
Higher-Order and Symbolic Computation, 13, 7–9, 2000

A Foreword to 'Fundamental Concepts in Programming Languages'

PETER D. MOSSES pdm@brics.dk
BRICS & Department of Computer Science, University of Aarhus, Ny Munkegade, B.540, DK-8000 Aarhus C, Denmark*

Christopher Strachey's paper on *Fundamental Concepts in Programming Languages* is being published here for the first time. Written in the autumn of 1967, it is based on the lectures given by Strachey at an International School in Computer Programming, held in Copenhagen in August 1967. Strachey intended the paper to be published in the proceedings of the School—but the proceedings never materialized, and Strachey's paper has remained an unpublished preprint for more than three decades. It is, however, one of Strachey's most significant and lengthy papers; widely circulated in the original typescript version, it has also been highly influential.

Strachey's paper starts with some philosophical remarks about the need to focus on semantic issues in the design of programming languages, and to "recognise and isolate the central concepts—things analogous to the concepts of continuity and convergence in analysis". Strachey then proceeds to give a clear and incisive exposition of many of his insights into programming language design and formal semantics, covering the following main topics:

- assignment commands, *L*- and *R*-values;
- expression evaluation and environments;
- commands and sequencing;
- modes of parameter-passing and variable binding;
- functions and routines as data items;
- types and (parametric) polymorphism; and
- compound data structures and pointers.

He also indicates how to model some of these concepts using λ-expressions.

The style of semantics proposed by Strachey in this paper was further developed (and put on a firm mathematical foundation) in his collaboration with Dana Scott, which started in 1969; initially referred to as 'mathematical semantics', or simply as 'Scott-Strachey semantics', the framework has since become known as denotational semantics. After Strachey's untimely death in 1975, his insights regarding fundamental concepts in programming languages have been exploited in various textbooks and papers by numerous authors (e.g.,

*Basic Research in Computer Science (`http://www.brics.dk`), Centre of the Danish National Research Foundation.

[1–4, 7, 9], and they have become established as the basis for our understanding of the issues involved.

Thus Strachey's paper has undoubtedly a high degree of historical interest and relevance. His practical experiences with programming throughout the early days of computing (he had an excellent reputation as a master programmer) together with his deep involvement in programming-language design, had made him exceptionally well-qualified to address the fundamental concepts of programming languages.

The paper may also be warmly recommended simply for its clearly formulated explanations of major concepts. I recall studying the paper in 1970 as part of the MSc course at Strachey's Programming Research Group in Oxford; then, it seemed quite challenging reading—but that may be ascribed at least partly to the unfamiliarity of the presented conceptual analysis, which should not be a problem for readers nowadays. However, a few minor caveats may be appropriate here:

One potential difficulty with reading Strachey's paper is that most of the illustrations of programs are given in a lesser-known language called CPL, which had been developed by Strachey and his colleagues during the mid-1960's. CPL was intended as a potential successor to Algol60, but it was never fully implemented—despite the hopes expressed by Strachey in his note at the beginning of the paper. Fortunately, the syntax of CPL is generally quite suggestive of the intended meaning, and in any case the latter is explained in detail in the text where necessary.

Another possible source of confusion is that the semantic functions $\mathcal{L}$ and $\mathcal{R}$ for expressions, introduced in Sect. 3.3.2, do not explicitly take any environment arguments. However, Strachey clearly explains that "we speak of evaluating an expression in an environment (or sometimes relative to an environment) which provides the values of component [identifiers]" (Sect. 3.2.2), and moreover, the representation of functions in Sect. 3.5.2 makes explicit reference to such an environment.

It may also be helpful to bear in mind that in his later work on denotational semantics [8], Strachey was careful to distinguish between the domains of 'denotable' and 'storable' values, rather than lumping them together as '*R*-values'; in this respect, Fig. 1, depicting the conceptual model, may be a bit misleading, in that a constant identifier would normally be mapped directly to a value by the environment, without any involvement of the abstract store, and moreover, a numeral would directly denote an abstract number.

Finally, the reader should not be disconcerted by Strachey's *operational* interpretation of λ-expressions: the paper was written a full two years before Scott provided a model for the λ-calculus and established the domain theory that forms the mathematical foundations of denotational semantics [5, 6]. In fact Strachey was not committed to a particular interpreter (or well-defined order of reduction) for λ-expressions, cf. the discussion of evaluation in Sect. 3.2.4 and the following comment at the end of Sect. 3.3.3:

> [. . .] all concept of sequencing appears to have vanished. It is, in fact, replaced by the partially ordered sequence of function applications which is specified by λ-expressions.

Strachey writes also (at the very end of the paper, when comparing his proposal for a semantic method to previous proposals):

> [. . .] the ultimative machine required (and all methods of describing semantics come to a machine ultimately) is in no way specialised. Its only requirement is that it should be able to evaluate pure λ-expressions.

There is however no indication that the intended machine should be deterministic.

In conclusion, Strachey's paper is still well worth reading carefully, some 30 years after he wrote it. Enjoy it!

References

1. Gordon, M.J.C. *The Denotational Description of Programming Languages*. Springer-Verlag, 1979.
2. Milne, R.E. and Strachey, C. *A Theory of Programming Language Semantics*. Chapman & Hall, 1976.
3. Mosses, P.D. Denotational semantics. In *Handbook of Theoretical Computer Science, Vol. B*. Elsevier Science Publishers, Amsterdam; and MIT Press, 1990, Ch. 11.
4. Schmidt, D.A. *Denotational Semantics: A Methodology for Language Development*. Allyn & Bacon, 1986.
5. Scott, D.S. Outline of a mathematical theory of computation. In *Proc. Fourth Annual Princeton Conference on Information Sciences and Systems*, 1970. A revised and slightly expanded version is Tech. Mono. PRG–2, Programming Research Group, University of Oxford, 1970.
6. Scott, D.S. and Strachey, C. Toward a mathematical semantics for computer languages. *Microwave Research Institute Symposia Series*, Vol. 21: *Proc. Symp. on Computers and Automata*. Polytechnic Institute of Brooklyn, 1971.
7. Stoy, J.E. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. The MIT Press, 1977.
8. Strachey, C. The varieties of programming language. In *Proc. International Computing Symposium*, pp. 222–233. Cini Foundation, Venice, 1972. A revised and slightly expanded version is Tech. Mono. PRG–10, Programming Research Group, University of Oxford, 1973.
9. Tennent, R.D. The denotational semantics of programming languages. *Commun. ACM* **19** (1976) 437–453.