LOGICAL SPECIFICATION AND IMPLEMENTATION

T S E Maibaum*, M R Sadler*, P A S Veloso**

* Dept. of Computing
  Imperial College of Science and Technology
  180 Queen's Gate, London SW7 2BZ

** Departmento de Informatica
  Pontificia Universidade Catolica
  Rua Marques de Sao Vicente, 225
  22453 Rio de Janeiro, RJBrazil

Abstract

It has become customary to focus attention on the semantic aspects of specification and implementation, a model theoretic or algebraic viewpoint. We feel, however, that certain concepts are best dealt with at the syntactic level, rather than via a detour through semantics, and that implementation is one of these concepts. We regard logic as the most appropriate medium for talking about specification (whether of abstract data types, programs, databases, specifications - as an interpretation between theories say, rather than something to do with the embedding of models or mapping of algebras. In this paper, we give a syntactic account of implementation and prove the basic results - composability of implementations and how to deal with structured (hierarchical) specifications modularly - for abstract data types.

Introduction

As we see it, the two key concepts in an approach to, or theory of, specification are the notions of specification (our objects if you like) and implementation (the morphisms between objects). At this stage we feel that is not as appropriate to investigate the category-theoretic properties of these notions (giving us the category(s) of specifications and implementations), as to continue to explore
particular ways of looking at these notions based on various mathematical formalisms (algebra, set-theory, logic for example) and how these formalisms support more complex ideas like parameterisation and other mechanisms for structuring specifications.

Our claim is that logic, or the logical approach, with an emphasis on syntactic ideas is a particularly fruitful formalism. In this paper we show how the logical approach supports specification of abstract data types and implementations of abstract data types within other abstract data types.

Given an area of computing science an important first step for a theory of specification with respect to that area is an identification of what is (are) the basic unit(s) of specification, see [LZ]. That is, the packages that are used as the atomic building blocks for building more complex, structured specifications. Here we consider abstract data types as our units, or atoms with programming as the obvious area in mind. Similarly any formalism offers, or studies, various structures: in the case of logic, logics or theories say. Now part of what we would like for an approach to specification is a natural match, in some sense, between the formal structures and the units of specification. And this natural match should also extend to a match between on the one hand the kinds of mappings between our formal structures and on the other the ways we naturally put our specifications together to form structured specifications and implement specifications in each other.

The formal structures to study in a logical approach are the theories given by languages, $L$, over some fixed consequence relation $\vdash$-determining the logic. The decision to follow a logical approach appears in no way to commit one to any particular $\vdash$, other than a requirement that certain meta-theorems are provable about $\vdash$, the Craig Interpolation Lemma for example. (See 'Theorem' in the section: Using Implementations).

We choose to use an infinitary logic and this is where criticism is often focused. However we feel that such criticism misses the point. The major focus of attention should be as to whether we have this natural match between units of specification and formal structures, here between abstract data types and theories. We claim this match is obviously natural.