Toward an understanding of (actual) data structures

William L. Honig* and C. Robert Carlson†

The myriad data structures provided by contemporary programming languages and data base management systems can no longer be differentiated by affixing a hopefully unique name to each one. Instead it is advisable to concentrate on the basic characteristics and qualities which make one data structure similar to or different from another. This paper proffers a model for common 'aggregate' data structures (including arrays, matrices, *n*-tuples, sequences, hierarchies, and sets) based upon this idea.

The data structure model delineates an *n*-dimensional space of possible data structures; each 'axis' of the space records one major characteristic for the class of aggregate data structures. Each axis is expressed as a question and a list of possible answers. The question describes the characteristic and the set of answers list the variations among the popular aggregates. A particular data structuring technique may be concretely defined by determining its value along each axis; thus, the need for individual descriptive names is lessened.

Since this model is motivated by an analysis of the variations observed among a collection of actual data structures, it is called an 'analysis' model. This kind of model is useful for selecting, comparing, and teaching data structures, as well as for simply understanding exactly what a data structure is.

(Received October 1976)

Downloaded from https://academic.oup.com/comjnl/article/21/2/98/477635 by guest on 19 April 2024

1. Introduction

Consider arrays, matrices, sets, *n*-tuples, hierarchies, and sequences. All these data structures have something in common: they are techniques for grouping together a collection of more primitive components. Nevertheless, there are wide variations among these data structures as they are known by today's programmers and data base designers. These variations result from the restrictions and generalisations imposed on the data structures by contemporary programming languages and data base management systems. This proliferation of data organisation techniques and names for them might be called a 'terminology overload'; it is not unusual for different programming systems or text books to use the same name for different data structures and to use unlike names for the same basic data organisation. Thus it becomes impossible to speak with precision about a data structure simply by giving it a name.

What is needed is an understanding of the basic characteristics of data structures—a means to unveil the differences and similarities among data structures. The data structure model presented here describes data structures in a way which is not dependent upon names for various data organisations. Instead, a model which makes clear the 'parameters' or 'degrees of freedom' for a class of similar data structures is defined. With such a model, it is possible to represent a particular data structure without assigning a (new or old) name to it, but by specifying a set of proper parameter values.

This paper presents such a model for a large class of data structures provided by today's programming languages and data base management systems. The model was developed from an analysis of the actual characteristics of these data structures and is thus termed an 'analysis' model. The model takes the form of a set of questions or 'axes'; each axis records one characteristic of the data structure.

Section 2 presents further motivation for this kind of model and introduces some examples to be used throughout the paper. Section 3 introduces the five axes of the model and demonstrates their use for modelling a wide variety of data structures. Section 4 comments on this particular model of data structures and the analysis-style approach to modelling. Further motivation of the need for a new type of data structure model has been presented elsewhere (Honig, 1974).

2. An example using arrays

The ubiquitous array provides a useful example of the current state of affairs. **Figs. 1, 2** and **3** show three uses of data structures which might be called arrays; each figure shows both the declarations of the array structure in a programming language and a pictorial representation of two instances of the data structures. All three arrays represent information from a personnel data base containing data about people, their salaries, and their skills.

Fig. 1 shows a traditional array as it might be expressed in FORTRAN (American Standards Association, 1966). The SALHIS ('salary history') array records an employee's five previous salaries. No one would quarrel with the claim that Fig. 1 shows an array.

Some people would, however, complain when the data structure of Fig. 2 is called an array. The SKILLS data structure records an employee's experience as a collection of skills and years of experience in each skill. Thus each element or constituent of SKILLS is itself composed of two sub-elements (named SKILCODE and SKLYRS in Fig. 2). The VERS2 declarations (Earley, 1973) shown define SKILLS as a 'sequence' of SKILL data elements; the VERS2 sequence allows, as one alternative, references to its constituents via ordinal numbers and is thus VERS2's version of the traditional array. However, VERS2 allows any data type whatsoever to occur as elements of a sequence and it permits instances of a sequence to have unequal numbers of elements. For these reasons, some people would feel the VERS2 sequence is not a true array.

Fig. 3 depicts a data structure which might also be part of a personnel data base; the data structure PEOPLE contains assorted information on each employee (including SALHIS and SKILLS which have just been considered as independent

*Advanced Programming Technology Department, Bell Laboratories, Naperville, Illinois 60540, USA. Current address: ITT Telecommunications Technology Center, 1351 Washington Blvd., Stanford, Connecticut 06902, USA.
†Northwestern University, Evanston, Illinois, USA.

