Review of *Action Semantics* by Peter D. Mosses. Cambridge Tracts in Theoretical Computer Science 26, Cambridge University Press, U.K., 372 + xx pages, 1992.

Varol Akman
Bilkent University, Ankara

July 2, 1998

In *ABC of Reading* (Faber and Faber, London, 1979, pp. 39–40), a little book of extraordinary beauty, Ezra Pound makes the following observations:

> When you start searching for 'pure elements' in literature you will find that literature has been created by the following classes of persons:
>
> 1. Inventors. Men who found a new process, or whose extant work give us the first known example of a process.
> 2. The masters. Men who combined a number of such processes, and who used them as well as or better than the inventors.
> 3. The diluters. Men who came after the first two kinds of writer, and couldn't do the job quite as well.
> 4. Good writers without salient qualities. Men who are fortunate enough to be born when the literature of a given country is in good working order, or when some particular branch of writing is 'healthy'. [...]
> 5. Writers of belles-lettres. That is, men who didn't really invent anything, but who specialized in some particular part of writing, who couldn't be considered as 'great men' or as authors who were trying to give a complete presentation of life, or of their epoch.
> 6. The starters of crazes.

Replacing "literature" with "programming language semantics," in this review I'll argue that Mosses should probably be seen as belonging to the third[1] category.

[1] *Caveat:* I am not a specialist in the area of programming language semantics—although I occasionally teach a senior-level course on denotational semantics. Again, in the words of Pound "you might at least beware and avoid accepting opinions from men who haven't produced notable work" (*ibid.*, p. 40).

That is, Mosses does present interesting work which has considerable promise, but I do not sincerely think that action semantics matches the richness and charm of say, denotational semantics. In fact, while reading this book, I have missed at several places the terse yet precise approach of Mosses's excellent survey in the *Handbook of Theoretical Computer Science* [3] (not to mention another superior survey by Gunter and Scott [2]). The theoretically oriented reader will find little in Mosses's book that is of fundamental (or foundational) importance. Mosses seems to accept this anyway, as the following excerpt (p. 5) shows:

> Programmers should find action semantic descriptions almost as easy to read as the usual reference manuals, without much preparation. On the other hand, although the foundations of action semantics are firm enough, the *theory* for reasoning about actions (and hence about programs) is still rather weak, and needs further development. This situation is in marked contrast to that of denotational semantics, where the theory (at least for dealing with deterministic, sequential programs) is strong, but severe pragmatic difficulties hinder its application to realistic programming languages.

As can be understood from the last remark of the above passage, the main goal of action semantics is to give useful semantics descriptions of realistic[2] programming languages. Action semantics can be seen as a mixture of the denotational, algebraic, and operational approaches to formal semantics. It has been under development since 1977, by Mosses, and since 1984 additionally by Watt [7].

What are actions? Basically, actions are dynamic, computational entities. The performance of an action directly represents information processing behavior and in some sense mirrors the stepwise nature of computation. A performance of an action either 'completes' (corresponding to normal termination), 'escapes' (corresponding to exceptional termination), 'fails' (corresponding to abandoning the performance of an action), or 'diverges' (corresponding to nontermination). Action performances reflect possible program behaviors. Mosses initially considers single-agent performances, and assumes that in any particular performance, steps occur in a definite order (rather than concurrently). Thus, the current information available when performing a step is well-defined and conflicting changes to the current information cannot arise. He later turns to multi-agent performances. In this case, a dynamically-changing distributed system of agents proceeds to perform separate actions with asynchronous message traffic among agents (a.k.a. 'true' concurrency).

