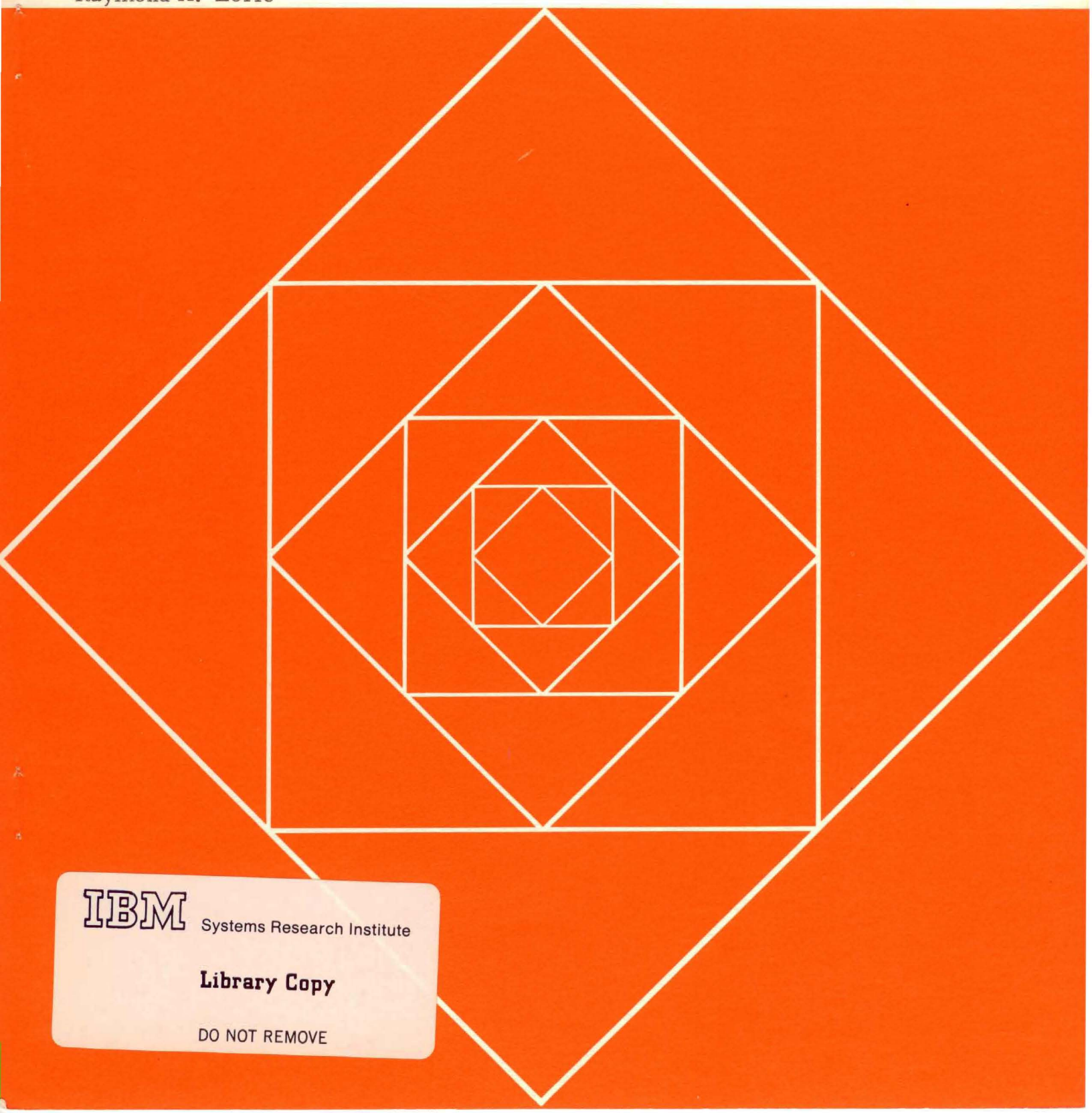
CAMBRIDGE SCIENTIFIC CENTER

G320-2067
March, 1971

IBM
Data Processing Division

APL as a Language for Handling a Relational Data Base

Raymond A. Lorie



G320-2067

March, 1971

APL AS A LANGUAGE FOR HANDLING A RELATIONAL DATA BASE

Raymond A. Lorie

IBM Corporation
Cambridge Scientific Center
545 Technology Square
Cambridge, Mass. 02139

APL AS A LANGUAGE FOR HANDLING A RELATIONAL DATA-BASE

R.A. Lorie

International Business Machines
Cambridge Scientific Center
Cambridge, Massachusetts.

