Generality Considered Harmful —

A Critique of Descriptive Semantics

E.A. Ashcroft
Department of Computer Science
University of Waterloo

W.W. Wadge
Department of Computer Science
University of Warwick

Research Report CS-79-01

May 1979

Generality Considered Harmful —

A Critique of Descriptive Semantics

E.A. Ashcroft
Department of Computer Science
University of Waterloo

W.W. Wadge
Department of Computer Science
University of Warwick

We would like in this note to offer a criticism of the use of mathematics in most of the current work in the semantics of programming languages. Our complaint is not that the wrong amount of mathematics (either too little or too much) is being used, or that the wrong kind is being used, but rather that mathematics is being used in the wrong way. In semantics, and computer science as a whole, there have always been two points of view concerning the role of mathematics.

One point of view sees mathematics as playing primarily a passive role. According to this point of view the entities considered by computer scientists (machines, languages, systems) are shaped mainly by forces outside mathematicians' control; the job of the mathematician is therefore to develop the necessary tools to study computer science, i.e. to describe, to model, to classify. This we might call the "descriptive" approach.

The other point of view sees mathematics as playing primarily an active role. According to this point of view, machines, languages, and systems are (or should be) our own creations, and we can freely choose to create them so that they conform to mathematically simple principles. The mathematical tools developed are used mainly for design rather than study. Mathematics is used not so much to describe

existing objects as to plan new objects. This we shall call the "prescriptive"† approach.

Our criticism of semantics, be it operational or mathematical, Vienna Definition Language [5] or Scott/Strachey semantics [7], functional or relational, is that it almost always takes the descriptive rather than the prescriptive approach.

In the field of syntax, the difference between the descriptive and prescriptive approaches, and the superiority of the latter, can clearly be seen. The first programming languages (such as FORTRAN or COBOL) were designed in an incremental and ad hoc manner and grammar (as well as everything else) was specified by an informal mixture of English and examples. This was quickly seen to be inadequate, and, in the course of designing Algol 58, Backus, Naur and others (and Chomsky independently) devised what are now known as context-free grammars (or BNF). These grammars are simple and powerful, but, even so, are still not adequate to describe completely the peculiarities of FORTRAN. The Algol 58 committee recognized the limitations of BNF but did not try to extend it to make it more general; instead, they fashioned Algol 60 in such a way as to allow a simple BNF specification. This is an example of the prescriptive approach.

Since that time however, a number of researchers has been unable to resist the temptation to "rectify" the descriptive inadequacies

---

† According to the Oxford English Dictionary, the first meaning of "prescriptive" is "That prescribes or directs, giving definite, precise directions or instructions".

of BNF. The most comprehensive of these generalizations is the system of two-level grammars of Van Wijngaarden, invented to describe Algol 68 [8]. These two-level grammars are completely general in the sense that any recursively enumerable set (for example, the set of valid first-order formulas) can be so specified. Yet in spite of this generality it cannot be said that two-level grammars are really successful. In fact, we maintain that their lack of success is a consequence of their generality. These grammars offer no guidance to the language designer. True, the grammars can describe any language, but in general the description will be so complex and unnatural as to be almost useless. In particular, there is no general way to devise a parsing algorithm for a grammar. This is not surprising, since there may be no such algorithm at all (of any kind) that can recognize the language.

In this field of grammar, the Van Wijngaarden two-level grammars exemplify the passive or descriptive approach.

The area of grammar is one of the most successful in theoretical computer science. We would claim that this is because the prescriptive approach has won out over the descriptive approach. Language designers take great care that the grammars of their languages can be easily specified in BNF or in some even simpler form (e.g. LR(k)†), and consequently can use all the well-understood theoretical and practical properties of such grammars.

---

† The definition of what constitutes an LR(k) grammar is more complicated than the definition of context-free grammars. However, the actual grammars are simpler and easier to understand.

The same tension between the two approaches can be seen in another branch of syntax, namely program verification†. At first, after the initial work of Floyd [3], and Hoare [4], much work went into trying to extend verification techniques to more and more complicated languages, for example, languages with go-to's and various types of parameter passing. This reflects the descriptive or passive viewpoint.

Recently, however, a prescriptive trend has developed, which is to design languages and write programs with verification in mind. Previously, programmers criticized verifiers because their systems were not general enough; now verifiers criticize programmers for using constructs which make their programs hard to verify.

The study of both grammar and inference started off in a passive or descriptive mode, but real progress was achieved only when the move was made to the active or prescriptive mode. It is our contention that the problem with semantics is that it is still in the immature, descriptive mode, and a move to the prescriptive mode is well overdue.

It would not be correct to say that there have been no attempts in semantics to use the prescriptive approach. In fact, one of the most important programming languages, LISP, was designed prescriptively. Other examples are ISWIM, APL, Dijkstra's guarded commands, PROLOG and Lucid. Unfortunately, most of the effort in semantics has gone not into the design of languages but into the design and study of semantic description

---

† We are using the word "syntax" in the way that logicians do, to denote everything pertaining to form rather than meaning. Thus grammar is a branch of syntax, but proof theory is also.

systems, the two main ones being the Vienna Definition Language (operational) and the Scott/Strachey method (mathematical). Both are intended to be as general as possible so that they can "handle" whatever the language designer might produce. Semanticists, from these two schools especially, regard programming languages almost as naturally occurring objects; they see the semanticist as being in essentially the same position as an astronomer gazing through his telescope at some star or galaxy†. We might continue this analogy further and liken the various semantic descriptions to the cosmological theories of the ancient astronomer Ptolomy, which described (reasonably adequately) the apparent motions of the planets in terms of an ingenious system of cycles and epicycles.

