Foundations of Computing Series

Basic Category Theory for Computer Scientists

Benjamin C. Pierce

Copyrighted material

Foundations of Computing
Michael Garey and Albert Meyer, editors

Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks, Frank Thomson Leighton, 1983

Equational Logic as a Programming Language, Michael J. O'Donnell, 1985

General Theory of Deductive Systems and Its Applications, S. Yu Maslov, 1987

Resource Allocation Problems: Algorithmic Approaches, Toshihide Ibaraki and Naoki Katoh, 1988

Algebraic Theory of Processes, Michael Hennessy, 1988

PX: A Computational Logic, Susumu Hayashi and Hiroshi Nakano, 1989

The Stable Marriage Problem: Structure and Algorithms, Dan Gusfield and Robert Irving, 1989

Realistic Compiler Generation, Peter Lee, 1989

Single-Layer Wire Routing and Compaction, F. Miller Maley, 1990

Categories, Types, and Structures: An Introduction to Category Theory for the Working Computer Scientist, Andrea Asperti and Giuseppe Longo, 1991

Basic Category Theory for Computer Scientists, Benjamin C. Pierce, 1991

Copyrighted image

Copyrighted material

Basic Category Theory for Computer Scientists

This One

Copyrighted image

Copyrighted material

Foundations of Computing
Michael Garey and Albert Meyer, editors

Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks, Frank Thomson Leighton, 1983

Equational Logic as a Programming Language, Michael J. O'Donnell, 1985

General Theory of Deductive Systems and Its Applications, S. Yu Maslov, 1987

Resource Allocation Problems: Algorithmic Approaches, Toshihide Ibaraki and Naoki Katoh, 1988

Algebraic Theory of Processes, Michael Hennessy, 1988

PX: A Computational Logic, Susumu Hayashi and Hiroshi Nakano, 1989

The Stable Marriage Problem: Structure and Algorithms, Dan Gusfield and Robert Irving, 1989

Realistic Compiler Generation, Peter Lee, 1989

Single-Layer Wire Routing and Compaction, F. Miller Maley, 1990

Categories, Types, and Structures: An Introduction to Category Theory for the Working Computer Scientist, Andrea Asperti and Giuseppe Longo, 1991

Basic Category Theory for Computer Scientists, Benjamin C. Pierce, 1991

Copyrighted material

Basic Category Theory for Computer Scientists

Benjamin C. Pierce

The MIT Press
Cambridge, Massachusetts
London, England

Copyrighted material

© 1991 Massachusetts Institute of Technology

All right reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

This book was typeset by the author using TEX3.0 and converted to PostScript using Y&Y's DVIPSONE™. Camera-ready copy was produced by Chiron, Inc.

Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Pierce, Benjamin C.
  Basic category theory for computer scientists / Benjamin C. Pierce.
    p.   cm. — (Foundations of computing)
  Includes bibliographical references and index.
  ISBN 0-262-66071-7
  1. Computer science—Mathematics. 2. Categories (Mathematics) I. Title. II. Series.
QA76.9.M35P54 1991
511.3—dc20   91-8489
                                                                          CIP

10 9 8

Copyrighted material

To Roger, Alexandra, and Jessica

Copyrighted material

Contents

Copyrighted material

Series Foreword

Theoretical computer science has now undergone several decades of development. The "classical" topics of automata theory, formal languages, and computational complexity have become firmly established, and their importance to other theoretical work and to practice is widely recognized. Stimulated by technological advances, theoreticians have been rapidly expanding the areas under study, and the time delay between theoretical progress and its practical impact has been decreasing dramatically. Much publicity has been given recently to breakthroughs in cryptography and linear programming, and steady progress is being made on programming language semantics, computational geometry, and efficient data structures. Newer, more speculative, areas of study include relational databases, VLSI theory, and parallel and distributed computation. As this list of topics continues expanding, it is becoming more and more difficult to stay abreast of the progress that is being made and increasingly important that the most significant work be distilled and communicated in a manner that will facilitate further research and application of this work. By publishing comprehensive books and specialized monographs on the theoretical aspects of computer science, the series on Foundations of Computing provides a forum in which important research topics can be presented in their entirety and placed in perspective for researchers, students, and practitioners alike.

Michael R. Garey
Albert R. Meyer

Copyrighted material

Preface

What we are probably seeking is a "purer" view of functions: a theory of functions in themselves, not a theory of functions derived from sets. What, then, is a pure theory of functions? Answer: category theory.

— Scott [104, p. 406]

Category theory is a relatively young branch of pure mathematics, stemming from an area—algebraic topology—that most computer scientists would consider esoteric. Yet its influence is being felt in many parts of computer science, including the design of functional and imperative programming languages, implementation techniques for functional languages, semantic models of programming languages, models of concurrency, type theory, polymorphism, specification languages, constructive logic, automata theory, and the development of algorithms.