Abstract

APL has received a good acceptance as a language to be used in an interactive system. The power of an APL system can be dramatically increased by providing data-base capabilities. The language may then be used for creating, updating and retrieving information.

We believe that the most promising general approach to modelling a wide variety of data-structures is relational. We show how the APL language may be used to handle a relational data-base.

The basic element of data is called a data-item. Some data-items, called entities, are given a unique identifier. Data-items may be linked to entities as attributes. Entities may be linked between themselves by relations. By introducing the universe of discourse as being the ordered set of all identifiers, a relational model becomes an "array" model as in APL and the power of expression of the APL language is automatically available.

The arrays representing attributes and relations are sparse and require an appropriate physical representation. But this paper is concerned only with a logical view of the data and does not propose any implementation.

Key-words

language, APL, data-base, data-bank, files, relation, relational model, information retrieval, problem-solving, interactive.

CR categories

3.64, 3.70, 3.73, 3.74, 4.29.

INTRODUCTION

The last few years have seen spectacular developments in the functional capabilities, performance and availability of time-sharing systems. By creating a new interactive environment for computer usage they have stimulated interest in new applications and in new approaches to more conventional ones. But the crucial importance of response time, the complexity of new applications, the fact that the user is more probably an application specialist rather than a professional programmer have contributed to make some needs more apparent.

Let us try to get more insight into these requirements by considering an inquiry system.

In such a system the organization of data on auxiliary storage is most important. Information must be stored and retrieved with good performance. This generally requires a sophisticated data-base structure. A language must be available to specify the queries. This requires not only logical but also arithmetic and editing capabilities. It is also necessary to be able to keep track of the data structure when information is brought into core for further selection and processing.

But it is interesting to note that interactive applications which seem to be very different in nature often tend to require very similar or even identical features. And the requirements for an inquiry system are also valid for any on-line problem solving application.

We summarize these requirements as follows:

- handling of data structures in core and on auxiliary storage
- use of a general language for retrieval and processing of data.

As already mentioned, an interactive environment asks for tools which serve best the needs of application programmers. We feel that a new approach to data organization, at a purely logical level, would facilitate the development of interactive applications. It must be emphasized that these requirements are only more apparent in interactive applications. They do hold for data processing in general. Some applications will use powerful data structures in core; some others will be more oriented toward the manipulation of a data-base. But most applications require simultaneously data structures and data base facilities. They would therefore also benefit from a logical and integrated view of data. Such a view is proposed in this paper. It is based on four concepts which concern successively the data structure, the language, the information coding and the array representation of relations.

CONCEPTS

Data structure

Most programming languages support some kind of basic data structures, for example vectors and arrays. Others include features needed to handle more complex structures such as pointers, lists and rings. On auxiliary storage the structure of data generally appears under another form: fields in records, records in files, indexes etc. This dual view of data is rather artificial in the sense that it has been imposed by the hardware technology. It is responsible for a drawback in programming. Input/output statements are required as soon as auxiliary storage must be used.

On a purely logical level the user wants to address data in the data universe. He should be unaware of a dual core/auxiliary storage residence. (This does not preclude the existence of a dual view at a lower level in order to take full advantage of the available hardware.)

But even the assumption of an "infinite" core is only a partial solution because the user wants to address data not by core address but rather by references meaningful to his application. In order to progress in such a direction a new logical view of data is needed. We feel that the relational approach proposed by E. Codd (2) constitutes an excellent frame for a new data definition.

For reference, let us recall Codd's definition of an n-ary relation. Given n sets S1,S2... Sn not necessarily distinct, an n-ary relation R on these sets is a set of n-tuples, each of which has its ith element in Si. Si is called the ith domain of R. In order to refer to some specific domains of R, the domain name may be used only if it is unique in the relation R. If not, a role name must be used to specify a domain without ambiguity.

At this point it is sufficient to understand intuitively that elements involved in relations are atoms of data, also called data-items. They will be defined later.

> Example : for illustration we shall use an example suggested in (2). In a firm there exists a set of projects, a set of parts, a set of suppliers. A relation
>
> Supply (part,project,supplier)
>
> is a set of n-tuples. Supply is the name of this set. The first element of an n-tuple is a part, the second one a project, the last one a supplier. It indicates what parts are supplied to which project, by what supplier.