FORTRAN

```
DIMENSION SALHIS (5)
REAL SALHIS
```

1	1333.33	1	1025.00
2	1095.00	2	912.00
3	925.00	3	875.12
4	675.33	4	0.0
5	600.00	5	0.0

Fig. 1

VERS2

```
SKILL :: ≺SKILCODE → INT, SKLYRS → INT≻
SKILLS :: SEQ(SKILL)
```

```
1 <1000, 5>        1 <1000,1>
2 <1002,1>         2 <1001,1>
3 <1557,3>         3 <1021,5>
4 <1907,1>
5 <1998,2>
```

Fig. 2

data structures in Figs. 1 and 2). Almost anyone will agree that there is more to this data structure than is normally implied by the term 'array'. Not only does the PEOPLE array consist of other than atomic elements, but its elements are of different kinds and some of its elements are themselves arrays. PASCAL (Wirth, 1971) allows this nesting of 'arrays' as shown in Fig. 3. The array PEOPLE consists of 999 instances (one per ID) of data type PERSON, which itself contains arrays.

Similar variations may be observed among the collection of data structures termed 'aggregates' (as discussed by Sammet, 1969, pp. 74-75). This collection includes numerous sorts of data structures which are called arrays, matrices, sets, n-tuples, hierarchies, and sequences. Such a wealth of data structures is hard to characterise using individual names for each distinct technique of data organisation. Additionally, so many names have already been used that introducing new names just aggravates the confusion. As it is now, one language's 'array' (FORTRAN) is another's 'sequence' (VERS2) or 'table' (COBOL).

3. An analysis model of aggregate data structures

This section presents an axis/answer model for a large class of aggregate data structures. No enumeration of this class is provided beforehand since aggregates are well known and because the following description of the model will make the class boundaries clear. As has been noted, the model is sufficiently general to characterise the aggregate data structures found in most existing programming languages and data base management systems.

Fig. 4 describes the model; it lists the five axes in the form of questions and gives a short description (immediately below each question), an abbreviation, and the list of answers for each axis (the latter two on the right of the figure). Each axis will be discussed below; however, an introductory example will show how they can be used. The simple array, as shown in Fig. 1, provides structure via indexing for a fixed size collection of numbers. Such an array could be modelled as follows (using the abbrevaiations from Fig. 4):

```
Array     Homogeneous: YES
          Basic items: YES
              Ordered: YES
               Number: FIXED
       Identification: NUMBER
```

In other words, the traditional array is modelled as an aggregate data structure which orders a fixed size collection of homogeneous, indivisible elements each of which is identified by a number (or 'index').

This modelling exercise defines a 'bare' array; the essence of 'arrayness' is captured by the values given for each axis above. What has been described might be called an abstract data structuring technique; to define a particular data structure, such as the array of Fig. 1, some additional information is needed. This additional information and the answers for each axis are conveniently presented pictorially as shown in **Fig. 5.** First, the kind or type of elements which compose the aggregate are listed at the bottom of the pictorial data definition. In the case of SALHIS, there is just one kind of element ('Homogeneous' YES). The element's type is enclosed within an inner box to re-inforce the idea that multiple instances of that type are present in the array.

Second, additional information provides further, specific

PASCAL

```
ID = 66001 .. 66999
SKILL = RECORD  SKILCODE: 1000 .. 1999;
                SKLYRS: INTEGER
        END;
PERSON = RECORD
         EMPNAME: ARRAY [1..20] OF CHAR;
         SKILLS: ARRAY [1..10] OF SKILL;
         SALHIST: ARRAY [1..5] OF 400.0..2500.0
         END;
PEOPLE = ARRAY [ID] OF PERSON;
```

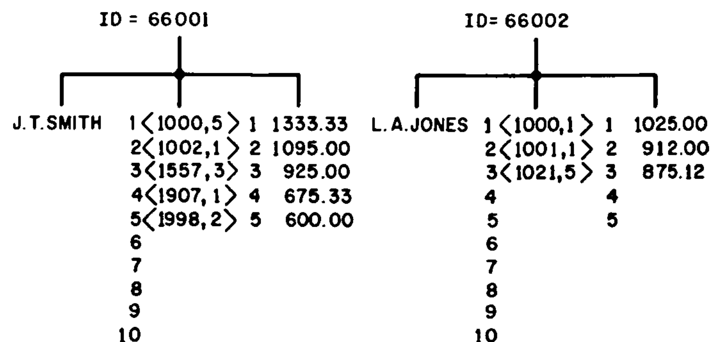

Fig. 3

Downloaded from https://academic.oup.com/comjnl/article/21/2/98/477635 by guest on 19 April 2024

1. ARE THE ELEMENTS <u>HOMOGENEOUS</u>? ARE ALL INSTANCES OF THE ELEMENTS DRAWN FROM THE SAME DATA DEFINITION? — HOMOGENEOUS: YES, NO