The problem with a mainly descriptive approach, whether in astronomy or computer science, is that description itself is not the same as understanding. The Scott/Strachey or Vienna methods may allow us to describe PL/I, but that doesn't mean we necessarily understand what is wrong with it, or what should be done about it. As with two-level grammars in syntax, these semantics systems offer no real guidance to language designers because they are so general and can describe anything the designer might produce. They offer language designers a blank cheque or, to paraphrase Dijkstra [2], they constitute an open invitation to the language designer to make a mess of his language.

---

† Algol is in fact the second brightest star in the constellation Perseus!

All criticisms of the prescriptive approach are essentially the same, namely that it is too restrictive. According to this complaint, the prescriptive approach does not allow programmers or language designers the freedom to use whatever constructs they wish. From this point of view, generality is the road to freedom. We claim, however, that it is understanding, not generality, that leads to freedom. For example, even if a programming language is general enough to allow go-to statements, programmers will not really be free to make their programs work unless they understand the necessity of restricting their use. Furthermore, language designers themselves will not be free to devote their energies to really interesting problems if they have to worry about the consequences of the unfettered interaction between different parts of the language being designed. In other words, they should first understand the necessity of restricting these interactions. In this context at least, we can agree with the maxim of the philosopher Engels that "freedom is the recognition of necessity".

It can be argued that the comparison of descriptive semantics to Ptolomaic cosmological theories loses a lot of its force when it is realized that no cosmological theory can be prescriptive (unless the cosmologist has awesome supernatural powers!). This is true because cosmology is in fact a natural (not an engineering) science; that is to say, it is in fact the study of naturally occurring objects (planets, stars, galaxies) shaped by forces outside our control. Nevertheless, it is not correct to say that cosmology, and natural sciences in general, must be purely descriptive. We would argue that the later theories of Newton and Einstein are different in kind from that of Ptolomy. The genius

of Newton was the realization that the reason an apple fell in his garden was the same as the reason the moon went around the earth. From this he formulated a general principle, the Law of Gravitation. Ptolemy described the motions of the planets but Newton was the first to really understand and explain them. In natural sciences the relevant distinction is not between the descriptive and the prescriptive, but between the descriptive and what we might call the "explicative".

The distinction between description and explanation is crucial in the engineering sciences as well, because the prescriptive method requires explanations. Newton's laws of motion can be used to describe the behaviour of particular naturally occurring objects, such as the solar system, but it also can be used to tell us what must be done to travel to the moon, for example, something Ptolemy's (or Copernicus') system could never do. For that matter, Newton's (and Einstein's) theories do allow, on a small scale, a prescriptive approach to cosmology itself: we can make new moons (satellites). Using the theory of relativity, we can even 'manufacture' suns (in the form of reactors and bombs).

In semantics the works of Kleene, Scott [6] and others constitute the explanatory basis of semantics - they have yielded real understanding of the nature of recursion. These principles are used as the basis of descriptive systems, but can also be used prescriptively, as the basis of (say) LISP or Lucid [1].

We would argue that all scientific endeavours involve description, and usually begin by going through a descriptive phase.

In fact, the better the description the more likely it is to lead to a correct general explanation. The works of Copernicus and Kepler, though descriptive, were essential preliminaries to the work of Newton. The periodic table of Mendeleev provided inspiration to Niels Bohr in his formulation of atomic structure. (High-energy physics is still at the descriptive, classifying stage, however.) A good description is not an end in itself, but a beginning.

We strongly hope that this is the stage that is being reached in computer science.

References

[1] Ashcroft E.A. and Wadge, W.W. "Lucid, a Nonprocedural Language with Iteration". CACM 20, No. 7, (1977) 519-526.

[2] Dijkstra E.W. "Goto Statement Considered Harmful". CACM 11, (1968), 147-148, 538, 541.

[3] Floyd R.W. "Assigning Meaning to Programs". In Proc. Sym. in Applied Math, 19, Mathematical Aspects of Computer Science (J.T. Schwartz, ed.) American Math. Soc. (1967), 19-32.

[4] Hoare C.A.R. "An Axiomatic Basis for Computer Programming". CACM 12 (1969) 576-580, 583.

[5] Neuhold E.J. "The Formal Description of Programming Languages." IBM Syst. J., 2 (1971), 86-112.

[6] Scott D.S. "Data Types as Lattices". SIAM Journal on Computing, 5, No. 3 (1976).

[7] Scott D.S. and Strachey C. "Towards a Mathematical Semantics for Computer Languages". Proceedings of the Symposium on Computers and Automata. Polytechnic Institute of Brooklyn, New York, (1971) 19-46.

[8] van Wijngaarden A., Sintzoff M., Lindsey C.H., Meertens L.G.L.T., and Fisker R.G. "Revised Report on the Algorithmic Language Algol 68". Acta Informatica 5 (1975) 1-236.