But relations may also be used to specify complex data structures in core. Consider a binary relation R(S,S)

defined on two identical domains S. It may be used to organize cells into a list. The domain S is the set of cells. A 2-tuple (si,sj) is a member of the set R if the cell sj follows immediately the cell si in the list. If one assumes that two particular cells 'top' and 'bottom' are also members of S, then the list is represented by R containing the following n-tuples :

top,s1
s1,s2
s2,...
si,sj
...,bottom

Language requirements

Several languages exist for specification of processing in an interactive environment. It is beyond the scope of this paper to analyze or compare the characteristics of these languages. But the power of expression of APL (5) (7) (8) (9) (11) has been recognized and its acceptance enhanced by the increasing number of users in different fields.

Specialized languages have also been proposed for specification of data retrieval, for queries for example (3). These languages must always be imbedded in a general purpose language in order to allow processing of retrieved information. We feel that an integrated approach, a single language with a uniform syntax, should be used. We shall see how APL can be used for such a purpose.

In this paper we always refer to APL as a language, never as a particular implementation or system.

We suppose the reader is familiar with the general concepts of APL. References to the language will use only basic statements and will be annotated. Only the final example requires a more extensive knowledge of APL.

A remark is necessary concerning the level of language which is considered here. The description of a computer central processing unit implies the description of the data representation or data model (character, word, address) and the definition of a "machine language". A machine language is a set of instructions (or statements) which specify operations to be performed on the data. The same relationship exists between the higher level language APL and the data model on which it operates. The APL data model is based on arrays. But what about the common criticism on APL: how to introduce input/output operations, how to implement complex data structures?

Our answer consists of defining a more powerful relational data model, on which the APL language may operate, without any syntax alteration. The scope of this paper is precisely the definition of this model. We do not intend to discuss the implementation of such a machine. We are looking only at the APL interface.

It is also clear that a higher level language could be designed. Statements would then be compiled or interpreted into APL form.

Information coding

Any representation of data in a computer implies an internal coding. We now define the coding schema used in the model.

One must first define the atom of information, the smallest piece of data which can be involved in a relation. This notion is intuitive. But we define it more formally by reference to the APL definition of a variable.

A data-item is a scalar or an array. The scalar or any element in the array may have one of the following types: logical (0-1), integer, floating point, character. All elements in an array have the same type. A single element of an array can be selected by specifying its indices. The number of indices required is called the rank of the array. A function ρ applied to an array A yields the size (or shape) of A, that is, a vector whose components are the dimensions of A. The ith dimension represents the number of different values which can be assigned to the ith index.

Going back to the definition of a relation, we now say that the elements in the sets S are data-items. A data-item is atomic as far as relations are concerned.

Several coding mechanisms may be used to reference a data-item. Let us first consider two opposite mechanisms.

a) The internal machine code is used to represent the numbers or characters of the data-item. This means that any reference to a data-item is done by specifying the whole value of the data-item. If it must be stored the whole value is stored. If the data-item is fetched, the operation returns the value. This is very convenient if the data is referenced only once. If it is referenced several times it becomes uneconomic as far as space is concerned. The highly variable format of different data-items could also constitute a major problem.

b) Each atom of information within the system is assigned a unique identifier (referred to as id). We assume, without loss of generality, that the id is a positive integer. All relations between data-items are stored by using the id's. Such coding has been used in (6) and (12). It assures a good use of space as the id's generally occupy less space than the data-items themselves. It also standardizes the format. But for each such data-item one introduces a new abstract data-item called entity. The original data-item is in fact an attribute of the entity. This is consistent with a definition of entity given by N. Webster: "the existence of something as contrasted with its attributes or properties".

In the proposed data model we adopt the mechanism b) for the representation of relations between data-items, but we also allow the mechanism a) to be used for a common special case. This is illustrated by an example.

Consider the table A (it could also be a file) containing the descriptions of parts.

Part Number	Description
M1637FT12	description.....
G123888dd	description.....
.....	.....

