Inductive types in constructive languages

Bruin, Peter Johan de

Summary

This dissertation deals with constructive languages: languages for the formal expression of mathematical constructions. The concept of *construction* does not only encompass computations, as expressed in programming languages, but also propositions and proofs, as expressed in a mathematical logic, and in particular the construction of structured mathematical objects like sequences and trees. *Types* may be conceived of as classes of such objects, and *inductive types* are types whose objects are generated by production rules.

The purpose of this dissertation is twofold. First, I am searching for languages in which the mathematician can express his inspirations well structured, correct, and yet as freely as possible. Secondly, I want to collect the diverging approaches to inductive types within one framework, so that it becomes apparent how the diverse construction and deduction rules arise from a single basic idea and also how these rules may be generalized, if desired. As basic idea I use the concept of *initial algebra* from category theory.

My research into mathematical languages has not led to a complete proposal. The present treatise is confined to general reflections and the partly formal, partly informal description of a language, *ADAM* (chapter 2). This language serves subsequently as a medium for the study of inductive types, which constitutes the main body of the dissertation.

The set-up of *ADAM* is as follows. To guarantee the validity of arguments expressed in the language, it needs a sound foundation. I develop a constructive type theory (called ATT) for this, a combination of the "Intuitionistic Theory of Types" of P. Martin-Löf and the "Calculus of Constructions" of Th. Coquand. In order to comprise all mathematical principles of deduction, I add the iota or description operator of Frege. It is not necessary to include inductive types as a basic principle; natural numbers suffice to construct these.

On this foundation I build the language *ADAM* by looking at how constructions and proofs that I encountered or drafted could be formulated as naturally as possible while adhering to the rules of type theory. The formal definition of *ADAM*, as far as it is available, and its semantics in the underlying type theory are simultaneously given by means of a two-level grammar. This makes it in principle possible to extend the language, while preserving validity, with notations or sublanguages for special applications, like program correctness. The proposed notations should therefore not be regarded as immutable. Perhaps the only typical language element is the notation for (and the consistent use of) *families* of objects.

As a preparation for inductive types, I start with rendering the classical approaches

to inductive definitions (chapter 3), followed by the introduction of the machinery which we require—elementary category theory and algebra (chapter 4).

The central part of the treatise consists of the description and justification of inductive types as initial algebras. First, I consider at an abstract level the various ways of specifying inductive types, and how these specifications designate (via a polynomial functor) an algebra signature, possibly with equations (chapter 5). Next, I analyse and generalize the ways of defining recursive functions on an inductive type (chapter 6). Then I investigate to what extent these construction principles can be dualized to co-inductive types, which are final co-algebras (chapter 7). Finally, I construct, using either elementary set theory or type theory, initial algebras and final co-algebras for an arbitrary polynomial functor, which actually proves the relative consistency of all discussed construction principles in relation to *ADAM*'s Type Theory ATT (chapter 8).

The dissertation is concluded with the treatment of some issues related to inductive types. In chapter 9, I consider recursive datatypes with partial objects, as they occur in programming language in which one should reckon with possibly non-terminating program parts. I summarize the required domain theory, and construct such domains in *ADAM* using final co-algebras. In chapter 10, I briefly discuss inductive types in impredicative languages, types as collections of type-free values, and the principle of bar induction, and I suggest the possibility of inductive definition of new type universes within a type theory. Chapter 11 gives a number of further reflections on mathematical language and proof notation, and summarizes the approaches to inductive types.

The appendices contain the basic principles of set theory and of ATT, the required addition of either the iota operation or proof elimination to type theory, and a study of uniformity properties (*naturality*) of polymorphic objects, which I need on certain occasions.