Chapter 7.6
Programming Languages as Mathematical Theories

Raymond Turner
University of Essex, UK

ABSTRACT

That computer science is somehow a mathematical activity was a view held by many of the pioneers of the subject, especially those who were concerned with its foundations. At face value it might mean that the actual activity of programming is a mathematical one. Indeed, at least in some form, this has been held. But here we explore a different gloss on it. We explore the claim that programming languages are (semantically) mathematical theories. This will force us to discuss the normative nature of semantics, the nature of mathematical theories, the role of theoretical computer science and the relationship between semantic theory and language design.

INTRODUCTION

The design and semantic definition of programming languages has occupied computer scientists for almost half a century. Design questions centre upon the style or paradigm of the language, (e.g. functional, logic, imperative or object oriented). More detailed issues concern the nature and content of its type system, its model of storage and its underlying control mechanisms. Semantic questions relate to the form and nature of programming language semantics (Tennent, 1981; Stoy, 1977; Milne, 1976; Fernandez, 2004). For instance, how is the semantic content of a language determined and how is it expressed?

Presumably, one cannot entirely divorce the design of a language from its semantic content; one is not just designing a language in order to construct meaningless strings of symbols. A programming language is a vehicle for the expression of ideas and for the articulation of solutions to problems; and surely issues of meaning are central to this. But should semantic considerations enter the picture very early on in the process of design, or should they come as an afterthought; i.e. should

DOI: 10.4018/978-1-61350-456-7.ch7.6

we first design the language and then proceed to supply it with a semantic definition?

An influential perspective on this issue is to be found in one the most important early papers on the semantics of programming languages (Strachey C., 2000).

I am not only temperamentally a Platonist and prone to talking about abstracts if I think they throw light on a discussion, but I also regard syntactical problems as essentially irrelevant to programming languages at their present state of development. In a rough and ready sort of way, it seems to be fair to think of the semantics as being what we want to say and the syntax as how to say it. In these terms the urgent task in programming languages is to explore the field of semantic possibilities….When we have discovered the main outlines and the principal peaks we can go about describing a suitable neat and satisfactory notation for them. But first we must try to get a better understanding of the processes of computing and their description in programming languages. In computing we have what I believe to be a new field of mathematics which is at least as important as that opened up by the discovery (or should it be invention) of the calculus.

Apparently, *the field of semantic possibilities* must be laid out prior to the design of any actual language i.e., its syntax. More explicitly, the things that we may refer to and manipulate, and the processes we may call upon to control them, needs to be settled before any actual syntax is defined. We shall call this the *Semantics First* (SF) principle. According to it, one does not design a language and then proceed to its semantic definition as a post-hoc endeavour; semantics must come first.

This leads to the second part of Strachey's advice. In the last sentence of the quote he takes computing to be a new branch of mathematics. At face value this might be taken to mean that the activity of programming is somehow a mathematical one. This has certainly been suggested elsewhere (Hoare, 1969) and criticized by several authors e.g. (Colburn T. R., 2000; Fetzer, 1988; Colburn T., 2007). But, whatever its merits, this does not seem to be what Strachey is concerned with. The early part of the quote suggests that he is referring to programming languages and their underlying structures. And his remark seems best interpreted to mean that (semantically) programming languages are, in some way, mathematical structures. Indeed, this is in line with other publications (Strachey C., 1965) where the underlying ontology of a language is taken to consist of mathematical objects. This particular perspective found its more exact formulation in denotational semantics (Stoy, 1977; Milne, 1976), where the theory of complete lattices supplied the background mathematical framework. This has since been expanded to other frameworks including category theory (Oles, 1982; Crole, 1993).

However, we shall interpret this more broadly i.e., in a way that is neutral with respect to the host theory of mathematical structures (e.g. set theory, category theory, or something else). We shall take it to mean that programming languages are, via their provided semantics, mathematical theories in their own right. We shall refer to this principle as the Mathematical Thesis (MT).

Exactly what MT and SF amount to, whether they are true, how they are connected, and what follows from them, will form the main focus of this paper. But before we embark on any consideration of these, we need to clarify what we understand by the terms *mathematical theory* and *semantics*.

MATHEMATICAL THEORIES

The nature of mathematical theories is one of the central concerns of the philosophy of mathematics (Shapiro, 2004), and it is not one that we can sensibly address here. But we do need to say something; otherwise our claim is left hanging in the air. Roughly, we shall be concerned with theories that are axiomatic in the logical sense.