Table A

Such information can be coded by defining entities called PART with an attribute PART_NUMBER, entities called DESC with an attribute DESCRIPTION, and a relation

D (PART_Id, DESC_id)

relating each entity PART to the corresponding entity DESC. But this is probably the only place where the descriptions are ever used, and the introduction of an id for each of them offers no advantage. If we consider each description as being the value of an attribute DESC of an entity PART, the information concerning the descriptions can be specified by

DESC (PART_Id) = description

Correspondingly

PART-NUMBER (PART_Id) = number

In summary, several attributes may be defined for an entity. A given attribute takes a single value for a given entity.

Array representation of relations

The three previous concepts lead us to the problem of representing attributes and relations using APL syntax.

The functional expressions,

Supply (id_part_1,id_project_2,id_supplier_3) = true
= 1 (2)

attribute (id) = value

for the specification of relations and attributes are quite natural. But APL - or any other programming language - does not allow this use of a function at the left of the assignment sign. By introducing U, the universe of discourse, as being the set of all possible id's, an n-ary relation may be seen as a logical n-dimensional array where indices are id's of entities and when a particular element has a value 1 if and only if the corresponding n-tuple of id's satisfies the relation. Expressions (2) become, using

APL notation

Supply [id_part_1,id_project_2,id_supplier_3] ← 1

attribute [id] ← value

A n-ary relation has thus an array form

R [i1;i2;i3....;in]

A geometric interpretation follows immediately. A n-ary relation is a set of points in a n-dimensional space (where only integer coordinates are used). For example a relation R containing the 2-tuples (2,3), (4,2) and (5,3) may be represented by A,B,C in Fig. 1.

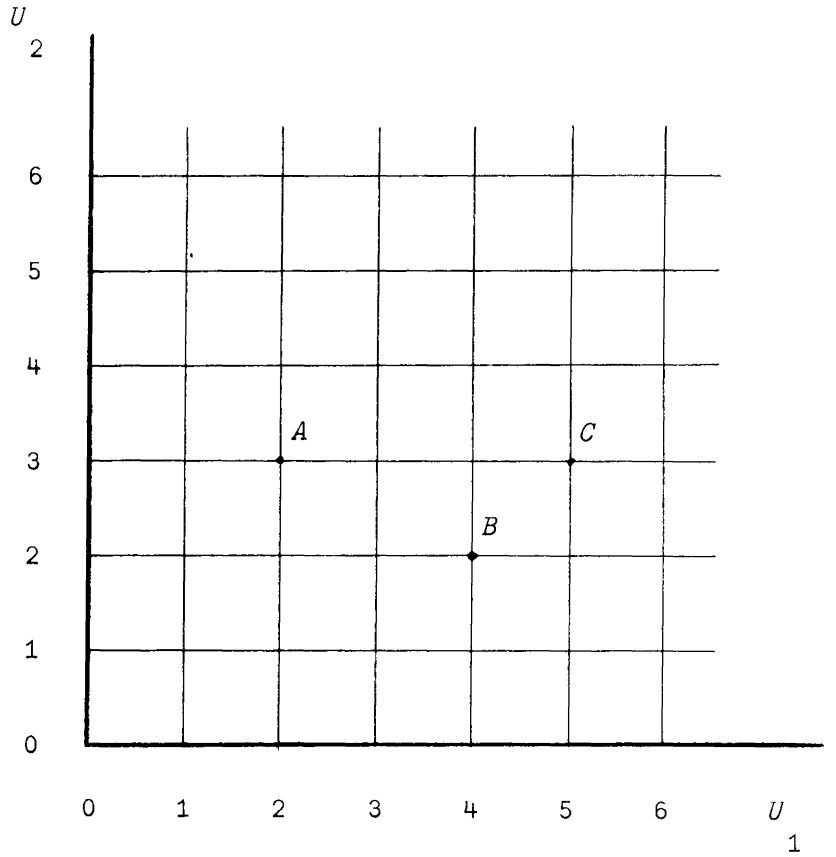


FIG. 1

DATA MODEL

Universe

Let us denote by N the maximum number of entities in the system. Then the Universe of discourse is defined as

$$U \leftarrow \iota N$$

U is the integer vector (1 2 3 ... N).

Entity_class

Entities may be grouped into sets called Eclasses. Each Eclass is referred to by an id. The characteristic of the set of Eclasses is defined as a vector ECLASS of N logical elements where ECLASS [i] has a value 1 if and only if i is the id of an Eclass. This vector is initialized by

$$ECLASS \leftarrow N \rho 0$$

Name_of_an_entity

NAME is defined as an array of characters of size (N,n) where NAME [i;] is a character vector associated with i. Name is initialized by

$$NAME \leftarrow (N,n) \rho \text{' '}$$

At this level we ignore the fact that the name may be variable in length. Fig.2 gives a geometrical interpretation.

