

Preface

As its title promises, this book provides foundations for software specification and formal software development from the perspective of work on algebraic specification. It concentrates on developing basic concepts and studying their fundamental properties rather than on demonstrating how these concepts may be used in the practice of software construction, which is a separate topic.

The foundations are built on a solid mathematical basis, using elements of universal algebra, category theory and logic. This mathematical toolbox provides a convenient language for precisely formulating the concepts involved in software specification and development. Once formally defined, these notions become subject to mathematical investigation in their own right. The interplay between mathematics and software engineering yields results that are mathematically interesting, conceptually revealing, and practically useful, as we try to show.

Some of the key questions that we address are: What is a specification? What does a specification mean? When does a software system satisfy a specification? When does a specification guarantee a property that it does not state explicitly? How does one prove this? How are specifications structured? How does the structure of specifications relate to the modular structure of software systems? When does one specification correctly refine another specification? How does one prove correctness of refinement steps? When can refinement steps be composed? What is the role of information hiding? We offer answers that are simple, elegant and general while at the same time reflecting software engineering principles.

The theory we present has its origins in work on algebraic specifications starting in the early 1970s. We depart from and go far beyond this starting point in order to overcome its limitations, retaining two prominent characteristics.

The first is the use of many-sorted algebras consisting of a collection of sets of data values together with functions over those sets, or similar structures, as models of software systems. This level of abstraction fits with the view that the correctness of the input/output behaviour of a software system takes precedence over all its other properties. Certain fundamental software engineering concepts, such as information hiding, have direct counterparts on the level of such models.

The second is the use of logical axioms, usually in a logical system in which equality has a prominent role, to describe the properties that the functions are required to satisfy. This property-oriented approach allows the use of formal systems of rules to reason about specifications and the relationship between specifications and software systems. Still, the theory we present is semantics-oriented, regarding models as representations of reality. The level of syntax and its manipulation, including axioms and formal proof rules, merely provide convenient means of dealing with properties of (classes of) such models.

Our primary source of software engineering intuition is the relatively simple world of first-order functional programming, and in particular functional programming with modules as in Standard ML. It is simpler than most other programming paradigms, it offers the most straightforward fit with the kinds of models we use, and it provides syntax (“functors”) that directly supports a methodology of software development by stepwise refinement. Even though some aspects of more elaborate programming paradigms are not directly reflected, the fundamental concepts we study are universal and are relevant in such contexts as well.

This book contains five kinds of material.

The requisite mathematical underpinnings:

Chapters 1 and 3 are devoted to the basic concepts of universal algebra and category theory, respectively. This material finds application in many different areas of theoretical computer science and these chapters may be independently used for teaching these subjects. Our aim is to provide a generally accessible summary rather than an expository introduction. We omit many standard concepts and results that are not needed for our purposes and include refinements to classical universal algebra that are required for its use in modelling software. Most of the proofs are left to the reader as exercises.

Traditional algebraic specifications:

Chapter 2 presents the standard material that forms the basis of work on algebraic specifications. From the point of view of an algebraist, much of this would be viewed as part of universal algebra. Additionally, Section 2.7 explores some of the ways in which these basics may be modified to cope with different aspects of software systems. Again, this chapter is a summary rather than an expository introduction, and many proofs are omitted.

Elements of the theory of institutions:

In Chapter 4 we introduce the notion of an *institution*, developed as a formalisation of the concept of a logical system. This provides a suitable basis for a general theory of formal software specification and development. Chapter 10 contains some more advanced developments in the theory of institutions.

Formal specification and development:

Chapters 5–8 constitute the core of this book. Chapter 5 develops a theory of specification in an arbitrary institution. Special attention is paid to the issue of structure in specifications. Chapter 6 is devoted to the topic of parameterisation, both of algebras and of specifications themselves. Chapter 7 presents a theory of formal software development by stepwise refinement of specifications. Chapter 8

introduces the concept of *behavioural equivalence* and studies its role in software specification and development.

Proof methods:

Chapter 9 complements the model-theoretic picture from the previous chapters by giving the corresponding proof methods, including calculi for proving consequences of specifications and correctness of refinement steps.

The dependency between chapters and sections is more or less linear, except that Chapter 10 does not depend on Chapter 9. This dependency is not at all strict. This is particularly relevant to Chapter 3 on category theory: anyone who is familiar with the concepts of category, functor and pushout may omit this chapter, returning to it if necessary to follow some of the details of later chapters. On first reading one may safely omit the following sections, which are peripheral to the main topic of the book or contain particularly advanced or speculative material: 2.6, 2.7, 3.5 except for 3.5.1, 4.1.2, 4.4.2, 4.5, 6.3, 6.4, 6.5, 8.2.3, 8.5.3, 9.5, 9.6 and Chapter 10.

This book is self-contained, although mathematical maturity and some acquaintance with the problems of software engineering would be an advantage. In the mathematical material, we assume a very basic knowledge of set theory (set, membership, Cartesian product, function, etc. — see for instance [Hal70]), but we recall all of the set-theoretic notation we use in Section 1.1. Likewise, we assume a basic knowledge of the notation and concepts of first-order logic and proof calculi; see for instance [End72]. In the examples that directly relate to programming, we assume some acquaintance with simple concepts of functional programming. No advanced features are used and so these examples should be self-explanatory to anyone with experience using a programming language with types and recursion.

