

ABRAMSKY,
GABBAY, and
MATEAUM

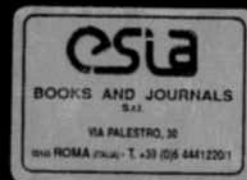
Handbook of Logic in
Computer Science

VOLUME 3

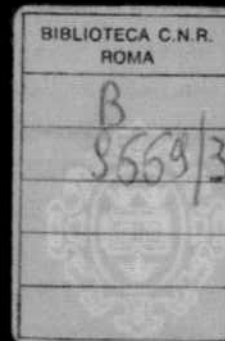
Handbook of Logic in Computer Science

Volume 3
Algebraic Methods

Edited by
ALEXANDER ABRAMSKY, JOHN DE GROOT,
AND ROBERT MATEAUM



0 19 853762 X



OXFORD

OXFORD SCIENCE PUBLICATIONS

HANDBOOKS OF LOGIC IN COMPUTER SCIENCE
and
ARTIFICIAL INTELLIGENCE AND LOGIC PROGRAMMING

Executive Editor

Dov M. Gabbay

Administrator

Jane Spurr

Handbook of Logic in Computer Science

- Volume 1** Background: **Mathematical** structures
- Volume 2** Background: Computational structures
- Volume 3** Semantic structures
- Volume 4** Semantic modelling
- Volume 5** Theoretical methods in specification and verification
- Volume 6** Logical methods in computer science

Handbook of Logic in Artificial Intelligence and
Logic Programming

- Volume 1** Logical foundations
- Volume 2** Deduction **methodologies**
- Volume 3** Nonmonotonic reasoning and uncertain reasoning
- Volume 4** Epistemic and temporal reasoning
- Volume 5** Logic programming

Handbook of Logic in Computer Science

Volume 3

Semantic Structures

Edited by

S. ABRAMSKY

Professor of Computing Science

DOV M. GABBAY

Professor of Computing Science

and

T. S. E. MAIBAUM

*Professor of Foundations of
Software Engineering*

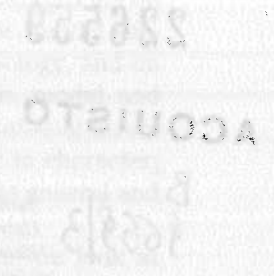
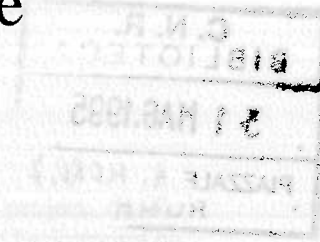
*Imperial College of Science, Technology and Medicine
London*

Volume Co-ordinator

S. ABRAMSKY

CLARENDON PRESS · OXFORD

1994



Oxford University Press, Walton Street, Oxford OX2 6DP

Oxford New York

Athens Auckland Bangkok Bombay
Calcutta Cape Town Dar es Salaam Delhi
Florence Hong Kong Istanbul Karachi
Kuala Lumpur Madras Madrid Melbourne
Mexico City Nairobi Paris Singapore
Taipei Tokyo Toronto

and associated companies in
Berlin Ibadan

Oxford is a trade mark of Oxford University Press

Published in the United States by
Oxford University Press Inc., New York

© The contributors listed on p. xv, 1994

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, without the prior permission in writing of Oxford University Press. Within the UK, exceptions are allowed in respect of any fair dealing for the purpose of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act, 1988, or in the case of reprographic reproduction in accordance with the terms of licences issued by the Copyright Licensing Agency. Enquiries concerning reproduction outside those terms and in other countries should be sent to the Rights Department, Oxford University Press, at the address above.

This book is sold subject to the condition that it shall not, by way of trade or otherwise, be lent, re-sold, hired out, or otherwise circulated without the publisher's prior consent in any form of binding or cover other than that in which it is published and without a similar condition including this condition being imposed on the subsequent purchaser.

A catalogue record for this book is available from the British Library

Library of Congress Cataloging in Publication Data

(Data available)

ISBN 0 19 853762 X

Typeset using LaTeX by Jane Spurr

Printed in Great Britain by
Biddles Ltd
Guildford and King's Lynn

Preface

We are happy to present Volume 3 of our Handbook project, on *Semantic Structures*. The previous two volumes presented the background on fundamental mathematical structures—consequence relations, model theory, recursion theory, category theory, universal algebra, topology, and on computational structures—term-rewriting systems, λ -calculi, modal and temporal logics and algorithmic proof systems. The computational structures considered thus far have been predominantly syntactic in character; while the discussion of mathematical structures has been quite general and free-standing.

In the present volume these threads are drawn together. We look at how mathematical structures are used to model computational processes. At first sight, this enterprise looks quite similar to the standard notions of model theory in logic. However, programming language semantics has significant special features and problems of its own, which have required an extensive development both of novel semantic techniques, and of the underlying mathematical foundations.

The first chapter of this volume concerns Domain Theory, a mathematical theory originated by Dana Scott to provide a mathematical foundation for the semantics of programming languages. In particular, Domain theory allows meaning to be given to *recursive definitions*, both of programs and of data types. In the 25 years since its inception, a very rich mathematical theory has been developed, and the chapter gives a systematic presentation of this theory.

