TLP **17** (1): 118–120, 2017. © Cambridge University Press 2016
doi:10.1017/S1471068416000181

Book review

Constraint Solving and Planning with Picat by Neng-Fa Zhou, Håkan Kjellerstrand, and Jonathan Fruhman, xi + 148 pages, Springer, 2015. Paperback, ISBN 978-3-319-25881-2.

Picat is a novel multi-paradigm programming language with strong Prolog roots. This book is a brief guideline to using Picat for solving combinatorial optimization and planning problems. It is written by a group of authors who are probably the most competent for producing such a book as they have been involved in the development of Picat and wrote a lot of programs in this language. Nevertheless, do not expect many technical details about the insides of Picat; the purpose of this book is different. By using many practical examples, the book shows how to effectively and efficiently encode well-known and less-known problems in Picat. It can be seen as a textbook on Picat programming and the readers might be surprised, as I was when I was first exposed to Picat, how compact and elegant the code could be even for non-trivial problems.

The book is published in the series Springer Briefs in Intelligent Systems so it is not long, less than 150 pages, and can be read in a short time. If you are a Prolog programmer, you will feel soon familiar with the presented codes. If you are programming in other languages, the surprise from elegance of presented encodings might be even higher. In seven chapters, you will find a lot of examples of small programs solving specific problems. So let us look now in more detail, what the reader can take away from this book.

Chapter 1 is a must-have reading for everyone who does not know Picat yet. It explains the core design principles behind Picat, using examples, it demonstrates the syntax of the language and available data structures, and it also shows how the well-known programming constructs such as if-then-else and foreach loops can be expressed in Picat. I can disclose that my favorite programming structure in Picat is the list comprehension as it significantly simplifies descriptions of lists that look more like a mathematical entry of sets than a programming construct. I have found quite useful three subsections highlighting core differences and similarities with Prolog, Haskell, and Python. They can make understanding of Picat easier for programmers knowing or using these languages. The chapter gives at the end some general rules how to write efficient programs in Picat; many of these rules are well known from Prolog. The chapter is concluded by bibliographical notes giving references to sources where the techniques used in Picat come from. There is also a set of programming exercises. This chapter structure is preserved in all subsequent chapters. I particularly like the programming exercises at the end – as every teacher will surely concur, there are never enough nice examples.

Chapters 2–3 cover modeling constraint satisfaction problems in Picat. Do not expect any deep modeling wisdoms there, it is more about expressing classical models in Picat. Again, the executable code is presented to demonstrate how natural these models look in Picat. You can find models for classical problems such as SEND + MORE = MONEY, Sudoku, and N-queens in Chapter 2, while Chapter 3 focuses on more advanced modeling techniques using for example `element`, `regular`, and `global_cardinality` constraints. Again, usage of specific constraints is demonstrated via examples of combinatorial problems such as nurse rostering, Knight's tour, Magic Sequence, and others. There is one Picat property that should be highlighted there and this is using more or less the same encoding for various solvers, namely CP (Constraint Programming), SAT (Boolean Satisfiability), and MIP (Mixed Integer Programming). Some simple comparison of solver efficiency is given in the text. This chapter also includes a short section on debugging constraint models, but I have not found it very useful. Debugging declarative programs is a long-lasting problem in general, and debugging Picat programs with many features beyond simple control flow would not be easy. Perhaps, a chapter dedicated solely to debugging would be a good addition for the next edition of the book.

Chapter 4 is dedicated to problems where dynamic programming is an appropriate method. Picat is well prepared for dealing with them thanks to tabling, a mechanism for memorizing answers to queries and reusing them when necessary. Again, Picat encodings of several classical problems including the shortest path and knapsack problems are presented. I appreciate that the book is not just a collection of codes. Instead, each code is carefully explained so that the reader can understand how and why it is working. Sometimes alternative encodings are presented and compared to demonstrate an advantage of one encoding method over another one. The examples nicely show the beauty of tabling that is hidden in the combination of declarative (so easy to read) code and its efficient execution.

Tabling is heavily exploited in the new `planner` module of Picat that is introduced in Chapter 5. As became a tradition of the book, Picat encodings of several classical planning problems such as Missionaries and Cannibals and Sokoban are used to demonstrate the concept. The new thing here is that the chapter also looks inside the `planner` module and implementation of depth-unbounded search from the module is presented there. It should not come as a surprise that this implementation is in Picat itself so it serves again as an example how to implement some more advanced techniques in Picat.

Chapter 6 continues on planning examples, now solved using the technique of resource-bounded search. Picat implementation of resource-bounded search is presented there together with encodings of other well-known planning problems such as 15-puzzle, Blocksworld, and Logistics planning. Again, it should not be a surprise that these declarative encodings are very compact, easy to understand, yet competitive with existing planners.

The book concludes by a chapter on the traveling salesman problem. One may be curious why the final chapter is dedicated to a single problem. Obviously, a traveling salesman problem is a well-known and well-studied problem and even books were written on this single problem as it is an important problem indeed. Probably the major reason for using traveling salesman problem in the book is to demonstrate and compare various encoding techniques from the book on a single problem. This chapter presents CP, MIP, and SAT encodings of the problem (of course, all in Picat) as well as encoding for the `planner` module so it is a good summary of what the reader has learnt in the previous chapters. These encodings are again nicely compact and easy to understand but they should be treated mainly as educational material as they do not correspond to the state of the art for solving a traveling salesman problem.

In summary, it is a nice booklet that presents capabilities of the Picat programing language using many examples of combinatorial and planning problems. The examples are carefully selected so that the code always fits on one page. I really like that they are all commented and explained, which makes it easier to follow them. All the examples are also available on-line for download. Nevertheless, keep in my mind that the examples are not necessary focused on presenting the state of the art for solving the problem, but still they nicely demonstrate that declarative code can be efficient as well. Thanks to this, the book is a great educational material not only for learning Picat but also to see the beauty and power of declarative programming. It is very appropriate for students and teachers of modern techniques of declarative programming and I believe that researchers and programmers will also benefit from reading it to encounter a new coding tool appropriate especially for AI (artificial intelligence) programming.

Roman Barták
Charles University in Prague
(*e-mail: bartak@ktiml.mff.cuni.cz*)