2. ARE THE ELEMENTS <u>BASIC ITEMS</u>? ARE ALL INSTANCES OF THE ELEMENTS ATOMIC AND INDIVISIBLE? — BASIC ITEMS: YES, NO

3. ARE THE ELEMENTS <u>ORDERED</u>? IS ANY ORDERING AMONG THE ELEMENT INSTANCES IMPOSED OR IMPLIED BY THE STRUCTURE? — ORDERED: YES, NO

4. WHAT IS THE <u>NUMBER</u> OF ELEMENTS? HOW MANY INSTANCES OF EACH KIND OF ELEMENT ARE COMBINED IN ONE INSTANCE OF THE AGGREGATE? — NUMBER: FIXED, LIMITED, UNBOUNDED

* WHEN "HOMOGENEOUS" IS NO, "NUMBER" MAY BE EXTENDED TO SPECIFY A DIFFERENT COUNT FOR EACH KIND OF ELEMENT.

5. HOW IS AN ELEMENT <u>IDENTIFIED</u>? HOW IS AN INDIVIDUAL ELEMENT INSTANCE NAMED, LABELED, OR IDENTIFIED WITHIN AN AGGREGATE INSTANCE? — IDENTIFICATION: NUMBER, NAME, POINTER, NONE

Fig. 4

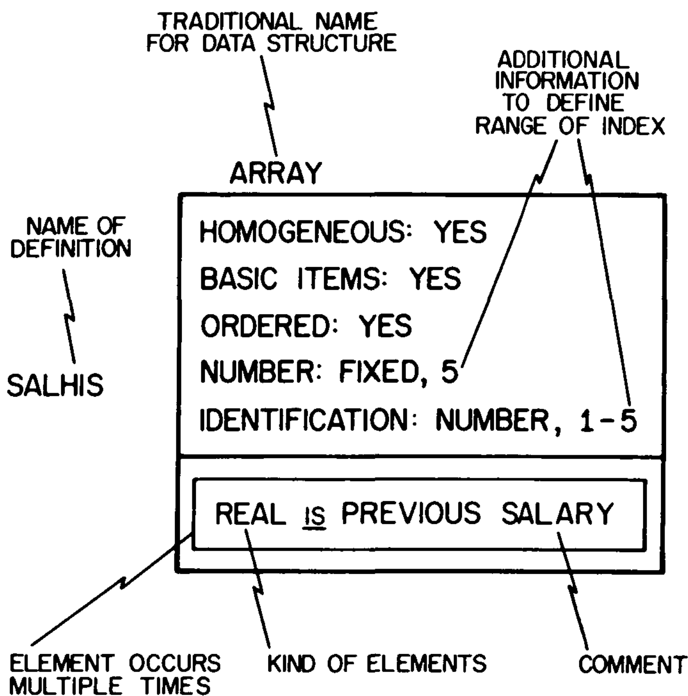

Fig. 5

details to the answers to the five axis questions. These details turn an abstract data structuring technique into a concrete data structure definition. In Fig. 5, the answer to the 'Number' question specifies that a fixed number of element instances occur in each array structure; the additional information indicates that this fixed number is five.

Third, the pictorial data definition of Fig. 5 contains a few other special features which make it easier to use. The definition is given a name so that it may be referred to conveniently and used in other definitions; this name appears on the left side of the data definition block. A traditional or descriptive name for the data structuring technique modelled may optionally appear above the block. Finally, the special notation 'is' precedes commentary information.

3.1 *The axes of the model*

Each axis shown in Fig. 4 will now be explained. Whenever possible, examples will be drawn from the array examples of Figs. 1 to 3. The abbreviated axis names from the right of Fig. 4 will be used in quotes (e.g. 'Homogeneous' for the first axis) in the following discussion.

The SALHIS array as modelled above contains data elements of exactly one kind or type so the 'Homogeneous' axis is answered YES. However, other aggregate structures need not consist of like elements. The aggregate commonly called a hierarchy, may also be known as a structure (PL/I), group (COBOL), class (SIMULA), or record (PASCAL). This data structure groups together elements of different types. The PERSON data structure of Fig. 3 (not to be confused with the PEOPLE array) is of this kind; it groups together three different kinds of arrays. The hierarchy structure in general can be modelled:

Hierarchy	Homogeneous:	NO
	Basic items:	NO
	Ordered:	NO
	Number:	FIXED
	Identification:	NAME

'Homogeneous' is answered NO because most languages allow any number of different kinds of elements to be part of a hierarchy structure. In summary, the 'Homogeneous' axis notes whether or not all the instances of the aggregate's elements are drawn from the same definition.

