Overall, this book contains a lot of valuable material, and due to the simplicity of the Event-B notation it should be a seriously considered for introductory courses on formal modelling with associated proof. As a basis for advanced study and research, it would be significantly better if it contained more conceptual clarity and methodological guidance on refinement, as well as a much more extensive index and bibliographic information. The case studies, however, still provide a rich source of example material for researchers.

References

Eerke Boiten (2011) *Perspicuity and granularity in refinement*, Refinement Workshop, EPTCS **55**, pp 155–165. DOI: 10.4204/EPTCS.55.10

Stefan Hallerstede (2009) *Proving quicksort correct in Event-B*, Refinement Workshop, ENTCS **259**, pp 47–65, DOI: 10.1016/j.entcs.2009.12.017

Steve Schneider, Helen Treharne & Heike Wehrheim (2011) *A CSP account of Event-B refinement*, Refinement Workshop, EPTCS **55**, pp 139–154. DOI: 10.4204/EPTCS.55.9

EERKE BOITEN
School of Computing, University of Kent, Canterbury, UK

Drawing Programs:
The Theory and Practice of Schematic Functional Programming, by Tom Addis and Jan Addis Springer, 2010, ISBN 978-1-84882-617-5, 379pp
doi:10.1017/S095679681200010X

The book presents the notion of schematic functional programming and demonstrates not only the concept but also how schematic functional programs can be processed, and how this approach can be used to develop small and even complicated programs.

A reader need not be familiar with the concept of functional programming, but on the other hand, it is probably expected that a reader is familiar with programming and programming languages. The development environment connected with schematic programming can be found on Internet and downloaded (but only for Windows 32bit systems) but it seems not to have been under active development for several years. Nevertheless, a reader can easily try the examples and follow the flow of the book.

Sometimes it is necessary to have broader knowledge to understand the examples presented, as they are built over problems from various areas of computer processing. Even when the book presents introduction to such areas (e.g. to Bayesian theory), it is not sufficient to fully understand the solution. Thus, a question comes to one's mind, whether such an introduction is necessary, as it is of little use for those who are familiar with the subject, but not adequate for those who do not know the subject. A reader familiar with these areas can go through the book quite easily and try to use the constructs and typical patterns of schematic programming, and, in such a way, learn how to use schematic programming and become familiar with it.

It is quite misleading to say that this is a functional language. It would be very helpful if the authors could give a clear explanation of how it is (and isn't) functional, so that a reader could recognize that it is an imperative language with stateful development and runtime environment, but that the schematic language itself exploits functions.

This also raises another issue – the theory behind the schematic programming is not presented at all. This includes no binding to any calculus, no presentation of any formal system that could be used as a formal basis of the schematic language or the underlying language. Thus, the 'theory' in the book title may be misleading.

Reviewing the chapters in more detail, the first chapter presents an introduction to schematic functional programming and also introduces the development environment, Clarity, used for schematic program development and evaluation. A reader can easily get to understand the very first constructs of the language and usage of the Clarity. Nevertheless, understanding is not completely straightforward as some presented screenshots do not match those that users can see for themselves; this especially applies to those showing the Clarity environment and database manipulation, rather than those of the basic schema. Some errors also occur in text (page 14: reference to page XX – what is XX?), and it is frustrating when the text refers to colors in a black and white book.

Chapter 2 further extends the presentation of schematic programming possibilities, especially the modification of programs, when necessary, and basic manipulation of lists as a built-in structured data type. So far the topic is presented well and can be followed easily. Drawbacks of the chapter are the pictures and screenshots: some of them are quite large even if presenting little information (see for instance page 68) on the other hand some are shrunk and of poor quality (page 69). Chapter 3 discusses ways of defining functions, how to reference parameters, and so on. Even if logically presented well, the pictures in it substantially mar the presentation of the chapter.

Chapter 4 goes even deeper into schema possibilities in function definitions before presenting pattern matching in the schematic programming. To support the way that pattern matching is implemented in Clarity, the chapter provides some material about it that is quite difficult to follow, especially presented tables of results – the presented results do not seem to be consistent among themselves. Moreover, this is the point where the first typos appear in the text. Last but not least, the quality and presentation level of pictures in the chapter is again poor.

