The Varieties of Programming Language Semantics (Summary)

Peter D. Mosses

BRICS & Dept. of Computer Science, Univ. of Aarhus, Denmark
http://www.brics.dk/~pdm/

Abstract. It may seem that there are almost as many varieties of programming language semantics as there are of programming languages. This brief summary surveys the main varieties of semantics, and considers which of them may be the most appropriate.

1 Introduction

In 1972, Christopher Strachey wrote the paper *The Varieties of Programming Language* [27]. (The title of the paper was by analogy with that of a book by William James: *The Varieties of Religious Experience* [10].) Strachey, in collaboration with Dana Scott, had developed the semantic description framework that became known as *denotational semantics*, and in the paper he showed how the relationship between the domains of so-called denotable, expressible, and storable values used in denotational semantics could reveal major characteristics of the described language.

In fact Strachey had an almost religious conviction in the superiority of the denotational approach to semantics over the operational and axiomatic approaches. He found it most natural to represent programming constructs as pure mathematical functions, and Scott-domains provided the foundations for his use of Curry's "paradoxical combinator" for obtaining fixed points of higher-order functions. Sadly, Strachey's work on the development of denotational semantics was cut short by his untimely death in 1975. His insight and ideas have nevertheless had a profound and lasting impact, as witnessed this year by the special issue of *HOSC* [2] published to commemorate the 25th anniversary of his death. The present author is particularly indebted to Strachey for the inspiration and guidance he provided during doctoral studies at the Programming Research Group in Oxford.

However, the proliferation of different frameworks for semantic description of programming languages over the past three decades makes one wonder whether Strachey's satisfaction with denotational semantics was fully justified. Let us (here, very briefly) survey the main varieties of semantics, and then consider which may be the most appropriate for various uses.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 624–628, 2000.

2 Operational Semantics

The main idea of the operational approach to semantics is to model computations step-by-step. Operational frameworks include:

- translation to code for abstract machines such as the SECD-machine [12] or that of VDL [31];
- translation to the λ-calculus [13], which is then evaluated to normal form;
- application of abstract interpreters, which operate on a direct representation of the structure of the program [16];
- application of sets of term rewriting rules, using evaluation contexts [4] to control the order of sub-computations; and
- syntax-directed inference rules for transitions between configurations [1,11, 22,24], where "small-step" transitions model single steps of information processing, whereas "big-step" transitions model entire (terminating) computations, corresponding to the transitive closure of a small-step relation.

3 Denotational Semantics

Here semantic functions, generally defined inductively by semantic equations, map programs to their denotations, which represent computations as functions taking initial states directly to final states without revealing intermediate steps [7,19,26].

Variations on this theme include:

- initial algebra semantics [5], where (abstract) syntax is an initial algebra in a class of algebras, and where semantic equations are replaced by the definition of a target algebra interpreting the syntactic constructs;
- monadic semantics [14,17], where for each type of value, there is the type of computations of such values, and where the values and computations form a monad (equipped with further operations); and
- translation between meta-languages [18], where the semantic functions map programs to meta-language terms that may then be interpreted as functions in monads.

4 Axiomatic Semantics

The main idea here is to exploit assertions about programs:

- Hoare Logic [9] uses inference rules to relate assertions about the values of variables before and after program phrases;
- weakest-precondition semantics [3] is essentially denotational, with denotations being functions from assertions about values of variables after computations to assertions about the values beforehand; and
- laws of programming [25] characterize computation by assertions about program equivalence.

5 Hybrid Approaches

The final varieties of semantics considered here combine features of operational, denotational, and axiomatic semantics:

- action semantics [20,23,30] is similar to monadic denotational semantics, but denotations are actions rather than functions, and the notation used for composing computations is interpreted operationally;
- modular monadic action semantics [29] provides the action notation used in action semantics with a monadic denotational semantics;
- type-theoretic interpretation [8] uses evaluation contexts and reduction rules to define an internal language for composing computations;
- specification frameworks such as OBJ3 [6] and Rewriting Logic [15] can be used to define interpreters and transition relations; and
- mathematical operational semantics [28] allows definition by rules that provide both an operational and a denotational semantics.

6 Conclusion

To give a semantic description of a full high-level programming in *any* framework requires a major effort. Some of the varieties of semantics referenced above reduce the effort required by allowing re-use of parts of descriptions of other languages; these include modular SOS, monadic denotational semantics, action semantics, and modular monadic action semantics. Other important pragmatic aspects of semantic descriptions concern the ease with which they can be used for setting standards for implementations (or merely for communication of the designers' intentions to implementors), program verification, and compiler generation.

It appears that at present, no single semantic framework is ideal with regard to all pragmatic aspects. Sometimes, it is suggested that one should therefore aim provide two or more *complementary* semantic descriptions of each language, in different styles—and prove that they are equivalent—but the extra effort involved would surely be quite prohibitive when describing the semantics of larger programming languages.

In the opinion of this author, it is generally advantageous to translate a (large and complex) programming language into a (smaller and simpler) notation for semantic entities, as in denotational semantics, and in the hybrid action semantics and type-theoretic interpretation frameworks. However, when the distance between the programming language and the semantic notation is large, the translation itself may be uncomfortably complex; when it is small, the semantic notation may be almost as difficult to define or reason about as the programming language. The matter of how best to define the semantic notation itself appears to be less crucial, especially if the same semantic notation can be exploited in the description of many different programming languages.

Acknowledgements The author wishes to thank Professor Ito for encouragement to give an open lecture in conjunction with TCS2000 in Sendai, and for financial support. BRICS (Basic Research in Computer Science) is a centre of the Danish National Research Foundation.