The second axis makes a distinction between atomic and complex constituents of an aggregate. The term 'basic items' denotes data which are not themselves further decomposable. The FORTRAN array of Fig. 1 has only basic items (of one

Downloaded from https://academic.oup.com/comjnl/article/21/2/98/477635 by guest on 19 April 2024

certain kind—regular numbers) as its elements. This array is modelled in Fig. 5 where 'basic items' is answered YES. However, many programming languages allow arrays of arbitrary data types, including other aggregates. The SKILLS array of Fig. 2 does not consist of basic item elements; each element can be viewed as a hierarchy structure. Such a view is represented by the modelling shown in **Fig. 6**. The SKILLS array (shown at the top of Fig. 6) has 'basic items' NO because each element is an instance of the SKILL hierarchy definition. The SKILL aggregate itself has 'basic items' YES since its elements are integers.

Encoding this distinction between basic item and more complex elements as an axis in the model reflects an important distinction between data structuring techniques. An array of numbers is basically different from an array of hierarchies. On the other hand, most people would agree that an array of reals and an array of integers are both of the same genre.

Most common data structures impose an order on their constituent elements. This need not always be so. The 'Ordered' axis makes this distinction explicit. The mathematically-inspired set structure simply groups together a collection of elements. Set data structures are provided by several programming languages, including MADCAP VI (Wells and Morris, 1972), SETL (Schwartz, 1973), and VERS2. When an aggregate imposes no order on its elements, 'Ordered' is answered NO. Set data structures which are true to their mathematical heritage may be modelled:

Set		
	Homogeneous:	YES
	Basic items:	NO
	Ordered:	NO
	Number:	UNBOUNDED
	Identification:	NONE

The hierarchy structure discussed above need not imply any order among its named components. In this case, it may be modelled as shown in Fig. 6 and **Fig. 7.** If for any reason an ordered hierarchy, such as provided by PL/I (IBM, 1972), is desired, the 'Ordered' answer may be changed to YES to reflect the true nature of the data structure.

Most aggregate data structures do impose some sort of order relationship among their elements. The SALHIS, SKILLS, and PEOPLE arrays of Figs. 1 to 3 are all ordered; their models in Figs. 5 to 7 all show 'Ordered' YES. Additional information may be specified with the YES answer to indicate the kind of ordering if it is not obvious. For instance, a 2-dimensional matrix could specify either row or column ordering.

Some programming languages require the user to declare the number of data element instances which can be in an aggregate data structure. In the traditional array of Fig. 1, the number of data element instances must exactly equal the range of the index. Alternatively, a programming language may allow the number of component data elements to vary but up to some specified maximum (which is stated in the data declaration statement). Such a method is used by PL/I for arrays with the 'varying' attribute. A few languages and several data base management systems provide aggregate data structures with varying numbers of components and for which the user need not specify any upper bound. For example, ALGOL 68 (Lindsey and van der Meulen, 1973) allows 'multiples' (similar to arrays) of unlimited size using 'flex'. Thus, while the first three axes are yes/no questions, the 'Number' question allows three answers—FIXED, LIMITED, and UNBOUNDED. The VERS2 SKILLS sequence of Fig. 2 (modelled in Fig. 6) is a data structure of UNBOUNDED 'Number'. (Of course, some arbitrary limit on the number of components may be made by the implementation of such languages; however, the important point is that the programmer is not responsible for explicitly limiting the data structure.)

