

Preface

This book takes an operational approach to programming language concepts, studying those concepts in interpreters and compilers for some toy languages, and pointing out their relations to real-world programming languages.

What is Covered

Topics covered include abstract and concrete syntax; functional and imperative programming languages; interpretation, type checking, and compilation; peep-hole optimizations; abstract machines, automatic memory management and garbage collection; the Java Virtual Machine and Microsoft's .NET Common Language Runtime; and real machine code for the x86 architecture.

Some effort is made throughout to put programming language concepts into their historical context, and to show how the concepts surface in languages that the students are assumed to know already; primarily Java or C#.

We do not cover regular expressions and parser construction in much detail. For this purpose, we refer to Torben Mogensen's textbook; see Chap. 3 and its references.

Apart from various updates, this second edition adds a synthesis chapter, contributed by Niels Hallenberg, that presents a compiler from a small functional language called micro-SML to an abstract machine; and a chapter that presents a compiler from a C subset called micro-C to real x86 machine code.

Why Virtual Machines?

The book's emphasis is on virtual stack machines and their intermediate languages, often known as bytecode. Virtual machines are machine-like enough to make the central purpose and concepts of compilation and code generation clear, yet they are much simpler than present-day microprocessors such as Intel i7 and similar.

Full understanding of performance issues in real microprocessors, with deep pipelines, register renaming, out-of-order execution, branch prediction, translation lookaside buffers and so on, requires a very detailed study of their architecture, usually not conveyed by compiler textbooks anyway. Certainly, a mere understanding of the instruction set, such as x86, conveys little information about whether code will be fast or not.

The widely used object-oriented languages Java and C# are rather far removed from the real hardware, and are most conveniently explained in terms of their virtual machines: the Java Virtual Machine and Microsoft's Common Language Infrastructure. Understanding the workings and implementation of these virtual machines sheds light on efficiency issues, design decisions, and inherent limitations in Java and C#. To understand memory organization of classic imperative languages, we also study a small subset of C with arrays, pointer arithmetics, and recursive functions. We present a compiler from micro-C to an abstract machine, and this smoothly leads to a simple compiler for real x86 hardware.

Why F#?

We use the functional language F# as presentation language throughout, to illustrate programming language concepts, by implementing interpreters and compilers for toy languages. The idea behind this is twofold.

First, F# belongs to the ML family of languages and is ideal for implementing interpreters and compilers because it has datatypes and pattern matching and is strongly typed. This leads to a brevity and clarity of examples that cannot be matched by languages without these features.

Secondly, the active use of a functional language is an attempt to add a new dimension to students' world view, to broaden their imagination. The prevalent single-inheritance class-based object-oriented programming languages (namely, Java and C#) are very useful and versatile languages. But they have come to dominate computer science education to a degree where students may become unable to imagine other programming tools, especially to use a completely different paradigm. Knowledge of a functional language will make the student a better designer and programmer, whether in Java, C# or C, and will prepare him or her to adapt to the programming languages of the future.

For instance, the so-called generic types and methods appeared in Java and C# in 2004, but have been part of other languages, most notably ML, since 1978. Similarly, garbage collection has been used in functional languages since Lisp in 1960, but entered mainstream use more than 30 years later, with Java. Finally, functional programming features were added to C# in 2010 and to Java in 2014.

Appendix A gives a brief introduction to those parts of F# used in this book. Students who do not know F# should learn those parts during the first-third of this course, using the appendix or a textbook such as Hansen and Rischel or a reference such as Syme et al.; see Appendix A and its references.

Supporting Material

There are practical exercises at the end of each chapter. Moreover, the book is accompanied by complete implementations in F# of lexer and parser specifications, abstract syntaxes, interpreters, compilers, and runtime systems (abstract machines, in Java and C) for a range of toy languages. This material, and lecture slides in PDF, are available separately from the book's homepage: <http://www.itu.dk/people/sestoft/plc/>.

Acknowledgements

This book originated as lecture notes for courses held at the IT University of Copenhagen, Denmark. I would like to thank Andrzej Wasowski, Ken Friis Larsen, Hannes Mehnert, David Raymond Christiansen and past students, in particular Niels Kokholm, Mikkel Bundgaard, and Ahmad Salim Al-Sibahi, who pointed out mistakes and made suggestions on examples and presentation in earlier drafts. Niels Kokholm wrote an early version of the machine code generating micro-C compiler presented in Chap. 14. Thanks to Luca Boasso, Mikkel Riise Lund, and Paul Jurczak for pointing out misprints and unclarities in the first edition. I owe a lasting debt to Neil D. Jones and Mads Tofte who influenced my own view of programming languages and the presentation of programming language concepts.

Niels Hallenberg deserves a special thanks for contributing all of Chap. 13 to this second edition.

Copenhagen, Denmark

Peter Sestoft



<http://www.springer.com/978-3-319-60788-7>

Programming Language Concepts

Sestoft, P.

2017, XV, 341 p. 87 illus., Softcover

ISBN: 978-3-319-60788-7