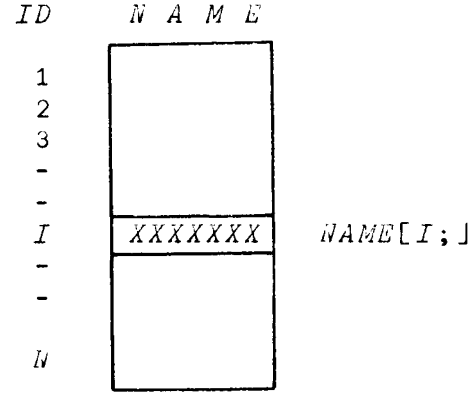


FIG. 2

A function ID permits us to retrieve the id corresponding to a given name. By definition

```
ID NAME[ i; ]  = i   is true
```

Definition of an Eclass

An Eclass with id=i can be defined by using

```
ECLASS [i] ← 1
```

An Eclass must also be named by a statement

```
NAME  [i;] ←'eclassname'
```

One associates with the Eclass a logical vector which is the characteristic of the Eclass and initializes it by

```
eclassname ←  N ρ 0
```

Definition of attributes

An attribute is a variable which may have a unique value for a given entity. Note that 'variable' is used in the API sense and may be an array.

The characteristic of the set of attributes is defined and initialized by

```
ATTR ← N ρ 0
```

An attribute is referred to by an id. An attribute with id=j can be defined by

```
ATTR [j] ← 1
```

An attribute must be named by a statement

```
NAME  [j;] ←  'attrname'
```

A variable must be defined with this name and appropriate shape and type by

```
attrname ← (N,x) ρ y
```

where x specifies the shape wanted for the attribute of a single entity and y the default value of the attribute. Fig.3 represents an attribute geometrically.

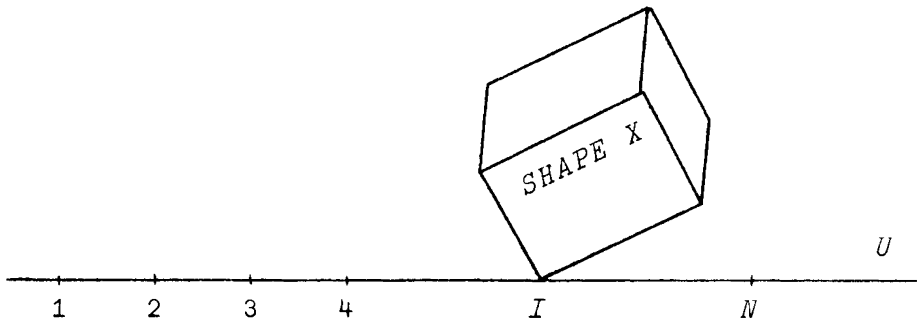


FIG. 3

If the attribute of a single entity is a scalar, the statement becomes

attrname ← N ρ y

Link_between_attribute_and_Eclass

There exists a system relation ATTRIBUTE with the corresponding array form

ATTRIBUTE [i;j]

where i is the id of an Eclass and j the id of an attribute and

ATTRIBUTE [i;j]

has a value 1 if and only if an attribute j may be applied to entities in Eclass i.

Note that NAME may be considered as a system attribute valid for all Eclasses.

Assignment_of_an_entity_to_an_Eclass

An entity with id=k is defined and assigned to the Eclass 'eclassname' by

eclassname [k] ← 1

The form of statements assigning values to attributes of an entity k, depends on the shape or portion of the value which must be assigned.

For example, if the value is a scalar,

attrname [k]← scalar

If it is a vector,

```
attrname[ k; ]← vector
```

If only one element of the vector has to be changed,

```
attrname[ k;h;l ] ← scalar
```

Relation_and_Relation_class (Rclass)

We have seen how an n-ary relation R is represented by a n-dimensional array. The relation R itself is an entity and attributes may be defined and values assigned to them. Thus a complete parallelism exists between ECLASS, entity class, entity and the following definitions.