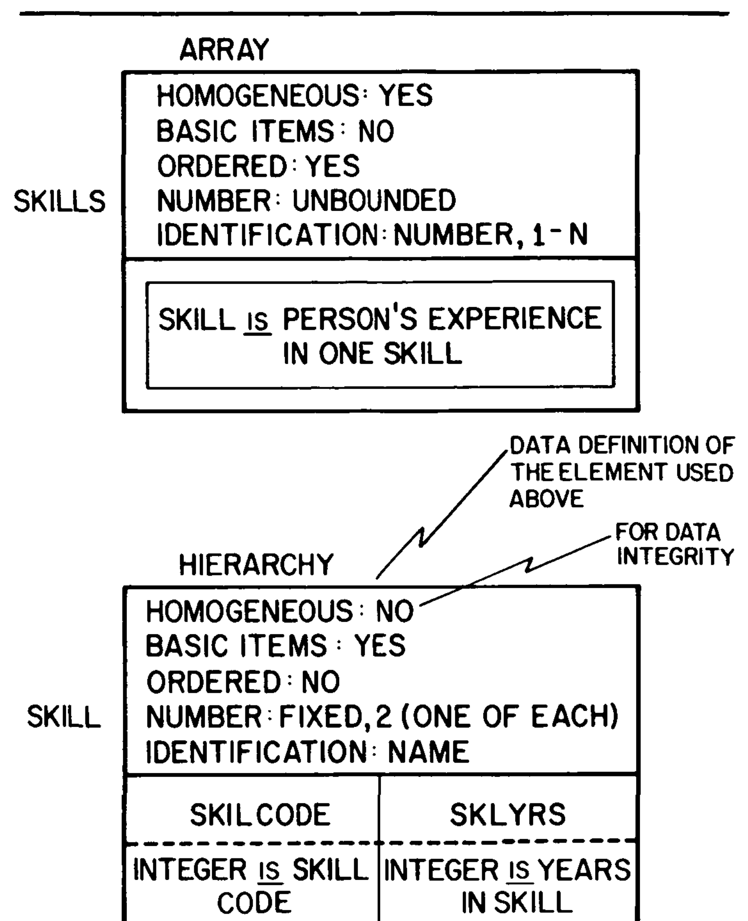

Fig. 6

When the 'Homogeneous' question is answered NO, the 'Number' axis may be extended to specify a cardinality for each kind of data element. **Fig. 8** shows a simple, contrived example which demonstrates the need for this extension. Without the extension, the two data structures shown cannot be distinguished. Both are simple sets consisting of exactly two elements where the elements may be integers and strings. However, the lefthand data structure is restricted so that every instance contains one integer and one string. To maintain data integrity, this restriction may be made part of the model. Thus, when 'Homogeneous' is answered NO, 'Number' may optionally be extended to specify a different count for each kind of element. This extension is not always needed, as demonstrated by the righthand data structure in Fig. 8.

'Identification', the last characteristic axis for aggregates, is also not a yes/no question. The purpose of 'Identification' is to indicate how a specific constituent data element is named, pointed to, or labelled. 'Identification' is a general term for this concept; the possible answers, as listed in Fig. 4, are NUMBER, NAME, POINTER, and NONE. The traditional array data structure is indexed or accessed by use of a NUMBER as an index. In the array of Fig. 1 (modelled in Fig. 5), each element is labelled with a number from one to five. Users of this sort of array think of this number as identifying an element of the array. Similarly, the elements of an n-dimensional matrix are identified by an ordered n-tuple of numbers.

The concept of the 'Identification' axis results from generalising the ideas of an array or matrix index so that it can apply to other aggregates. A hierarchy structure, as discussed and modelled above, distinguishes its elements by NAME. For the SKILL hierarchy of Figs. 2 and 6, these names are SKILCODE and SKLYRS. In contrast, a true mathematical set does not provide any way whatsoever of identifying its ele-

Downloaded from https://academic.oup.com/comjnl/article/21/2/98/477635 by guest on 19 April 2024

