PROGRAM PORTABILITY, DATA MANIPULATION LANGUAGES, AND DATA DESCRIPTION LANGUAGES

J.E. Rodriguez

This note considers the relationship between program portability and the facilities available in data manipulation and data description languages.

We say that a program in a procedural language is portable if its procedural statements remain invariant under changes of environment. Examples of change of environment are a different set of hardware, a different operating system, and different storage structure for the data on which the program operates.

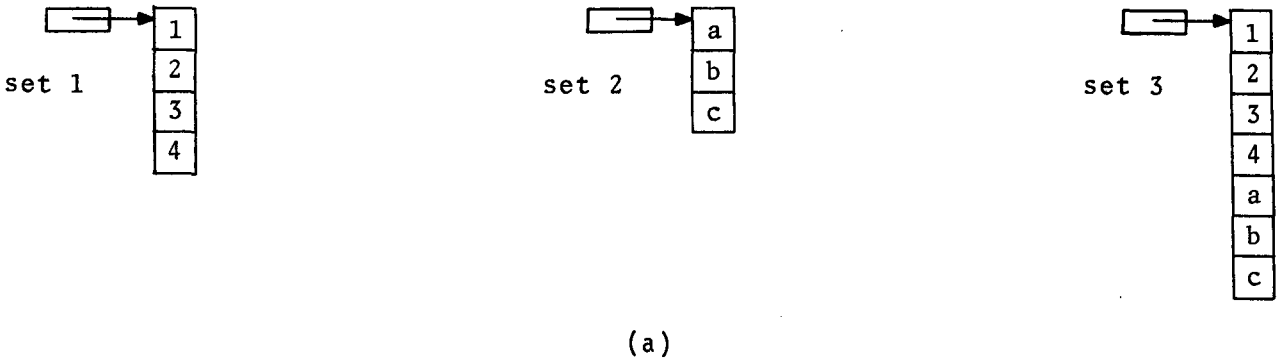
The degree of portability of a program is a measure of the type and magnitude of environmental changes under which the program remains operative. Ideally, a program should be portable for all environment changes. However, present technological limitations preclude this ideal situation for all but the simplest programs.

We propose that there exists a relationship between the time at which certain characteristics of a program are bound and the degree of portability achievable for that program. Specifically, a higher degree of portability is achieved by binding these characteristics as close as possible to the run time of the program. With the current hardware/software technology, greatest portability is achieved when the binding takes place at compile-load time. Thus, we are led to consider characteristics of data description and data manipulation languages that permit the maximum postponement of binding of design and implementation details.

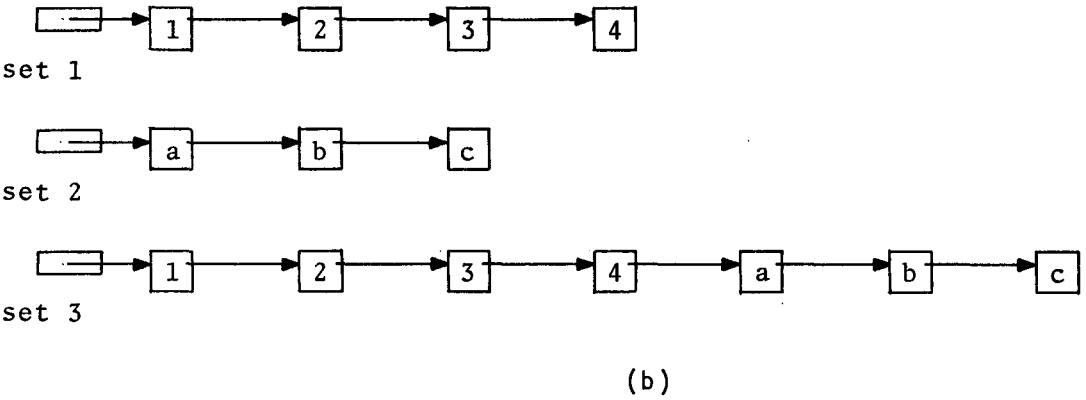
The following view of the relationship between the three components of a program (namely, algorithm, data, and data structure) yields an operationally effective view of portability. An algorithm defines a process over an abstract set of data. By virtue of that process, the algorithm induces certain abstract data structure on that set of data. The specialization of an algorithm to an actual data base is accomplished by a set of declarations (and ancillary mechanisms) which map the abstract data and data structure required by the algorithm into an isomorphic image realized in the data and structure of the data base. Such an isomorphism preserves the integrity of a given algorithm with the data upon which that algorithm operates. If several different sets of declarations yield such isomorphisms with different data bases, the algorithm, and hence the process, is highly portable.