Relation may be grouped into an Rclass. The characteristic of the set of Rclasses is defined and initialized by

```
RCLASS ← N ρ 0
```

The definition of an Rclass with id=r is done by

```
RCLASS [ r ]← 1
```

and

```
NAME [ r; ] ← 'relclass'
```

Attributes may be defined and assigned exactly as for Eclasses.

A relation with id=m, in Rclass 'relclass' is defined by

```
relclass[ m ]← 1
```

and

```
NAME[ m;] ← 'rname'
```

and the number of domains, n, on which the relation is defined, is specified by

```
rname ← (n ρ N) ρ 0
```

This also initializes the relation as empty.

Basic operations on relations include

```
n-tuple insertion in R     R[ i;j;k...] ← 1
       deletion in R       R[ i;j;k... ]← 0
       existence test      R [i;j;k... ]
```

System attributes for relation

System attributes may be applied to relations to specify some physical or logical properties. For example

```
sysatt1[ m ]← 1
```

would indicate that this particular property holds for relation m. The choice of these attributes is implementation dependent.

Domain definition

A higher level language should be able to identify the domain of a relation. This information must be included in the data model.

A domain may be an Eclass or the union of several Eclasses. A system relation is defined, with an array form

```
DOMAIN [i;j;p ]
```

where i is the id of a relation, j the id of an Eclass and p an integer and

```
DOMAIN [i;j;p]
```

has a value 1 if and only if the pth domain of relation i is Eclass j or includes j as a subset.

It is initialized by

```
DOMAIN ← [(N,N,N) ρ 0
```

Role definition

A higher level language must also be able to refer to a particular domain by name. We already mentioned the necessity of introducing a role name when the domains are not unique. This is also true for a domain which is not defined by a single Eclass. The information concerning the roles must also appear in the model.

The characteristic of the set of roles is defined and initialized as

```
RATTR ← N ρ0
```

A role is referred to by an id. A role with id=j can be defined by

```
RATTR [j] ← 1

NAME [j;] ←'rolename'
```

There exists a system relation

ROLE [i;j;p]

where i is the id of a relation, j the id of an Eclass or an attribute, p is an integer and

ECLE [i;j;p]

has a value 1 if and only if the pth domain of relation i may be referred to by role j.

It is initialized by

ECLE ← (N,N,N) ρ 0

MORE COMPLEX OPERATIONS

The reader who is familiar with APL will see immediately how the power of expression of the language may be used in conjunction with the proposed data model. But it is worthwhile to point out some basic functions.

- If an Eclass is called 'eclassname' a vector containing the id's of all entities in the Eclass is

eclassname/U

as this expression selects from U the id's for which eclassname [id] has a value 1.

- Set operations are performed on the characteristics. If C1, C2 are the characteristics of two sets, C1 ∨ C2 is the characteristic of the set obtained by taking the union of the two sets and C1 ∧ C2 the characteristic of the set obtained by intersecting the two sets.

- In a relation R [i;j;k...] the characteristic of the set of all j's satisfying R [i0;j;k0...] = 1 is

R [i0;;k0...]

- A very common operation used in relation processing is the projection which eliminates some domains of a relation. We refer to (2) for analysis of this operation and the following ones. Elimination of domain i may be written as

∨/ [i] R

(compression along the ith dimension, using the OR operation).

- A permutation of domains in a relation is equivalent to the APL transpose operator.

- Another important operation on relations is called

join. We consider the join between two binary relations (extension to higher order relation is immediate). If R[i;j] and S [k;l] are two relations joinable on the identical domains of j and k, then the AND outer product

Z ← R ∘.∧ S

yields a 4-dimensional array Z in which Z [i;j;k;l] = 1 if R [i;j] = 1 and S [k;l] = 1 ; and the transpose operation

1 2 2 3 ⌽ Z

eliminates the common domain. The join may thus be written as

1 2 2 3 ⌽ R ∘.∧ S

Conclusion

This paper has shown how APL may operate on a relational data-base. Any implementation based on this model needs to define the physical characteristics of the data, that is, the way they are actually stored on auxiliary storage, and then code the algorithms corresponding to the APL operators. This would enable implementations to be compatible with existing data-bases. Only a subset of the operators may be implemented if one wishes to specialize the system for pure data retrieval.