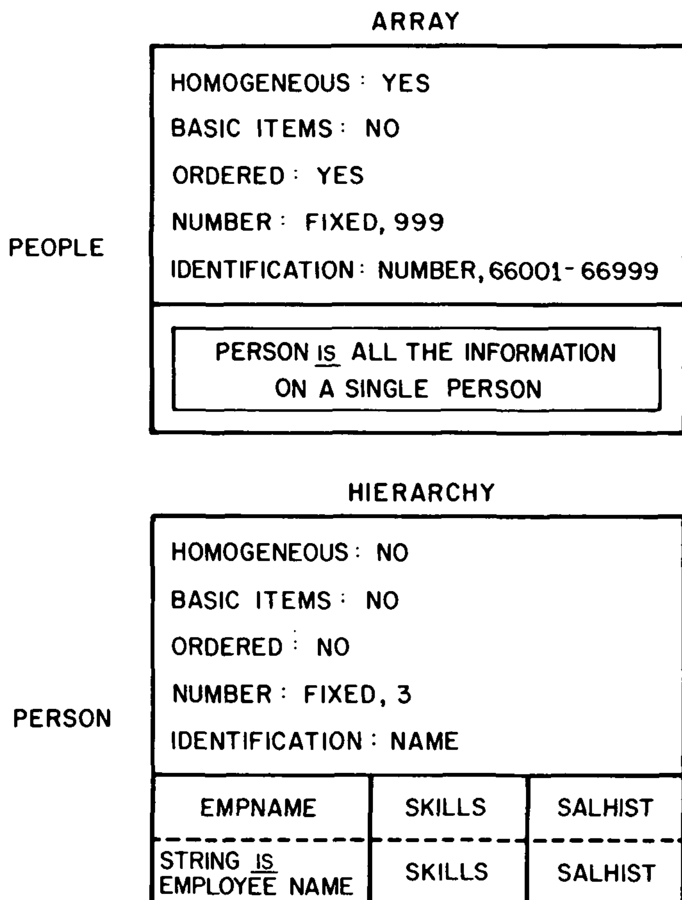

Fig. 7

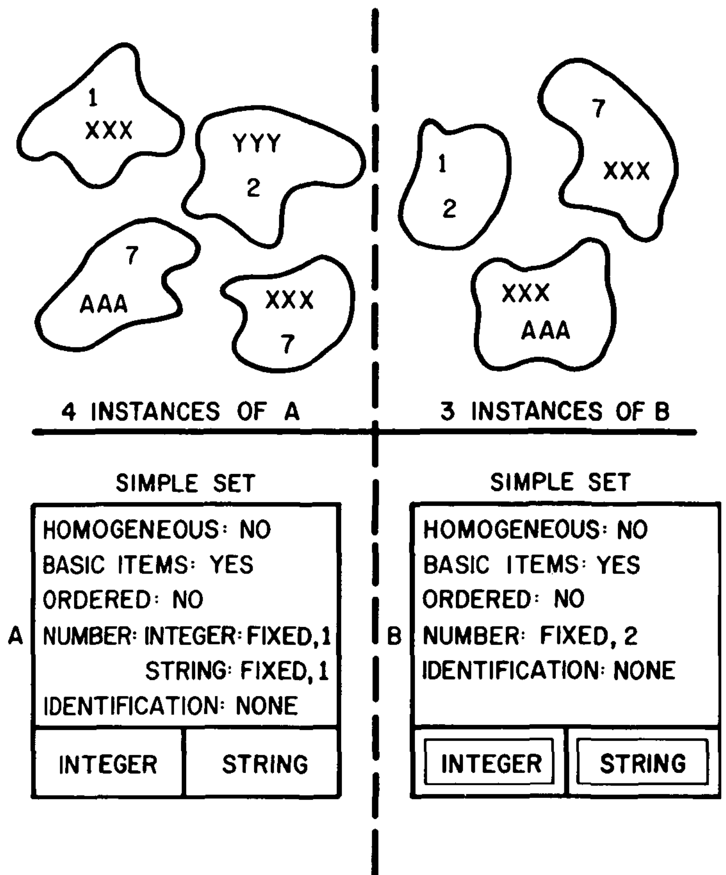

Fig. 8

ments. In this case, 'Identification' is answered NONE (as shown above where a set is modelled) because set elements are manipulated only with operations such as union and intersection; there is no way to select a specific, individual element of a set. (VERS2 does, however, provide a limited way to select or sequence through the elements of a set with 'range expressions'). The programming language PASCAL originally provided a data structure ('class') which is very much like a set of limited 'Number' except individual elements of it may be selected by use of a POINTER which is created whenever new elements are added.

The four possible answers to 'Identification' do not exhaust all the possibilities. More detailed answers may be appropriate in some cases. For example, it might be useful to distinguish between different kinds of NAME identifiers: unique, system-wide names versus names which are valid only within a specified scope. Thus 'Identification' may be adapted, within reason, to various applications.

The discussion of the five characteristic questions for modelling aggregates is now complete. The next section presents some more examples to demonstrate how the model can be used.

3.2 *Using the model*

First, consider a few miscellaneous data structures. In the previous section a set was modelled using the five axes. The particular data structuring technique selected was true to the mathematical definition of set in that the members of the set could be complex, nonbasic items (for examples a set of sets). However, a set consisting only of basic items may be desirable; such a set can be easily represented by the model, without introducing any new names, as follows:

Simple Set	Homogeneous:	YES
	Basic items:	YES
	Ordered:	NO
	Number:	UNBOUNDED
	Identification:	NONE

(The name on the left is added for convenient reference only.) Note that only the answer to the 'Basic items' axis is changed from the mathematical set of Section 3.1. A further possible modification is to limit the size of the set by changing the 'Number' axis, as follows:

Bounded Set	Homogeneous:	NO
	Basic items:	YES
	Ordered:	NO
	Number:	LIMITED
	Identification:	NONE

The elements of the bounded set must all be the same; however, a set consisting, say, of both real numbers and integers could be modelled again by changing just one axis:

Varied Set	Homogeneous:	NO
	Basic items:	YES
	Ordered:	NO
	Number:	LIMITED
	Identification:	NONE

The point to be made is that numerous different kinds of sets can be modelled precisely without any confusion as to exactly what is implied bythe term 'set'.

Looking at the entire aggregate model in Fig. 4, it is interesting to speculate on the total universe of data structures as modelled by the five questions. Clearly there are at least $2^3 \cdot 3 \cdot 4 = 96$ different possible ways of picking an answer for each axis question (since there are three yes/no questions, three possible answers to 'Number' when it is not extended for reasons of data integrity, and at least four possible answers to 'Identification'). Thus the model would seem to imply the existence of 96 distinguishable data structuring techniques for aggregates.

Downloaded from https://academic.oup.com/comjnl/article/21/2/98/477635 by guest on 19 April 2024

Picking a random set of answers as an experiment, consider:

???	Homogeneous:	NO
	Basic items:	YES
	Ordered:	NO
	Number:	UNBOUNDED
	Identification:	NAME

which seems to indicate an aggregate of different type basic items selected by name. This data structure seems appropriate for a symbol table: the identification names are symbol names used by a direct access technique to retrieve a data value associated with each symbol. The values may be various types (e.g. integer, real, address), all of which are basic items.

