On the Need for a Popular Formal Semantics

David A. Schmidt
Kansas State University
Computing and Information Sciences Department
Manhattan, KS 66506 USA

The technology transfer from semantics research to mainstream computing has been hampered by specialization of techniques at the expense of calculational clarity. A "popular semantics" is proposed as a remedy.

Classical music faces a dilemma: as its practitioners specialize into new realms, the field's present-day popularity rests upon the tried-and-true war horses from centuries ago. Among the general populace, there is little excitement for the current state of the art—classical music is no longer popular.

Is formal semantics a similar "classic"? Is it in danger of specializing itself out of the consciousness of the computing public?

Recall that the fundamental advance of Scott-Strachey-style denotational semantics was that it "extended BNF to semantics" [6; 7]. This remains a revolutionary idea: one uses BNF-like techniques to understand the meanings of programs. If its mission had been completed successfully, denotational semantics would permeate the programming languages field: users of language manuals would study the BNF and denotational semantics definitions side by side, compiler writers would employ the semantics to implement compilers, and language designers would map their intuitions into semantics definitions, which are analyzed and fed to rapid prototyping tools.

This mission was never accomplished: typically, denotational (and related operational) semantics definitions are used to dissect fragments of popular languages and extract reasons why a feature is correct or incorrect in some formal sense. Using Reynolds's analogy, semanticists play the role of morticians in the programming language community.

Of course, the situation is not uniformly gloomy: as indicated in the Strategic Directions document [4], semantics research has influenced the design of procedural and type-checking mechanisms in several widely used programming languages, and current research on object-orientation promises guidance for some next-generation object-oriented language. But formal-semantics techniques have not played the same central role in language design, implementation, and understanding as have formal-syntax techniques.

Why isn't there more "technology transfer" from formal semantics? It is not an accident that modern compiler design texts use attribute grammars as a substitute semantics notation: attribute grammars implement the BNF-extended-to-semantics concept with a minimum of technical machinery. In contrast, formal semantics has fed upon increasing complexity of concepts and notation at the expense of calculational clarity. A newcomer to the area is expected to specialize in one or more of domain theory, intuitionistic type theory, category theory, linear logic, process algebra, continuation-passing style, or whatever. These specializations have generated more experts but fewer general users.

General users desire semantics definitions with strong calculational flavor. It is no accident that a beginning programmer debugs a program by drawing a picture of the program's state and calculating state transitions upon it. For formal semantics definitions to attain the popularity they deserve, they must be presented with their underlying calculational behaviour as their primary feature.

A challenge for semantics writers is the following: design a calculational semantics for a significant subset of, say, Java, that can be learned and applied by first-year university students to debug their programs. Perhaps such a semantics will use a graphical state and small-step transitions upon the state for its calculations, but calculation upon program properties—that is, a variation on predicate transformer semantics [3]—is also a possibility. In any case, it is crucial that the formulation be "mathematical" in the sense that denotational semantics was first termed mathematical: there must be an underlying, consistent proof theory. And of course, the semantics should be some form of extension of BNF.

The relationship of this "popular" semantics of Java to a structural-operational or denotational semantics must be formalized. Abstract interpretation [2] is an obvious candidate for this job, because abstract interpretation is a story of property abstraction, where the property might be a sequence of graphical state pictures, or the extensional behaviour of the program, or a partial correctness proof in a programming logic. Indeed, abstract intepretation should be recognized as being more than just a formalization of data-flow analysis: it is the framework upon which complementary semantics definitions, in the sense of Hoare and Lauer [5], are hung [1].

The formulation of popular semantics definitions for programming languages benefits researchers as well as general users: the process of designing calculational semantical mechanisms forces a language researcher to focus more upon the "why," that is, the pragmatics, of a language, as opposed to merely the "what." This helps designer, researcher, and user establish consensus about the language's purpose and its worth. This shift, from formal to popular semantics, is necessary to ensure the semantics field its deserved status as both classic *and* popular.

REFERENCES

[1] P.Cousot The abstract interpretation perspective. This proceedings.

[2] P. Cousot and R. Cousot Abstract interpretation. Proc. 4th ACM Symp. on Principles of Programming Languages, 1977, pp. 238-252.

[3] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[4] C. Hankin, H.R. Nielson, and J. Palsberg. Strategic Directions for Research on Programming Languages. *ACM Computing Surveys* (1997).

[5] C.A.R. Hoare and P.E. Lauer. Consistent and Complementary Formal Theories of the Semantics of Programming Languages. *Acta Informatica* 3 (1974) 135-153.

[6] D.S. Scott. Outline of a mathematical theory of computation. Report PRG-2, Programming Research Group, Univ. of Oxford, 1971.

[7] D.S. Scott and C. Strachey. Toward a mathematical semantics for computer languages. Computers and Automata, Microwave Research Institute Symposia 21, 1971, pp 19-46.