Elementary Standard ML by Greg Michaelson
University College London Press, UK, 1995, ISBN 1-85728-398-8 PB, pp 306

Standard ML is a functional programming language that is often used for teaching the principles of programming to first year Computer Science students. Often the teacher of a functional programming language has to be like a doctor: The medicine is bitter but he says that it is good for you. Most students may know some C, Pascal or Basic, and they will find a functional language alien, dislikable or even useless. It is quite a challenge to write a book that will appeal to such students.

Elementary Standard ML tries to meet the challenge by approaching the subject in a way that is a little different from that chosen by the classical texts (for example [1, 2]). Elementary Standard ML does not make heavy use of mathematics as many of the competitors do. At first I thought that this would be a missed opportunity but after having read the book, I felt positive about it. Michaelson presents the important concepts in a way that is as simple as possible. In spite of this, rigour is not sacrificed: The book contains a large number of derivations that show step-by-step how an expression is evaluated to its result. The derivations are presented in a natural term rewriting style without betraying their mathematical origins. Theorems and proofs would be out of place in Elementary Standard ML, and they are thus not present.

The second aspect that I liked about the book is the modelling approach to problem solving as described in the first chapter. The fundamental idea that a problem can be solved by decomposing it into more manageable sub-problems is nicely explained. The discussion then naturally leads on to presenting the concept of a type as a means of conveying information about the solutions of problems and to maintain the consistency of solutions.

The second chapter discusses basic types, as they form the foundations of both programming in a strongly typed languages (SML) and problem solving. The SML basic types are discussed in rather a lot of detail. This is hard to avoid, mainly because SML does not support overloading properly.

The third chapter gives an in depth discussion of functions over values of basic types. The separation of the concepts of naming and abstraction is nicely presented, even though the notation provided by SML for this separation is a bit awkward. In between the lines I read the essence of the λ-calculus, but Michaelson has skillfully avoided the mathematical notation (which would put students off completely), the intricacies of β-conversion, and even difficult words and definitions. The computational model for SML is nevertheless developed in a highly effective way. I regard this as a strong pedagogical feature of the book.

Chapter 4 discusses more advanced use of functions, using pattern matching and recursion. Here I felt that more emphasis should have been put on the case analysis that precedes the creation of functions. Important issues such as 'should all cases be disjoint', and 'should all cases be covered' and 'what to do when not all cases are covered' are unfortunately not discussed systematically.

Chapter 5 discusses lists in a rather operational fashion. This did not appeal to me, but Michaelson also gives the usual inductive definition of a list (using plain English). The computational model introduced earlier is a bit too simple to cope well with pattern matching, but the presentation used in the book is acceptable.

Chapter 6 discusses higher order functions, mainly operating on lists. Many of the examples here use accumulating arguments which seems a little artificial. Neater solutions are available to many of the problems presented. These solutions are discussed later, when `let` expressions are introduced.

Chapter 7 discusses the combination of using lists and tuples, to create the data structures that were called for in the analyses of the first chapter. We are now on page 155, so the reader has to go through a lot of detail before the problems discussed in the first chapter can finally be solved. This is the weakest point of the book. I would have welcomed a stronger connection between the study of the SML language, and problem solving. Some of the solutions in chapter 7 are a bit *ad hoc*, where combinations of `map`, `filter`, `foldr` and/or `foldl` would have been appropriate. I realise that students dislike `foldl` and `foldr`, but `map` and `filter` are generally appreciated.

Chapter 8 puts the material of the preceding 7 chapters into practice by building a collection of text processing functions. Here the detailed step-by-step derivations gets a bit long winded; perhaps the student might at this stage be able to fill in some intermediate steps. SML does not treat a list of characters as a string, hence the functions in this chapter make profligate use of `explode` and `implode`. This is unpleasant but alas unavoidable.

Chapter 9 introduces algebraic data types. My only worry is that the running example here (lexical analysis) is not explained as clearly as it *could* have been. Some pictures of the finite state machines involved would have

been useful.

Chapter 10 generalises lists to trees, using the algebraic data types of the previous chapter. The running example here is the construction of a parser for a simple language. This is a challenge to the first year student, but it is well explained. In this chapter we encounter some genuine mathematics when discussing the relative complexity of lists and trees. The students should probably have heard about complexity before to fully appreciate this.

Chapter 11 mainly discusses some aspects of the (impure) I/O system of SML. It also puts the components of the previous chapters together by building an interpreter for a simple language. This is a nice example of how to build a system out of smaller components. This would have been an ideal opportunity to introduce at least modules, so as the wrap up the components and make the problem decomposition more effective.

Chapter 12 continues to explore the remaining impure facilities of SML. It is rather unfortunate that abstract data types, structures and signatures are discussed only briefly. The module system of SML is one of its strongest points, and even in a first programming course modules can be used. Students should also be introduced to the concept of information hiding.

Appendices are provided that explain how to use an SML system, what the syntax of SML is, and what library functions can be used. This is all useful information. The book also contains many worked examples.

Most aspects of software engineering that apply to programming in the small are addressed in the book, but there is one topic that is a bit lightly treated: Testing. Most other books on functional programming would devote considerable effort to proving functions correct with respect to some specification. Such books would not devote much effort to testing. Elementary standard SML does not use mathematics, so the void should have been filled by more thorough testing. Most chapters do contain a section at the end that mentions testing, but not in sufficient depth.

In one respect, the suppression of mathematics in favour of clear, everyday language has gone too far, and rather unnecessarily so. In many places, the types of functions and expressions are analysed informally so that the student will learn how to analyse the type of an expression and its constituent parts. The rules for making such an analysis are never given explicitly. I am not suggesting that there should be a type inference algorithm on the basis of unification. I think that a case analysis style, in much the same way as for example the inductive definition of a list in plain English, could have been provided, thus guiding quite a lot of the discussions on types. There are other opportunities where a simple set of rules could have been given in plain English, that would have avoided some potential ambiguities due to insufficient explanation.

Summarising, I consider Elementary Standard ML a good text for teaching a functional language as the first programming language. I would not use the book if the students can be expected to have a reasonable amount of mathematical maturity.

Reviewed by Pieter H. Hartel, University of Southampton, UK

References

[1] R. S. Bird and P. L. Wadler. *Introduction to functional programming.* Prentice Hall, New York, 1988.

[2] L. C. Paulson. *ML for the working programmer.* Cambridge Univ. Press, New York, 1991.