GORDON76

Gordon, Michael J.C., "The Denotational Description of Programming Languages: An Introduction", Springer-Verlag, 1976 Three questions organise this review: What is denotational semantics? When is it useful? Does this book succeed at teaching denotational semantics? There are several ways for describing what computer programs mean. Many of these approaches can be grouped under the umbrella of denotational semantics. Giving a denotational semantics for a programming language consists of defining a function that maps programs to denotations. These denotations serve as meanings for the programs. Hopefully the realm of denotations is better understood, or in some way more convenient, than the programs themselves. This book teaches what is now often referred to as Scott-Strachey style denotational semantics. The language of mathematics is used to define the meaning function and its target space. The denotations are mathematical functions. For example, functions from initial states to final states are commonly used to model the meanings of programs written in simple imperative languages. One might want a formal semantics to help prove programs correct. To accomplish this within a denotational semantics framework, denotations that are easy to reason about are needed. The denotations used in axiomatic approaches are designed for this kind of reasoning, whereas Scott-Strachey style denotations are not. Scott-Strachey semantics can help, however, by saying exactly what a particular construct does--and doesn't. Most modern language manuals contain a formal syntax description. It would be nice if the language designer could communicate semantics to the programmer in a similar way. Scott-Strachey semantic equations are ill-suited to this purpose because most programmers can't read them. Since the language of mathematics--including lambda notation and higher order functions--is used, few will want to learn. Scott-Strachey style semantics can be used as a design tool. Giving a formal description of a language often reveals ambiguities in the informal description. The designer can then resolve these choices, rather than the implementor. Sometimes, formal descriptions of two distinct language constructs turn out to be nearly the same. In this situation, the designer may be able to simplify the language. Designers can communicate between themselves using formal semantics. And the final language standard can be expressed exactly. The functions and target spaces used to give a denotational account of a language are usually defined recursively. What this means, in effect, is that one provides equations in which the object being defined appears on both sides of the equal sign. The hope is that there is exactly one solution. But there might be an infinity of solutions, or none at all. The mathematical theory developed by Dana Scott provides guidelines that, if followed, guarantee that these equations do indeed define something. This book does not discuss the underlying mathematical theory. Nor does it do a good job of presenting the guidelines that the theory provides. This book teaches enough to write defining equations that make intuitive sense and are probably within the guidelines, but not enough to be sure that they are. On the positive side, an ability to read Scott-Strachey style denotational descriptions is a good thing to have, and this book is an excellent place to get it.