The aggregate model can be used in a similar manner to study the universe of possible aggregate data structures. For example, some axis values can be held constant while others are varied. This procedure, in effect, explores the various restrictions and generalisations which are possible from the original data structure.

4. The role of analysis-style models

The preceding section has introduced a model of aggregate data structures which captures their true variety as found in contemporary programming systems. This type of model, based upon an analysis of existing, practical data structures, is unusual; this section comments first on the aggregate model and then on the general use of this sort of modelling method.

The aggregate model of Section 3 is an intelligible, readable, compact, easy to use model of a fairly large class of data structures. The model is particularly useful for a 'check list' approach to data structure design. Each axis can be considered separately and the proper characteristics for a particular application selected by picking the proper answer to each question. Since each axis is independent of the others, the degrees of freedom available to the designer are kept under control; at each stage only a single question need be considered. Such an approach is in direct contrast to the typical method of trying to intuit the best data structure from among all those available.

The data structure model presented here is intended to represent logical, conceptual data structures. For example, the unordered sets detailed in Section 3.2 may be modelled as being truly unordered regardless of the fact that most implementations will have to store the elements sequentially. Since the model's chief purpose is to further the understanding of data structures, it is of paramount importance that users concentrate on the conceptual data structure and ignore extraneous details which are assumed by implementations. As another example, an array can be modelled without assuming it is stored contiguously in memory. The guideline should be to make sure no additional, unnecessary organisation is imposed on the data structure under consideration. The model facilitates this approach by making clear the choices which are available for each data structure characteristic.

The aggregate data structure model is a contribution to the understanding and use of data structures. Specifically, it is valuable for the following:

1. To teach data structures in a language-independent manner
2. To choose and contrast data structures for practical programming tasks
3. To design and document data bases in an easy to read manner.

All these benefits arise from this work's success at modelling a wide class of common, real world data structures in a way which reveals their true substance.

A possible extension of the aggregate data structure model would be its implementation as an 'automatic programming' system (see Floyd, 1972). The model is a nice, descriptive technique for describing the logical structure of data. Then, unfortunately, the user is 'left hanging' as to how to implement the data structure. It seems that the axes and their answers could be used to select automatically ways of implementing a wide class of data structures. A feasible approach may be to associate a collection of implementation techniques with individual axis answers. Thus, the answers to each axis would suggest a set of possible implementations for a given data structure. Earley suggested a similar idea, termed an 'implementation facility' (Earley, 1971), for the VERS language. The resulting automatic programming tool would provide a replacement for the numerous kinds of data declarations used now in programming languages and data base management systems.

At any rate, the general concept of an analysis-style model should have wide applicability. The approach allows the answering of questions arising from data structures as they actually exist in today's systems. In comparison with other approaches to modelling, analysis seems to be both more practical and general. By its very nature, it yields a representation of the true structure of data bases. In contrast, many existing models might be called 'prototype' systems; they provide a single framework which can be used to imitate real world data structures. Examples of this approach include VERS, 'data graphs' (Rosenberg, 1971), 'list structures' (Fleck, 1971), and DIAM (Senko *et al.*, 1973). In addition, existing higher level approaches to data base management such as ALPHA (Codd, 1972) and 'structured data structures' (Shneiderman and Scheuermann, 1974) operate at a higher semantic level which assumes a structural description is available. The analysis model for aggregates provides this structural description.

The analysis approach of examining what already exists has been applied to two other classes of data structures (Honig, 1975). These two classes ('associations' and 'files'), together with the aggregates discussed here, represent all the data structures which were identified in a survey of 21 representative programming languages and data base management systems. (The systems surveyed were: ALGOL 68, BLISS, COBOL, EL1, FORTRAN, MADCAP VI, PASCAL, PL/I, SETL, SIMULA, a typical assembly language, VDL, VERS2, ALPHA, CODASYL DBTG, IDS, IMS, LEAP, MacAIMS, RIQS, and TDMS.) This exercise suggests the generality of the analysis-style model and will, hopefully, prompt further use of the method in other areas such as data access and data integrity.

5. Conclusion

The model of aggregates succeeds in describing data structures without using either new or old names for each particular data structure. Instead attention is concentrated on the basic structural differences between data structures: each question and its possible answers describe one axis along which data structures may vary. Thinking about data structures has been moved to a higher plane, a level above that provided by individual names. The programmer or designer need no longer choose between PL/I arrays, PASCAL powersets, EL1 self products, COBOL tables, SETL tuples, CODASYL DBTG sets, IDS group items, TDMS repeating groups, and so on, *ad infinitum*. Instead, a fixed set of relatively independent questions can be asked and answered one at a time.

