part of the material covered in Part II was developed jointly with him as reported in [32, 33]. As already mentioned, much of my own understanding of category theory and its role in computing science grew from discussions with him.
- *Antónia Lopes* and *Michel Wermelinger*: The fact that Part III of this book was essentially extracted from [37, 79] is a good indication of how important and extensive their contribution has been. CommUnity as we know it today is as much theirs as it is mine.
- *Tom Maibaum*: His encouragement and support in the earlier phases of the production of the book were decisive.
- *Uwe Wolter*: He had the courage to follow an early draft in a course that he gave in 2002/2003 (repeated in 2003/2004) at the University of Bergen (Norway). As a result, I received precious amounts of feedback, which was invaluable for the final tuning of the material and the way it is now presented.

Finally, I would like to thank the EPSRC for an eight-month visiting fellowship at King's College London in 1999, which gave me the opportunity to make a significant advance in the writing of the book, to Janet Maibaum for her help in setting Microsoft Word up to the job[1] and to the team at Springer for their enthusiasm, support and advice.

Gesse,[2] April 2004 *José Luiz Fiadeiro*

[1] Yes, this book is a proof that writing about category theory is not reserved to users of a well-known typesetting system that I will not name... And this remark is not meant as a recommendation for the products developed by a company that I have already named...

[2] The little village in the French Pyrenees, by the river Aude, where this book was written and revised from 2000 until completion.

Contents

Part II Advanced Topics

Part III Applications