According to Mosses, denotational semantics is quite weak in terms of pragmatic features (due to its heavy dependence on the λ-notation). In fact, the original motivation for the development of action semantics was Mosses's considerable discontent

[2]For instance, the following report may be seen as an illustration of this goal: *Pascal Action Semantics*, Version 0.6, by P. D. Mosses and D. A. Watt, available from the first author. This report provides, according to the authors, an almost complete formal specification of the dynamic semantics of Standard Pascal.

with practical aspects of denotational semantics. While Mosses accepts that higher-order functions on Scott domains have an elegant mathematical theory [2, 4], he complains that the functions required to represent the semantics of the usual constructs of programming languages are invariably rather complex. He notes that a description of the semantics of a conventional programming language is difficult to read when it is given in terms of semantics equations using purely functional representations. (Besides, it may be a formidable task to grasp the operational implications of such a description. After all, λ-abstraction corresponds to call-by-name, not call-by-value parameter passing.) But the really annoying practical problem with the denotational paradigm, asserts Mosses, concerns the poor modifiability and extensibility of denotational descriptions. He 'proves' this last claim as follows. In all textbooks on denotational semantics (so far!), the simple semantic equations for expression evaluation need to be altogether reformulated when declarations and statements are added.

In general, Mosses explains concepts in an informal manner before giving the associated formal[3] notation. Part I introduces the main ideas and formalism used in action semantics. Part II introduces the details of action notation. It is intended to be read together with Part III which introduces a series of examples of action semantic descriptions. The examples describe assorted programming constructs of—among all things—ADA. This may reflect a deep-rooted bias on my part but I was not at all happy to see that this was the case. Even the 'encouraging' remarks of Mosses ("No previous familiarity with ADA is required, as the necessary concepts are all explained here") did not really help me shake my disappointment. Accordingly, while Mosses does try to justify his insistence to base some of the exercises in his book on ADA[4], I do not find this as a valid excuse.

The book is aimed primarily at researchers (including graduate students) in semantics. In the remaining portion of this review, I would like to compare it with a few books that I am familiar with. Stoy's book [6] is based on C. Strachey's lectures at Oxford and being more than 15 years old, cannot possibly stress the important syntactic constructs of later programming languages. However, it is still one of the best sources available, primarily due to its lucid style. The book of Gordon [1] is more on the engineering side and delves into the foundational problems only in a few places. Still, it is one of the best places to look to have a quick (yet thorough) appreciation of λ-calculus. Schmidt's book [5] includes a rather useful description of domain theory (apparently to be outdated by the upcoming volume of Roscoe and Reed [4]). Its only weakness is (arguably) the coverage of a number of minor or subordinate items which at times is damaging to the overall coherence.

I would like to end this review on a lighter note. As I have stated earlier, a main concern of Mosses in advancing action semantics has been comprehensibility. Accord-

[3] Unfortunately, this gives rise to a fair amount of redundancy in the text. One encounters a more formal version of a current discussion later just to discover that there was probably no need for the less formal version at all. This 'repetitive' style of presentation especially makes itself felt in the prefaces and the summaries that accompany each chapter; I am simply at a loss to apprehend the necessity of such material in a research-level text.

[4] Cf., for instance, p. 19: "ADA was chosen here primarily because its Reference Manual, although large, is quite accessible to the casual reader—thanks to its Glossary and copious cross-referencing."

ing to Mosses, the comprehensibility of action semantic descriptions is high because they use suggestive words. (For example, the action combinator corresponding to sequential performance is written as "`then`".) This way, Mosses states, the action notation becomes a reasonable compromise between say, denotational semantic notations and informal English [sic]. Strange, because long time ago I was taught that the following is one of the most fundamental 'laws' of computer programming:

> Make it possible for programmers to write in English and you will find out that programmers cannot write in English.

References

[1] M. J. C. Gordon, *Programming Language Theory and Its Implementation*, Prentice Hall, 1988.

[2] C. A. Gunter and D. S. Scott, "Semantic domains," *Handbook of Theoretical Computer Science*, Vol. B, pp. 633–674, J. van Leeuwen (ed.), Elsevier, 1990.

[3] P. D. Mosses, "Denotational semantics," *Handbook of Theoretical Computer Science*, Vol. B, pp. 573–631, J. van Leeuwen (ed.), Elsevier, 1990.

[4] A. W. Roscoe and G. M. Reed, *Domains for Denotational Semantics*, Prentice Hall, 1993 (to appear).

[5] D. A. Schmidt, *Denotational Semantics: A Methodology for Language Development*, Allyn & Bacon, 1986.

[6] J. E. Stoy, *The Scott-Strachey Approach to Programming Language Theory*, MIT Press, 1977.

[7] D. A. Watt, *Programming Language Syntax and Semantics*, Prentice Hall, 1991.