The second Chapter concerns denotational semantics. The idea here is to assign meaning to programs as elements of, or functions between, suitable mathematical structures. Crucially, this assignment of meanings should be *compositional*; the meaning $[C(P_1, \dots, P_n)]$ of a complex program formed by applying some constructor C to the sub-programs P_1, \dots, P_n should be a function of the meaning of its parts:

$$[C(P_1, \dots, P_n)] = C([P_1], \dots, [P_n]),$$

where C is a suitable mathematical operation on meanings. In practice, the “suitable mathematical structures” will very often be domains; note also that the above equation can be read as saying that semantics is a *homomorphism*, thus invoking concepts of universal algebra. Chapter 2 on

denotational semantics sets out the general principles and techniques of compositional semantics, and shows how they can be applied to a range of programming language features.

Chapter 3 takes up the algebraic theme, focussing on how syntax can be viewed as a free algebra, and a semantics as another algebra, with the semantic algebra as a homomorphism between them (uniquely given because of the freeness of the syntactic algebra). For this programme to work properly it is necessary to combine ideas from Universal algebra (Volume 1) and Domain theory; ideas from category theory are also used extensively.

Finally, Chapter 4 concerns the semantics of types. Types are of increasing importance in modern programming languages. They provide a very effective conceptual discipline in program design, supported by efficient type-checking algorithms which allow many program errors to be detected by the compiler. A wide variety of type systems for programming languages have been considered, embracing such notions as higher-order and recursive types, polymorphism and subtyping. Chapter 4 provides an overview of such type systems and their semantics, which plays an essential role e.g. in establishing the soundness of type inference systems. For the very rich type systems currently being studied, many foundational problems are raised by the work on semantics, so this area shows an important interplay between theory and practice.

The Handbooks

The Handbook of Logic in Theoretical Computer Science and its companion, *The Handbook of Logic in Artificial Intelligence and Logic Programming*, have been created in response to a growing need for an in-depth survey of the application of logic in computer science and AI.

We see the creation of the Handbook as a combination of authoritative exposition, comprehensive survey, and fundamental research exploring the underlying unifying themes in the various areas. The intended audience is graduate students and researchers in the areas of computing and logic, as well as other people interested in the subject. We assume as background some mathematical sophistication. Much of the material will also be of interest to logicians and mathematicians.

The tables of contents of the volumes were finalized after extensive discussions between Handbook authors and second readers. The first two volumes present the Background—Mathematical Structures and Computational Structures.

The chapters, which in many cases are of monographic length and scope, are written with emphasis on possible unifying themes. The chapters have an overview, introduction, and main body. A final part is dedicated to more specialized topics.

Chapters are written by internationally renowned researchers in their

respective areas. The chapters are co-ordinated and their contents were discussed in joint meetings. Each chapter has been written using the following procedures:

1. A very detailed table of contents was discussed and co-ordinated at several meetings between authors and editors of related chapters. The discussion was in the form of a series of lectures by the authors. Once an agreement was reached on the detailed table of contents, the authors wrote a draft and sent it to the editors and to other related authors. For each chapter there is a second reader (the first reader is the author) whose job it has been to scrutinize the chapter together with the editors. The second reader's role is very important and has required effort and serious involvement with the authors.

Second readers for this volume are:

Chapter 1: Domain Theory—R Heckman

Chapter 2: Denotational Semantics—C Wadsworth

Chapter 3: Algebraic Semantics—I Guessarian

Chapter 4: Semantics of Types—R Crole

2. Once this process was completed (i.e. drafts seen and read by a large enough group of authors), there were other meetings on several chapters in which authors lectured on their chapters and faced the criticism of the editors and audience. The final drafts were prepared after these meetings.
3. We attached great importance to group effort and co-ordination in the writing of chapters. The first two parts of each chapter, namely the introduction-overview and main body, are not completely under the discretion of the author, as he/she had to face the general criticism of all the other authors. Only the third part of the chapter is entirely for the authors' own personal contribution.

The Handbook meetings were generously financed by OUP, by SERC contract SO/809/86, by the Department of Computing at Imperial College, and by several anonymous private donations.

We would like to thank our colleagues, authors, second readers, and students for their effort and professionalism in producing the manuscripts for the Handbook. We would particularly like to thank the staff of OUP for their continued and enthusiastic support, and Mrs Jane Spurr, our OUP Administrator, for her dedication and efficiency.

