J. Functional Programming **6** (2): 375–377, March 1996

Book review

An Introduction to Formal Specification and Z by Ben Potter, Jane Sinclair and David Till, Prentice-Hall, 1991.

Z: An Introduction to Formal Methods (second edition) by Antoni Diller, John Wiley & Sons, 1994.

Z is a specification language invented by Abrial and developed at the Oxford Programming Research Group (PRG). The intention is that Z specifications are written as part of the requirements analysis and system specification stages of the software lifecycle. Bugs arising from imprecision at these early stages can be very costly to fix if only discovered late on. Hence, the argument goes, although the formality of Z may make the initial stages lengthier than if natural language specifications were used, Z is more likely to catch design bugs early on. Overall, software development costs are reduced. Indeed, joint work between the PRG and IBM's Hursley Lab and INMOS shows that Z can reduce development costs in industrial settings. Z, like functional programming, is an attempt to use an abstract, formal notation to produce better software than otherwise would be possible. Diller's book makes the connection explicit by animating a Z specification using Miranda.†

Z is already documented by Spivey's *The Z Notation: A Reference Manual* and Hayes' *Specification Case Studies*. Spivey's book is regarded by both the reviewed books as the effective Z standard. Hayes' book describes many examples of how Z has been effectively used in practice. Both Spivey and Hayes have recently produced second editions. Potter *et al.* was written before these new editions were available. Diller has produced a second edition, to reflect the changes made in the second edition of Spivey. As far as I know there have been no fundamental changes in Spivey, so Potter *et al.* is still suitable for teaching Z. You should be aware, though, that some details may be out of date.

Both of the books reviewed include an introduction to Z specifications that complements the more advanced material in Spivey and Hayes. They both cover the basics of specification in Z: first-order logic, typed set theory, relations and functions, sequences and schemas. They both recommend a structure for presenting complete Z specifications. In addition to this core material, Diller includes several case-studies, an animation of a Z specification using Miranda, and material on Floyd-Hoare logic and refinement. In contrast, Potter *et al.* point the reader to Spivey and Hayes for supplementary examples and reference material. They devote a chapter to locating Z specification in the software lifecycle, and another to discussing the pros and cons of real-world Z projects and their relation to other software development methodologies. They include notes for new users and discussion of two of the major Z applications, the CICS product at IBM Husley and the floating point unit of the INMOS transputer. Potter *et al.* include some material on reasoning and refinement, but make clear their primary aim is to teach software specification.

Both books include many bibliographic citations for further study. Some of Diller's, such as the citation regarding 'deviant' logic on the opening page, are perhaps out of place in an introductory textbook. Potter *et al.* have exercises covering all the technical material they introduce. Diller has fewer exercises, mainly on the core material, but includes answers.

† Miranda is a trademark of Research Software Ltd.

Diller treats more examples than Potter *et al.*. He includes versions of well-known specification examples such as Sufrin's text editor and Wing's library database. In both books, all the examples have the familiar form of a global state, together with operations to transform it and perhaps input or output some data. One of the most plausible ideas of object-oriented programming is that use of local states leads to clearer programs than a single global state. Apart from a passing reference in Potter *et al.* to using CSP to model a system as a collection of communicating processes, neither book discusses the advantages of using local state to structure a specification. The use of global state seems to be standard practice in the Z community, but even so, I would like to have seen a discussion of the issue in both books.

The strength of Diller's book is that the one volume contains introductory material, many case-studies and a reference manual. Although the core material is a good introduction to Z, the book as a whole is not such a good introduction to formal methods. Although Diller devotes more space to discussing how to reason about and refine specifications than do Potter *et al.*, he includes no guidance on when and how to apply Z. This is a pity because, as Potter *et al.* point out, formal methods projects can fail, and new users should be aware of the pitfalls. Diller includes a summary of the key features of Z, but it is misleading to call it a 'reference manual' as so many details, including syntax, are left undefined. To be fair to Diller, Z has historically only been loosely defined. Nonetheless, his reference manual falls short of the standard set by Spivey.

There are new chapters in Diller on Floyd-Hoare logic and its use in verifying an imperative implementation of a Z specification. In the absence of mechanised tools, it would be forbidding to teach from these chapters, which have no accompanying exercises. But they do pose an interesting research question: what tools could support this process?

Both books run the risk that none of the specified systems is any more complex than examples found in introductory programming textbooks. Students will wonder what is the point of specifying them in Z. After all, a Z specification is not far from being a functional program. Proofs of functional programs, like proofs about Z specifications, are typically carried out informally, but rigorously, on paper. But there are at least two advantages of specifications that are not executable functional programs. Sometimes it is useful to specify a program as a relation, to avoid 'over-specification'. Sometimes a program may usefully be specified as the inverse of a function (parsing, say, can be specified as the inverse of printing). In my view, these possibilities are surprisingly rare. Executable functional programs are good for many specifications, though a few specifications are much clearer if they are not functional programs. It is a pity that neither of these books explains where in their examples they exploit non-executable specification.

Diller's section on 'Specification Animation' shows how a Miranda script can be derived from the specification of the telephone directory. The code in the script corresponds closely to a collection of Z schemas. It uses combinators from Bird and Wadler's *Introduction to Functional Programming*, perhaps already familiar to students. It is not optimised – it re-runs an invariant checker after each operation. This connection between Z and functional programming is a good start, but I would like to have seen the study taken further. As there are no exercises, it would be hard to use the chapter for teaching. Diller does not discuss the formal relationship between Miranda and Z. We have no guarantee that the Miranda program accurately animates the Z specification. What the chapter does show is how a Miranda interpreter could exercise and debug a Z specification. But what was the point of beginning the process in Z at all? With probably less work we could have specified the telephone directory directly as an executable Miranda script.

In summary, both books cover the basics of Z and include many case studies and hence would usefully accompany a course on formal specification. My preference is for Potter *et al.* Their book covers the basics well – without excess of detail and with exercises throughout – and places Z in the context of software engineering. Diller covers the basic material reasonably well, and includes many case studies, but he fails to discuss the role of Z in practice. His more advanced chapters are let down by lack of exercises, perhaps because they are still

research topics. Diller's chapter on animating Z specifications using Miranda is of interest to functional programmers, but the connection and its implications are not pushed far enough to be very useful.

ANDREW D. GORDON
Computer Laboratory
University of Cambridge