Chapter 5 introduces some principles of function evaluation in the context of an introduction to functional programming techniques. Part of the chapter is a substantial project; indeed, projects are part of other previous chapters too, but in this case it is first worth noting; exercises are also an essential part of every chapter. The logic of the chapter is clear, but its larger part is the project. Thus, splitting it or presenting more functional programming techniques would be better.

In Chapter 6, we meet the language Faith, to which the schematic programs of Clarity are translated and then evaluated. Evaluation in the underlying interpreter of language Faith is demonstrated here, and some schema constructs are translated to it. Next, higher order functions are presented in some examples. Then, another large project is presented. A drawback here is the function makelist (diagram 6.6, page 210), a definition of which was not presented in the book so far. The curiosity of the function makelist is that it can accept arbitrary numbers of parameters, while other functions, especially the user defined ones, cannot. Even if the feature of function makelist is mentioned in the text, it is not explained why it is so.

Chapter 7 addresses side effects, which gives the language an imperative aspect. The chapter presents other aspects of Faith: we learn how to define relations, sequential evaluation, and, at the chapter end, a large project consisting of a "General Problem Solver" is presented together with necessary background on the subject. Even if exploitation of imperative features spoils the concept, the chapter and its project are interesting from other viewpoints.

Chapter 8 is mixture of additional features, including: some features of built-in libraries for GUI programming, manipulation of program schema file operations, importing the Faith code, and others. A brief introduction to Bayesian theory is presented together with small example – it is a little extension of what we can found on Wikipedia by these days, aiming at Bayesian filtering (we know from email clients). The project of this chapter demonstrates inlining of Faith code in Clarity schema. This chapter probably completes the description of the features of the Clarity environment; otherwise its importance is low. The crucial fault of this chapter is the equation (4) on page 284 presenting Bayes' theorem, which is wrong – conditional probability is dependent on itself, the two identifiers should be swapped in the equation.

Chapter 9 is also a mixture of various topics. It introduces some builtin functions for low-level manipulation of functions defined by enumeration, etc. Those readers familiar with Prolog may recognize some names and behavior. Next, interconnection and remote processing in Clarity is presented. Moreover, possibility of interfacing the Clarity, namely via Pascal/Delphi is demonstrated. Finally, a quite substantial Sudoku solver is presented as a project. This project is full of screenshots, but contains little by way of explanation

Finally, Chapter 10 extends the Sudoku solver with an uncertainty model. To present this, the chapter contains several theoretical parts to present some parts of probability theory, game theory, and others. At the end of the chapter, a simple learning system based on Bayesian networks is presented. Examples and usage of Clarity in this chapter are quite rare. Thus, overall contribution of this chapter (and the previous one) to understand the concept of schematic programming is low.

The book has three appendices, covering a BNF grammar for Faith, its extension to Clarity and the structure of .seg file, which is natural for Clarity.

In summary, the book Drawing Programs: The Theory and Practice of Schematic Functional Programming cannot address those who are programmers by nature or by job. Nor is not for beginners either. Nevertheless, it can address those who can exploit algorithms and computers in general, but probably not as a first step to professional programming. It provides a nice introduction to all constructs of the schematic programming and manipulation of schema. Moreover, a skilled programmer using several programming paradigms can get through the notion of the book in a short time.

A drawback of the book is its inconsistency – it is built from several articles and the binding between them is quite sparse, especially in the last chapters. Moreover, text uses references and description based on colorful pictures, but the book contains only pictures in shades of gray. Thus, the readability is low in some places. Pictures (quality, size, readability, etc.) are the poorest part of the book at all. Last, but not least, the last chapters present "hacks" inside Clarity/Faith to achieve some extra features, nevertheless, such features are not natural for schematic programming and undermine the initial positive view of the nature of schematic programming gained from the first few chapters.

DUŠAN KOLÁŘ
Department of Information Systems, Faculty of Information Technology,
Technical University of Brno, Božetěchova 2, 612 66, Brno Czech Republic

https://doi.org/10.1017/S095679681200010X Published online by Cambridge University Press