6. Acknowledgements

Some previous work on data structures led to the recognition of 'analysis' as an approach to modelling. These works are Smith (1972); Prywes and Smith (1972); Hsiao and Harary (1970); each of these might properly be called analysis models

Downloaded from https://academic.oup.com/comjnl/article/21/2/98/477635 by guest on 19 April 2024

covering different classes of data structures than the aggregates presented here. Also, we wish to thank Ben Mittman of Northwestern University for his advice and for helping to test the aggregate model. Finally, we must acknowledge that Jay Earley's work (1971) not only kindled our interest in data structures, but also motivated the title.

References

AMERICAN STANDARDS ASSOCIATION (1966). *American standard FORTRAN*, X3.9-1666, American Standards Association, New York.
CODD, E. F. (1972). Relational completeness of data base sublanguages, *Data Base Systems*, Rustin, R., ed., Prentice Hall, Englewood Cliffs, NJ, pp. 65-98.
EARLEY, J. (1971). Toward an understanding of data structures, *CACM*, Vol. 14, No. 10, pp. 617-627.
EARLEY, J. (1973). Relational level data structures for programming languages, *Acta Informatica*, Vol. 2, pp. 293-309.
FLECK, A. C. (1971). Towards a theory of data structures, *J. Computer and System Sciences*, Vol. 5, No. 5, pp. 475-488.
FLOYD, R. W. (1972). Toward interactive design of correct programs, *Information Processing 71*, North Holland Publishing Co., Amsterdam, pp. 7-10.
HONIG, W. L. (1974). Bringing data base technology to the programmer. *FDT*, Vol. 6, No. 3, pp. 2-15.
HONIG, W. L. (1975). A model of data structures commonly used in programming languages and data base management systems, Ph.D. thesis, Northwestern University, August 1975.
HSIAO, D. and HARARY, F. (1970). A formal system for information retrieval from files, *CACM*, Vol. 13, No. 2, pp. 67-73.
IBM (1972). PL/I(F): language reference manual, International Business Machines, Form GC-28-8201-4.
LINDSEY, C. H. and VAN DER MEULEN, S. G. (1973). *Informal introduction to ALGOL 68*, North Holland Publishing Co., Amsterdam.
PRYWES, N. S. and SMITH, D. P. (1972). Organization of information, *Annual Review of Information Science and Technology*, Vol. 7, Cuadra, C.A., ed., American Society for Information Science, Washington, DC, pp. 103-158.
ROSENBERG, A. L. (1971). Data graphs and addressing schemes, *J. Computer and System Sciences*, Vol. 5, No. 3, pp. 193-238.
SAMMET, J. E. (1969). *Programming languages: history and fundamentals*, Prentice Hall, Englewood Cliffs, NJ.
SCHWARTZ, J. T. (1973). On programming: an interim report on the SETL project, installment II, Computer Science Department, Courant Institute of Mathematical Studies, New York University, October 1973.
SENKO, M. E., ALTMAN, E. B., ASTRAHAN, M. M. and FEHDER, P. L. (1973). Data structures and accessing in data base systems, I, II, and III, *IBM Systems J.*, Vol. 12, No. 1, pp. 30-93.
SHNEIDERMAN, B. and SCHEUERMANN, P. (1974). Structured data structures. *CACM*, Vol. 17, No. 10, pp. 566-574.
SMITH, D. P. (1972). An approach to data description and conversion, Ph.D. thesis, University of Pennsylvania, University Microfilms, order number 72-17, 425.
WELLS, M. B. and MORRIS, J. B. (1972). The unified data structure capability in Madcap VI, *International J. Computer and Info. Sci.*, Vol. 1, No. 3, pp. 193-208.
WIRTH, N. (1971). The programming language Pascal, *Acta Informatica*, Vol. 1, No. 1, pp. 35-63.

First Vice-Chancellor for computer profession

Ewan Page was a research student in the Statistics Laboratory at Cambridge from 1951-54. This was the period when the EDSAC came into service and Page was so much taken by it that he was seen as much in the Mathematics Laboratory as in the Statistics Laboratory. He had a foretaste of later prosperity when he won a prize of £5000 in a football pool, having filled in the coupon, so it was rumoured, with the aid of Tippet's random sampling numbers. Perhaps he felt that this was unprofessional behaviour for a statistician for he decided to make computing rather than statistics his life's work.

He built up the Computing Laboratory at Newcastle and it was here that he acquired a taste for university administration. He sharpened his teeth on the Computer Board and in doing so demonstrated his prowess to such an extent that the University made him a pro-Vice Chancellor. When Dr Henry Miller died suddenly in the summer of 1976 Ewan Page stepped into the breach and served with distinction as acting Vice Chancellor.

He now becomes a Vice Chancellor in his own right, at Reading University, and will be the first Fellow of the British Computer Society to do so. All readers of The Computer Journal will give him their best wishes.

MAURICE WILKES

Professor Ewan Page

Downloaded from https://academic.oup.com/comjnl/article/21/2/98/477635 by guest on 19 April 2024