The breadth of this list underscores an important point: category theory is not specialized to a particular setting. It is a basic conceptual and notational framework in the same sense as set theory or graph theory, though it deals in more abstract constructions and requires somewhat heavier notation. The cost of its generality is that category-theoretic formulations of concepts can be more difficult to grasp than their counterparts in other formalisms; the benefit is that concepts may be dealt with at a higher level and hidden commonalities allowed to emerge.

Recent issues of theoretical computer science journals give ample evidence that category theory is already an important tool in some parts of the field. In a few areas—notably domain theory and semantics—it is now a standard language of discourse. Fortunately for the beginner, most computer science research papers draw only on the notation and some relatively elementary results of category theory. The ADJ group, early proponents of category theory in computer science, sound a reassuring note in the introduction to one of their papers [116]: "...do not succumb to a feeling that you must understand *all* of category theory before you put it to use. When one talks of a 'set theoretic' model for some computing phenomenon, [one] is not thinking of a formulation in terms of measurable cardinals! Similarly, a category theoretic model does not *necessarily* involve the Kan extension theorem or double categories."

The first drafts of this book were written while I was studying category theory myself, as background for graduate research in program-

Copyrighted material

ming languages. Its aim, therefore, is not to promote a particular point of view about how category theory can be applied in computer science—a task better undertaken by more experienced practitioners—but simply to orient the reader in the fundamental vocabulary and synthesize the explanations and intuitions that were most helpful to me on a first encounter with the material.

The tutorial in Chapters 1 and 2 should provide a thorough enough treatment of basic category theory that the reader will feel prepared to approach current research papers applying category theory to computer science or proceed to more advanced texts—for example, the excellent new books by Asperti and Longo [2] and Barr and Wells [5]—for deeper expositions of specific areas. It covers essential notation and constructions and a few more advanced topics (notably adjoints) that are sometimes skipped in short introductions to the subject but are relevant to an appreciation of the field. Chapter 3 illustrates the concepts presented in the tutorial with a sketch of the connection between cartesian closed categories and λ-calculi, an application in the design of programming languages, a summary of work in categorical models of programming language semantics, and a detailed description of some category-theoretic tools for the solution of recursive domain equations. Chapter 4 briefly surveys some of the available textbooks, introductory articles, reference works, and research articles on category theory applied to computer science. A summary of notation and an index appear at the end.

This book could not have been written without the encouragement and generous assistance of my teachers, colleagues, and friends. I am especially grateful to Nico Habermann for suggesting the project; to DEC Systems Research Center, Carnegie Mellon University, and the Office of Naval Research for support while it was underway; to Rod Burstall, Luca Cardelli, Peter Freyd, Robert Harper, Giuseppe Longo, Simone Martini, Gordon Plotkin, John Reynolds, and Dana Scott for informative conversations about its subject matter; to Bob Prior at MIT Press for patient editorial advice; and to Lorrie LeJeune for efficient handling of the manuscript. Comments and suggestions from Martín Abadi, Penny Anderson, Violetta Cavalli-Sforza, Scott Dietzen, Conal Elliott, Andrzej Filinski, Susan Finger, Robert Goldblatt, John Greiner, Nico Habermann, Robert Harper, Nevin Heintze, Dinesh Katiyar, Peter Lee, Mark Maimone, Spiro Michaylov, Frank Pfenning, David Plaut, John Reynolds, Dwight Spencer, Robert Tennent, James Thatcher, Todd Wilson, Elizabeth Wolf, and two anonymous referees greatly improved my presentation of the material and eliminated a number of errors in previous drafts.

Copyrighted material

Finally, I am pleased to acknowledge a huge debt to the labors of other authors, particularly to Robert Goldblatt [40], Saunders Mac Lane [67], and David Rydeheard [95,96,98]. Their books, foremost among many others, were frequent guides in the choice of examples and exercises, organization of material, and proper presentation of the subject's "folklore." There are a few points—marked in the text—where I have closely followed the structure of a particularly beautiful presentation of a concept by another author. Errors in these sections, as in the rest of the text, are of course solely my responsibility.

Pittsburgh, Pennsylvania
January 25, 1991

Copyrighted material

1 Basic Constructions

This chapter and the following one present a brief tutorial on basic concepts of category theory. The goals of the tutorial are, first, to be complete enough to prepare the reader for more difficult textbooks and research papers applying category theory in computer science; second, to cover important topics in sufficient depth that the reader comes away with some sense of the contribution of category theory to mathematical thinking; and third, to be reasonably short. Most sections begin with a rigorous definition, prefaced with an informal explanation of the construction and followed by examples and exercises illustrating its use in various contexts.

1.1 Categories

We begin by defining the notion of category and presenting a variety of examples from computer science and algebra.