It is obvious that an implementation could not use the conventional physical representation of arrays as in the APL System (10), Fortran or PL/I. As arrays representing classes, attributes and relations are sparsely populated, one wants to adopt a more compact representation. Because auxiliary storage is used, a segmentation of information must exist and be based on logical criteria. Report (4) describes a specific implementation of a relational data-base, which is based on these principles. Such an approach may be used to organize physically the information contained in sparse arrays.

Acknowledgments

The author is greatly indebted to Dr. A.J. Symonds for many discussions and most helpful suggestions on the manuscript.

APPENDIX

EXAMPLE

An extensive example is presented. The context of the example has been mentioned before. One defines sets of entities: projects, parts, suppliers. Parts may have an attribute (quantity on hand). A relation is defined

R (part,project,supplier)

(R is used instead of Supply for conciseness).

The example has been tested using a simulation on a standard APL system with a very small universe.

The first ten lines are used for simulation. They initialize the system values and relations.

```
      ∇DEMO[□]
    ∇ DEMO                          DIMENSION OF THE UNIVERSE
[1]   N←25                                N=25
[2]   U←⍳N                          UNIVERSE VECTOR
[3]   NAME←(N,4)ρ'X'                NAME ARRAY
[4]   ECLASS←Nρ0                    INITIALIZE CHARACTERISTIC FOR ECLASS,
[5]   ATTR←RCLASS←RATTR←ECLASS            FOR ATTR, RCLASS ...
[6]   ATTRIBUTE←(N,N)ρ0             ATTRIBUTE(ECLASS,ATTR)
[7]   DOMAIN←(N,N,N)ρ0              DOMAIN(RELATION,ECLASS,POSITION)
[8]   ROLE←DOMAIN                   ROLE(RELATION,ATTR OR ECLASS,POSITION)
[9]   ⍝                             ---------------------------------------
[10]  ECLASS[1 2 3 4]←1             DEFINE FOUR ECLASSES
[11]  NAME[1;]←'PART'                     WITH THEIR NAMES
[12]  NAME[2;]←'PROJ'
[13]  NAME[3;]←'SUPP'
[14]  NAME[4;]←'ASSB'
[15]  PART←PROJ←SUPP←ASSB←Nρ0       CHARACTERISTICS OF ECLASSES
[16]  ⍝
[17]  PART[7 8 9 10 20 21]←1        DEFINE SOME PARTS
[18]  PROJ[11 12 13 14]←1                 PROJECTS
[19]  SUPP[15 16 17 22 23]←1              SUPPLIERS
[20]  ⍝
[21]  ATTR[5]←1                     DEFINE 'QUANTITY ON HAND' AS ATTRIBUTE
[22]  NAME[5;]←'QUOH'
[23]  QUOH←Nρ0
[24]  ⍝
[25]  RCLASS[6]←1                   DEFINE AN RCLASS
[26]  NAME[6;]←'RCL1'
[27]  RCL1←Nρ0                      DEFINE A RELATION
[28]  RCL1[18]←1
[29]  NAME[18;]←'R'
[30]  R←(3ρN)ρ0                     R IS A 3-ARY RELATION
[31]  ⍝
[32]  RATTR[19]←1                   DEFINE A ROLE NAME
[33]  NAME[19;]←'COMP'
[34]  DOMAIN[18; 1 4 ;1]←1          FIRST DOMAIN IS UNION OF ECLASSES 1,4
[35]  DOMAIN[18;2;2]←1              SECOND DOMAIN
[36]  DOMAIN[18;3;3]←1              THIRD DOMAIN
[37]  ROLE[18;19;1]←1               FIRST ROLE NAME IS 'COMP'
[38]  ROLE[18;2;2]←1                SECOND ROLE NAME IS ECLASS NAME
[39]  ROLE[18;3;3]←1                THIRD ROLE NAME IS ECLASS NAME
[40]  ATTRIBUTE[1;5]←1              QUOH IS ATTRIBUTE OF PARTS
[41]  ⍝
    ∇
[42]  ∇
```

```
      NAME[ATTRIBUTE[;5]/U;]        FOR WHICH ECLASS IS ATTRIBUTE 5 VALID?
PART
      NAME[ECLASS/U;]               FIND NAMES OF ALL ECLASSES
PART
PROJ
SUPP
ASSB
      NAME[ATTR/U;]                 FIND NAMES OF ALL ATTRIBUTES
QUOH
      PROJ/U                        FIND ALL PROJECTS
11  12  13  14
      NAME[DOMAIN[18;;1]/U;]        FIND ECLASSES INVOLVED IN THE DEFINITION
PART                                      OF FIRST DOMAIN OF RELATION 18
ASSB
      NAME[(∨/ROLE)[18;]/U;]        FIND THE ROLE NAMES
PROJ
SUPP
COMP
```

