ABSTRACT

Modularity of programs is studied from a semantic point of view. A simple model of modular systems and modularization mechanisms is presented, together with correctness criteria for modular systems. A concept of locality of modular systems is defined; it is a property which "good" modular decompositions should have. The locality of certain kinds of modularization mechanisms is studied, and the results are applied to parameterless procedures.

1. INTRODUCTION

Modularity is one of the most important concepts in computer science. It is the key to mastering the complexity of large programs and is therefore central in the theory of programming. There is quite an extensive body of research on specific modularization mechanisms, such as procedures, processes and abstract data types. Modularity in itself, in abstraction, has not been equally well investigated. The most notable studies in this direction are those by Parnas (e.g. /4,5,6/), in which he discusses general principles for program modularization, covering such aspects as the appropriate choice of program modules and abstraction levels, gives different explications of the notion of hierarchicality and discusses the loss of transparency when using abstraction. The incorporation of abstract data types in recent programming languages such as Ada and Modula has somewhat renewed the interest in basic modularization principles (see e.g. /3/).

It is our belief that the concept of modularity, as a general principle for organizing programs, can be studied in abstraction, and that one can derive nontrivial properties which well-modularized systems should have. The present paper reports on one way of approaching this problem. We study modular systems from a semantic point of view, by giving a semantic model of modular systems and defining a notion of correctness for such systems. Our basic concern is to characterize modularization mechanisms in which the correctness of a modular system can be established by checking for each module separately that it satisfies its specification, provided all modules it uses satisfy their specification. Such modularization mechanisms will be called local. We will give a precise definition of this property within the framework of our semantic model and then discuss the conditions under which modularization mechanisms are local.
A semantic approach, as opposed to a syntactic approach within some fixed formal system, is chosen for a number of reasons. It allows us to study properties of modular systems without being too much distracted by questions of expressibility within a specific formal system. The theory can also be developed with a minimum of assumptions. A small disadvantage of this approach is that we in some situations are forced to give semantic definitions of concepts (like terms, declarations and hierarchicality) which are rather syntactic in flavour.

2. DECLARATION MECHANISMS

The main problem in the approach we have chosen is to find a simple semantic model for modular systems. We want to describe semantically modular system like the following one:

\[
\begin{align*}
DCL: & \ x: A(y,z) \\
 & \ y: B(w) \\
 & \ z: C(z,w) \\
 & \ w: D
\end{align*}
\]

dependency graph

module declarations

Figure 1. An example modular system

The left hand side describes the way in which the modules \( x, y, z \) and \( w \) depend on each other. We see that module \( x \) uses both modules \( y \) and \( z \), that both \( y \) and \( z \) use module \( w \) (sharing), that module \( z \) also uses itself (recursion) and that module \( w \) does not use any other modules. The right hand side shows schematically the way in which such a modular system usually would be declared, by associating an implementation \( A(y,z) \) with \( x \), \( B(w) \) with \( y \) and so on. The dependency of e.g. \( x \) on \( y \) and \( z \) is indicated by the free occurrences of the names \( y \) and \( z \) in the implementation \( A(y,z) \) of \( x \).

The declaration DCL above will define a meaning for each module \( x, y, z \) and \( w \). What exactly this meaning is depends on the application at hand. If we are declaring procedures, then \( A(y,z) \), \( B(w) \), \( C(z,w) \) and \( D \) would be procedure bodies, and the meaning of \( x, y, z \) and \( w \) would be some kind of state transformations. If again we were declaring (abstract) data types, then the meanings could be algebras. It is our intention here to abstract away from the specific choice of meaning for modules and study only those properties which are common to all modular systems, independently of what the meanings of the modules are assumed to be. We therefore simply postulate that a set \( \text{Obj} \) of possible meanings of modules is given. The elements of \( \text{Obj} \) are referred to as objects.