London
April 1994

S. Abramsky and D. M. Gabbay

Contents

List of contributors

xv

Domain theory

Samson Abramsky and Achim Jung

1	Introduction and Overview	2
1.1	Origins	2
1.2	Our approach	4
1.3	Overview	5
2	Domains individually	6
2.1	Convergence	6
2.2	Approximation	15
2.3	Topology	27
3	Domains collectively	32
3.1	Comparing domains	32
3.2	Finitary constructions	39
3.3	Infinitary constructions	44
4	Cartesian closed categories of domains	52
4.1	Local uniqueness: Lattice-like domains	54
4.2	Finite choice: Compact domains	55
4.3	The hierarchy of categories of domains	62
5	Recursive domain equations	66
5.1	Examples	67
5.2	Construction of solutions	69
5.3	Canonicity	74
5.4	Analysis of solutions	79
6	Equational theories	84
6.1	General techniques	85
6.2	Powderdomains	94
7	Domains and logic	107
7.1	Stone duality	108
7.2	Some equivalences	114
7.3	The logical viewpoint	124
8	Further directions	148
8.1	Further topics in 'classical domain theory'	148
8.2	Stability and sequentiality	151
8.3	Reformulations of domain theory	152

8.4	Axiomatic domain theory	154
8.5	Synthetic domain theory	155
9	Guide to the literature	156
Denotational semantics		169
<i>R. D. Tennent</i>		
1	Introduction	170
1.1	Approaches	171
1.2	An example: binary numerals	172
1.3	Compositionality	175
1.4	Criteria	176
1.5	Overview	178
1.6	Bibliographic notes	179
2	A simple imperative language	180
2.1	Expressions and commands	180
2.2	Assignment commands	185
2.3	Indefinite iterations	187
2.4	Programs	191
2.5	Operational semantics	192
2.6	Programming logic	196
2.7	Non-determinism	203
2.8	Bibliographic notes	205
3	A simple applicative language	205
3.1	Definitions and function applications	206
3.2	Function definitions	212
3.3	Defined notation	214
3.4	Elementary properties	217
3.5	Programming logic	220
3.6	Bibliographic notes	228
4	Recursion	228
4.1	Recursive definitions	228
4.2	Domain-theoretic semantics	231
4.3	Operational semantics	237
4.4	Programming logic	239
4.5	Full abstraction	242
4.6	Untyped procedures	243
4.7	Bibliographic notes	245
5	An Algol-like language I	245
5.1	Syntax	246
5.2	Semantics	250
5.3	Call by value	253
5.4	Programming logic	256

5.5	Bibliographic notes	260
6	An Algol-like language II	260
6.1	Coercions	261
6.2	Local variables	268
6.3	Product types and arrays	270
6.4	Lists	274
6.5	Acceptors	279
6.6	Jumps	282
6.7	Intermediate output	287
6.8	Block expressions	288
6.9	Bibliographic notes	289
7	Possible worlds	290
7.1	Functor–category semantics	291
7.2	Semantic-domain functors	292
7.3	Semantic valuations	295
7.4	Semantics of local variables	298
7.5	Specifications	302
7.6	Non-interference specifications	304
7.7	Semantics of block expressions	308
7.8	Bibliographic notes	311

Algebraic semantics

323

Eric G. Wagner

1	Introduction and motivation	324
1.1	General remarks	324
1.2	Some motivating examples	326
1.3	A notational ‘crisis’	332
2	Algebraic theories, definitions and examples	332
2.1	General remarks	332
2.2	Basic definitions	332
2.3	Algebraic theories as categories	336
2.4	Examples of algebraic theories	337
2.5	Notations and basic identities	341
3	Ordered theories	343
3.1	Definition of ordered and continuous theories	343
3.2	Examples of ordered theories	344
4	Theories with iteration operators	348
4.1	Iteration operators	348
4.2	Iteration, rational, and iterative theories	349
4.3	The ∇ -operator	356
4.4	Iteration operators and flowcharts	356

4.5	Interpretations of flowcharts	364
5	More about iteration operators	366
5.1	Relationships between iteration, rational, and iterative theories	366
5.2	Some identities for iteration operators	372
6	Iteration closure, and normal form theorems	374
7	Free theories and Herbrand interpretations	378
7.1	Free theories	379
7.2	The general case	380
8	Recursive hierarchies	386

The semantics of types in programming languages 395

Carl A. Gunter

1	Introduction	396
2	Types in programming	397
2.1	Higher types	397
2.2	Recursive types	400
2.3	Parametric polymorphism	402
2.4	Subtypes	404
3	Simple types as sets	408
3.1	Types and equations	409
3.2	Sets as a model	412
3.3	Type frames	415
3.4	Completeness for sets	418
4	Simple types as domains	420
4.1	A programming language for computable functions	420
4.2	Operational semantics	422
4.3	Operational equivalence	425
4.4	bc-domains and dl-domains	426
4.5	Full abstraction	427
5	Types as invariants	430
5.1	Run-time safety	430
5.2	Implicit types	434
5.3	Run-time safety for assignments and continuations	441
6	Types as subsets	446
6.1	Untyped λ -calculus	446
6.2	What is a model of the untyped λ -calculus?	448
6.3	What models of the untyped λ -calculus are there?	449
6.4	Inclusive subsets as types	451
6.5	Subtyping as subset inclusion	455
7	Types as partial equivalence relations	458
7.1	Sets as a model of ML_0 types	458

7.2	Another typing system for ML_0	460
7.3	The polymorphic λ -calculus	461
7.4	Sets as a model of polymorphic types?	464
7.5	Simple types as PERs	466
7.6	PERs as a model of polymorphic types	468
8	Conclusion	470
	Index	477