A simple example will serve to illustrate the concept. Consider the problem of taking two partially ordered sets of data objects and producing a third set consisting of all the members of the first set followed by all the members of the second set. An algorithm to solve this problem is to scan the first set copying each data object into the third set, then do the same for the second set. In order to perform the scanning process, certain structure is required, namely, a *next* relation between two consecutive members of a set.

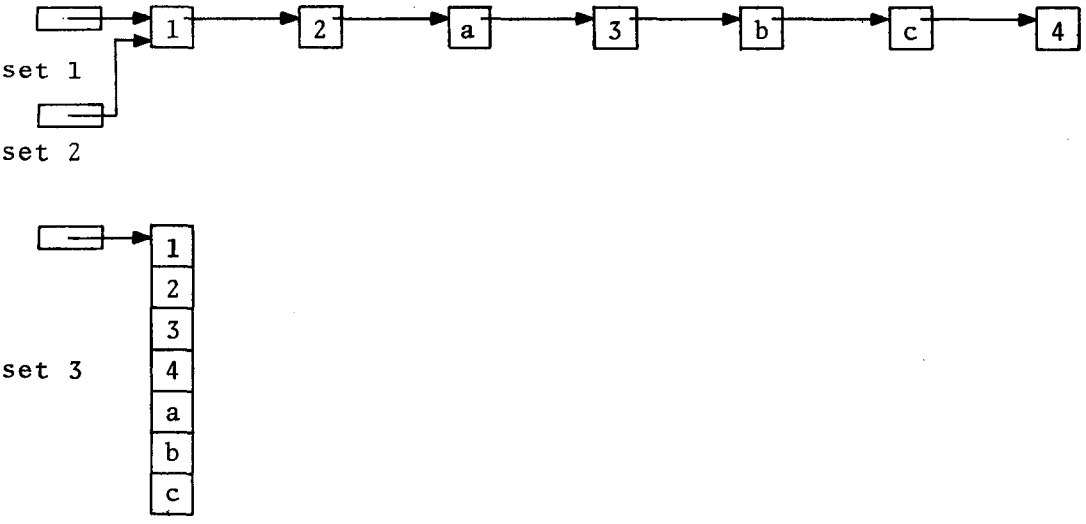
Figure 1 shows examples of three environments under which the previous algorithm could be called to operate. In this case, the environments differ in



(a)



(b)



(c)

FIG. 1. Three Implementations of the Same Data Structure: (a) Incrementing a Pointer; (b) Extracting a Direct Pointer; (c) Discrimination Procedure for Subset Extraction.

the storage structure of the various sets involved. The portability criterion requires the expression of the algorithm to remain unchanged for each of these environments.

By writing the algorithm in terms of three *next* relations, one for each set involved, it is possible to use the same program in each of the three environments.

In the case of Fig. 1a, all next relations are implemented essentially by incrementing a pointer. In Fig. 1b, the implementation calls for extracting a direct pointer. In the case of Fig. 1c, the next relations for set 1 and set 2 are somewhat more elaborate since it is required to extract a subset through some discrimination procedure. The example also illustrates how the degree of portability suffers when the implementation of these relations is bound too early. For example, if it is assumed and imbedded in the program that all the storage occupied by a set is contiguous, as in Fig. 1a, then the program will have to be modified for the other environments shown.

As the example illustrates, there are two aspects of the structural relations: the abstract ideal and its implementation. The ideal or conceptual aspect remains invariant for a given algorithm while the implementation is a function of the environment. Furthermore, the existence of these two aspects is not limited to structural relations, but it also applies to nonstructural properties of the data objects themselves.

The functions of data manipulation and data definition languages can be defined on the basis of these two aspects of structural relations and data object properties.

The function of a data manipulation language should be to provide an expressive vehicle for the representation of algorithms in terms of representation-independent relation and data object properties.

The function of a data description language should be to bind the representation and access characteristics of relations and data object properties to some particular implementation. The binding process can be dissected in two steps: First, a mechanization in terms of a *referent* type is chosen, e.g., a relation may be mechanized as a pointer, macro, procedure, etc. Second, the chosen mechanization is implemented through appropriate encoding. The key to achieving complete freedom in the choice of referent type is the existence of a *uniform referent* notation [*1*] in the data manipulation language. This ability to choose any referent type significantly contributes to the degree of portability of the program for the algorithm remains unchanged when various implementations are specified using the data description language.

REFERENCES

1. Ross, D.T., "Uniform Referents: An Essential Property for a Software Engineering Lanugage." Paper presented at the 3rd International Symposium on Computer and Information Sciences (COINS-69), Miami Beach, Florida, 18–20 December 1969.