In an attempt to give a complete treatment of the topics covered without going on at much greater length, quite a few important results are relegated to exercises with the details left for the reader to fill in. Fairly detailed hints are provided in many cases, and in the subsequent text there is no dependence on details of the solutions that are not explicitly given in these hints.

This book is primarily a monograph, with researchers and advanced students as its target audience. Even though it is not intended as a textbook, we have successfully used some parts of it for teaching, as follows:

Universal algebra and category theory:

A one-semester course based on Chapters 1 and 3.

Basic algebraic specifications:

A one-semester course for undergraduates based on Chapters 1 and 2.

Advanced algebraic specifications:

An advanced course that follows on from the one above based on Chapters 4–7.

Institutions:

A graduate course with follow-up seminar on abstract model theory based on most of Chapter 4 and parts of Chapter 10.

The material in this book has roots in the work of the entire algebraic specification community. The basis for the core chapters is our own research papers, which are here expanded, unified and taken further. We attempt to indicate the origins of

the most important concepts and results, and to provide appropriate bibliographical references and pointers to further reading, in the final section of each chapter. The literature on algebraic specification and related topics is vast, and we make no claim of completeness. We apologize in advance for possible omissions and misattributions.

Acknowledgements

All of this material has been used in some form in courses at the University of Edinburgh, the University of Warsaw, and elsewhere, including summer schools and industrially oriented training courses. We are grateful to all of our students in these courses for their attention and feedback.

This book was written while we were employed by the University of Edinburgh, the University of Warsaw, and the Institute of Computer Science of the Polish Academy of Sciences. We are grateful to our colleagues there for numerous discussions and for the atmosphere and facilities which supported our work. The underlying research and travel was partly supported by grants from the British Council, the Committee for Scientific Research (Poland), the Engineering and Physical Sciences Research Council (UK), the European Commission, the Ministry of Science and Higher Education (Poland), the Scottish Informatics and Computer Science Alliance and the Wolfson Foundation.

We are grateful to the entire algebraic specification community, which has provided much intellectual stimulation and feedback over the years. We will not attempt to list the numerous members of that community who have been particularly influential on our thinking, but we give special credit to our closest collaborators on these topics, and in particular to Michel Bidoit, Till Mossakowski and Martin Wirsing. Our Ph.D. students contributed to the development of our ideas on some of the topics here, and we particularly acknowledge the contributions of David Aspinall, Tomasz Borzyszkowski, Jordi Farrés-Casals and Wiesław Pawłowski.

We are grateful for discussion and helpful comments on the material in this book and the research on which it is based. In addition to the people mentioned above, we would like to acknowledge Jiří Adámek, Jorge Adriano Branco Aires, Thorsten Altenkirch, Egidio Astesiano, Hubert Baumeister, Jan Bergstra, Pascal Bernard, Gilles Bernot, Didier Bert, Julian Bradfield, Victoria Cengarle, Maura Cerioli, Rocco De Nicola, Răzvan Diaconescu, Luis Dominguez, Hans-Dieter Ehrich, Hartmut Ehrig, John Fitzgerald, Michael Fourman, Harald Ganzinger, Marie-Claude Gaudel, Leslie Ann Goldberg, Joseph Goguen, Jo Erskine Hannay, Robert Harley, Bob Harper, Rolf Hennicker, Claudio Hermida, Piotr Hoffman, Martin Hofmann, Furio Honsell, Cliff Jones, Jan Jürjens, Shin-ya Katsumata, Ed Kazmierczak, Yoshiki Kinoshita, Spyros Kominos, Bernd Krieg-Brückner, Sławomir Lasota, Jacek Leszczyłowski, John Longley, David MacQueen, Tom Maibaum, Lambert Meertens, José Meseguer, Robin Milner, Eugenio Moggi, Bernhard Möller, Brian Monahan, Peter Mosses, Tobias Nipkow, Fernando Orejas, Marius Petria, Gordon Plotkin, Axel Poigné,

John Power, Horst Reichel, Grigore Roşu, David Rydeheard, Oliver Schoett, Lutz Schröder, Douglas Smith, Stefan Sokołowski, Thomas Streicher, Eric Wagner, Lincoln Wallen and Marek Zawadowski. We apologize for any omissions. We are grateful to Stefan Kahrs, Bartek Klin and Till Mossakowski for their thoughtful and detailed comments on a nearly final version which led to many improvements, to Mihai Codescu for helpfully checking examples for errors, to Ronan Nugent of Springer and to Springer's copyeditor.

Finally, we would like to express our very special appreciation to Rod Burstall and Andrzej Blikle, our teachers, supervisors, and friends, who introduced us to this exciting area, brought us to scientific maturity, and generously supported us in our early careers.

Edinburgh and Warsaw,
September 2011

Don Sannella
Andrzej Tarlecki



<http://www.springer.com/978-3-642-17335-6>

Foundations of Algebraic Specification and Formal
Software Development

Sannella, D.; Tarlecki, A.

2012, XVI, 584 p., Hardcover

ISBN: 978-3-642-17335-6