1.1.1 Definition A **category** $\mathbf{C}$ comprises:

1. a collection of **objects**;
2. a collection of **arrows** (often called **morphisms**);
3. operations assigning to each arrow f an object $\mathit{dom}\ f$, its **domain**, and an object $\mathit{cod}\ f$, its **codomain** (we write $f : A \to B$ or $A \xrightarrow{f} B$ to show that $\mathit{dom}\ f = A$ and $\mathit{cod}\ f = B$; the collection of all arrows with domain A and codomain B is written $\mathbf{C}(A, B)$);
4. a composition operator assigning to each pair of arrows f and g, with $\mathit{cod}\ f = \mathit{dom}\ g$, a **composite** arrow $g \circ f : \mathit{dom}\ f \to \mathit{cod}\ g$, satisfying the following *associative law*:

   for any arrows $f : A \to B$, $g : B \to C$, and $h : C \to D$ (with A, B, C, and D not necessarily distinct),

   $$h \circ (g \circ f) = (h \circ g) \circ f;$$
5. for each object A, an **identity** arrow $id_A : A \to A$ satisfying the following *identity law*:

   for any arrow $f : A \to B$,

   $$id_B \circ f = f \quad \text{and} \quad f \circ id_A = f.$$

Copyrighted material

1.1.2 Remark Categories are defined here in terms of ordinary set theory. "Collections" are just sets, or occasionally proper classes, since we want to talk about things like the "collection of all sets," which is too big to be a set. "Operations" are set-theoretic functions. "Equality" is set-theoretic identity.

Our first example, an important source of intuition throughout the tutorial, is the category whose objects are sets and whose arrows are functions. There is no circularity here: we are not *defining* sets in terms of categories, but merely *presenting* a well-known mathematical domain as a category.

1.1.3 Example The category **Set** has sets as objects and total functions between sets as arrows. Composition of arrows is set-theoretic function composition. Identity arrows are identity functions.

To see that **Set** is a category, let us restate its definition in the same format as Definition 1.1.1 and check that the laws hold:

1. An object in **Set** is a set.
2. An arrow $f : A \to B$ in **Set** is a total function from the set A into the set B.
3. For each total function f with domain A and codomain B, we have $dom\ f = A$, $cod\ f = B$, and $f \in \mathbf{Set}(A, B)$.
4. The composition of a total function $f : A \to B$ with another total function $g : B \to C$ is the total function from A to C mapping each element $a \in A$ to $g(f(a)) \in C$. Composition of total functions on sets is associative: for any functions $f : A \to B$, $g : B \to C$, and $h : C \to D$, we have $h \circ (g \circ f) = (h \circ g) \circ f$.
5. For each set A, the identity function id_A is a total function with domain and codomain A. For any function $f : A \to B$, the identity functions on A and B satisfy the equations required by the identity law: $id_B \circ f = f$ and $f \circ id_A = f$.

1.1.4 Remark There is one subtlety in the definition of the category **Set**: each function on sets corresponds to many arrows in **Set**. For example, the function that takes every real number r to r^2 maps elements of R (the set of real numbers) into elements of R, and hence corresponds to an arrow $s : \mathsf{R} \to \mathsf{R}$. But it also maps elements of R into elements of R^+ (the set of nonnegative real numbers), and hence corresponds to an arrow $s' : \mathsf{R} \to \mathsf{R}^+$. These are two *different* arrows of the category **Set**.

To be rigorous, we should define a **Set**-arrow $f : A \to B$ to be a tuple (f, B), where f is a total function with domain A and B is a set

Copyrighted material

Basic Category Theory for Computer Scientists
Benjamin C. Pierce

Category theory is a branch of pure mathematics that is becoming an increasingly important tool in theoretical computer science, especially in programming language semantics, domain theory, and concurrency, where it is already a standard language of discourse.

Assuming a minimum of mathematical preparation, *Basic Category Theory for Computer Scientists* provides a straightforward presentation of the basic constructions and terminology of category theory, including limits, functors, natural transformations, adjoints, and cartesian closed categories. Four case studies illustrate applications of category theory to programming language design, semantics, and the solution of recursive domain equations. A brief literature survey offers suggestions for further study in more advanced texts.

The tutorial in section 1 provides a treatment of basic category theory that is deep enough to prepare readers for some of the current research papers applying category theory to computer science. It covers essential notation and constructions and a few more advanced topics, such as adjoints, that are sometimes skipped in short introductions but are relevant to an appreciation of the field.

Section 2 illustrates the concepts presented in the tutorial with four case studies — a sketch of the connections between cartesian closed categories and lambda calculi, an application to the design of programming languages, a summary of work in categorical models of programming language semantics, and a detailed description of some category-theoretic tools for the solution of recursive domain equations.

Section 3 provides a useful guide to the existing literature, including textbooks, standard reference works, and selected research papers.

Benjamin C. Pierce received his doctoral degree from Carnegie Mellon University.

Foundations of Computing series, Research Reports and Notes

ISBN 0-262-66071-7
90000>
9 780262 660716

The MIT Press
Massachusetts Institute of Technology
Cambridge, Massachusetts 02142

Copyrighted material