MULTIPLE REPRESENTATIONS OF ABSTRACT DATA TYPES


Computer Science Department
Oklahoma State University
Stillwater, Oklahoma 74078

1. INTRODUCTION

The concept of abstract data types allows the use of complex objects without concern about their implementation. This is accomplished by the separation of specification from implementation details. Modern programming languages incorporate this concept into their design to attempt to provide adequate facilities for the separation of specification from implementation. For example, Ada allows specification (eg. package specification) and implementation (eg. package body) to be separate library units. The implementation bundles together a storage representation and the operations of the abstract data type (ADT). The specification can be viewed as an ADT which defines an object and a set of operations on it. There is usually a one-to-one correspondence between the specification and implementation. The user of an ADT has no influence on the representation chosen for the ADT.

In certain applications it is useful to choose the representation and algorithms. This paper presents the idea of providing such a facility in the context of abstract data types. The separation of specification from implementation also must be preserved.

2. REALIZATION AS A TYPE

An ADT can be implemented using several data structures. We call an implementation of an ADT "t" a realization of t. A realization essentially is a data structure. A data structure consists of three components: a set of function definitions, a storage structure, and a set of algorithms, one for each function. For our discussion, an algorithm is identified with its function name. Therefore, we can view a data structure as a pair (S,F) where S represents a storage structure and F represents a set of functions. An equivalence relation can be imposed on a set of data structures. Let D be a set of data structures. Two data structures d1 = (S1,F1) and d2 = (S2,F2) in D are equivalent if d1 and d2 are realizations for the same set of ADTs. If D consists of exactly one equivalence class and if the equivalence is imposed by an ADT t, then D is called a realization class for t. In other words, every member of D will be an implementation of the ADT t. The set D contains more information (eg. storage structure) than the ADT t itself. If certain information hidden in D is available to the user of t, then that user will be able to choose the appropriate realization based on the application from D. This can be achieved by viewing D as a type. We call this type realization. Using D and t, we can define a new type which allows one to select a realization for an ADT. This new type is called rabstype where, as an ADT, it is called abstype. The formal definitions are given below:

1) TYPE is realization
   VALUES are in D
   OPERATION is selection: D --> d, where d is a member of D.
2) TYPE is rabstype
   VALUES are in t x D
   OPERATION is binding: (t,d) --> td

   where td refers to an instance of t with realization d. The concepts described above are incorporated into a "mini-language representation" which is a small language designed to illustrate these concepts.

3. CONCLUSION

The idea of abstract data types with multiple implementations is explored in this paper. At the same time specification and implementation are separated. It is possible to choose an implementation dynamically. Our current work in progress addresses this problem in object oriented programming languages.