References

1. E. Astesiano. Inductive and operational semantics. In E. J. Neuhold and M. Paul, editors, *Formal Description of Programming Concepts*, IFIP State-of-the-Art Report, pages 51–136. Springer-Verlag, 1991.
2. O. Danvy and C. Talcott. Editorial. *Higher-Order and Symbolic Computation*, 13(1/2):5–6, 2000.
3. E. W. Dijkstra. Guarded commands, non-determinacy, and formal derivations of programs. *Commun. ACM*, 18:453–457, 1975.
4. M. Felleisen and D. P. Friedman. Control operators, the SECD machine, and the λ-calculus. In *Formal Description of Programming Concepts III, Proc. IFIP TC2 Working Conference, Gl. Avernæs, 1986*, pages 193–217. North-Holland, 1987.
5. J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright. Initial algebra semantics and continuous algebras. *J. ACM*, 24:68–95, 1977.
6. J. A. Goguen and T. Winkler. Introducing OBJ3. Technical Report SRI-CSL-88-9, Computer Science Lab., SRI International, 1988.
7. C. A. Gunter and D. S. Scott. Semantic domains. In J. van Leeuwen, A. Meyer, M. Nivat, M. Paterson, and D. Perrin, editors, *Handbook of Theoretical Computer Science*, volume B, chapter 12. Elsevier Science Publishers, Amsterdam; and MIT Press, 1990.
8. R. Harper and C. Stone. A type-theoretic interpretation of Standard ML. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Robin Milner Festschrifft*. MIT Press, 1998. To appear.
9. C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12:576–580, 1969.
10. W. James. *The Varieties of Religious Experience*. MacMillan Publishing Company, 1997. Reprint edition.
11. G. Kahn. Natural semantics. In *STACS'87, Proc. Symp. on Theoretical Aspects of Computer Science*, volume 247 of *LNCS*, pages 22–39. Springer-Verlag, 1987.
12. P. J. Landin. The mechanical evaluation of expressions. *Computer Journal*, 6:308–320, 1964.
13. P. J. Landin. A formal description of Algol60. In *Formal Language Description Languages for Computer Programming, Proc. IFIP TC2 Working Conference, 1964*, pages 266–294. IFIP, North-Holland, 1966.
14. S. Liang and P. Hudak. Modular denotational semantics for compiler construction. In *ESOP'96, Proc. 6th European Symposium on Programming, Linköping*, volume 1058 of *LNCS*, pages 219–234. Springer-Verlag, 1996.
15. N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. In D. Gabbay, editor, *Handbook of Philosophical Logic*, volume 6. Kluwer Academic Publishers, 1998. Also Technical Report SRI-CSL-93-05, SRI International, August 1993.
16. J. McCarthy. Towards a mathematical science of computation. In *Information Processing 62, Proc. IFIP Congress 62*, pages 21–28. North-Holland, 1962.

17. E. Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, Computer Science Dept., University of Edinburgh, 1990.
18. E. Moggi. Metalanguages and applications. In A. M. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, Publications of the Newton Institute. CUP, 1997.
19. P. D. Mosses. Denotational semantics. In *Handbook of Theoretical Computer Science*, volume B, chapter 11. Elsevier Science Publishers, Amsterdam; and MIT Press, 1990.
20. P. D. Mosses. *Action Semantics*. Number 26 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.
21. P. D. Mosses. Foundations of modular SOS. Research Series BRICS-RS-99-54, BRICS, Dept. of Computer Science, Univ. of Aarhus, 1999. Full version of [22].
22. P. D. Mosses. Foundations of Modular SOS (extended abstract). In *MFCS'99, Proc. 24th Intl. Symp. on Mathematical Foundations of Computer Science, Szklarska-Poreba, Poland*, volume 1672 of *LNCS*, pages 70–80. Springer-Verlag, 1999. Full version available [21].
23. P. D. Mosses and D. A. Watt. The use of action semantics. In *Formal Description of Programming Concepts III, Proc. IFIP TC2 Working Conference, Gl. Avernæs, 1986*, pages 135–166. North-Holland, 1987.
24. G. D. Plotkin. A structural approach to operational semantics. Lecture Notes DAIMI FN–19, Dept. of Computer Science, Univ. of Aarhus, 1981.
25. A. W. Roscoe and C. A. R. Hoare. The laws of occam programming. Technical Report PRG-53, Programming Research Group, Oxford University, 1986.
26. D. S. Scott and C. Strachey. Toward a mathematical semantics for computer languages. In *Proc. Symp. on Computers and Automata*, volume 21 of *Microwave Research Institute Symposia Series*. Polytechnic Institute of Brooklyn, 1971.
27. C. Strachey. The varieties of programming language. In *Proc. International Computing Symposium*, pages 222–233. Cini Foundation, Venice, 1972. A revised and slightly expanded version is Tech. Mono. PRG–10, Programming Research Group, University of Oxford, 1973.
28. D. Turi and G. D. Plotkin. Towards a mathematical operational semantics. In *Proc. LICS'97*. IEEE, 1997.
29. K. Wansbrough and J. Hamer. A modular monadic action semantics. In *Conference on Domain-Specific Languages*, pages 157–170. The USENIX Association, 1997.
30. D. A. Watt. *Programming Language Syntax and Semantics*. Prentice-Hall, 1991.
31. P. Wegner. The Vienna definition language. *ACM Comput. Surv.*, 4:5–63, 1972.