* COMMENT

THE ABOVE QUESTIONS SHOW HOW THE DESCRIPTION OF THE DATA-BASE CAN BE INTERROGATED WITH THE SAME LANGUAGE.

```
      DEMO2
      ∇DEMO2[□]
    ∇ DEMO2
[1]   R[7;11;17]←1                    MAKE ENTRIES IN RELATION R
[2]   R[7;11;23]←1
[3]   R[7;12;16]←1
[4]   R[7;13;16]←1
[5]   R[7;14;23]←1
[6]   R[8;13;15]←1
[7]   R[9;13;22]←1
[8]   R[10;13;16]←1
[9]   R[20;13;17]←1
[10]  R[9;12;22]←1
[11]  R[10;12;17]←1
[12]  R[10;14;17]←1
[13]  R[10;14;22]←1
[14]  R[10;11;22]←1
[15]  R[10;11;17]←1
[16]  R[10;13;17]←1
[17]  R[10;13;16]←1
    ∇
[18]  ∇      .


      R[7;11;17]                       EXISTENCE TEST
1                                            ANSWER IS YES
      R[7;11;17]←0                     DELETE ENTRY IN R
      R[7;11;17]                       EXISTENCE TEST
0                                            ANSWER IS NO
      R[7;11;17]←1                     MAKE ENTRY
      R[7;11;]/U                       WHO SUPPLIES PART 7 TO PROJECT 11
17  23
      R[;13;16]/U                      WHAT PARTS ARE SUPPLIED BY 16 TO
7  10                                         PROJECT 13


      (∨/∨/R)/U                        FIND ALL PARTS BEING SUPPLIED
7  8  9  10  20
      ρ(∨≠∨/R)/U                       NUMBER OF PROJECTS WHICH ARE SUPPLIED
4
      (∨≠∨/R)/U                        FIND THESE PROJECTS
11  12  13  14
      QUOH[7]←12                       SET VALUES FOR QUOH
      QUOH[8]←26
      QUOH[10]←99
      (A≥25)/A←QUOH[PART/U]            FIND VALUES OF QUOH WHICH ARE ≥25
26  99
```

```
      Y←(∨/[2]R)[7;]
      (SUPP∧~Y)/U
15  22
```

CHARACTERISTIC OF SUPPLIERS WHO SUPPLY PART 7 AND ALL SUPPLIERS WHO DO NOT SUPPLY PART 7

```
      (∧⌿PROJ⌿∨⌿R)/U
17  22
```

FIND ALL SUPPLIERS WHO SUPPLY TO ALL PROJECTS

```
      (∨⌿∧/[2]PROJ/[2]R)/U
17
```

FIND ALL SUPPLIERS WHO SUPPLY THE SAME PART TO ALL PROJECTS

```
      R[;13;16]/U
7  10
```

FIND ALL PARTS SUPPLIED BY 16 *TO PROJECT* 13

```
      Z←∨⌿R
      (∧⌿Z[;15]⌿Z)/U
15  16  17  22
```

FIND ALL SUPPLIERS WHO SUPPLY AT LEAST THOSE PROJECTS WHICH ARE SUPPLIED BY 15

REFERENCES

(1) Berry, P.C., APL/360 Primer, IBM Corporation (1968).

(2) Codd, E.F., A relational model of data for large shared data banks, Communications of the ACM, Vol. 13, No 6 (June 1970), 377-387.

(3) Codd, E.F., Notes on a data sublanguage, personal communication (January 1970).

(4) Crick, M.F.C., Lorie R.A., Mosher, E.J., Symonds, A.J., A data-base system for interactive applications, Cambridge Scientific Center, IBM Corporation, Report No G320-2058 (July 1970).

(5) Falkoff, A.D., Iverson K.E., APL/360: User's Manual, IBM Corporation (1968).

(6) Goldstein, R.C., Strnad, A.J., The MacAIMS management system, Proceedings ACM - SIGFIDET Workshop (November 1970), 201-229.

(7) Iverson, K.E., A Programming Language, Wiley (1962).

(8) Lathwell, R.H., APL/360: Operator's Manual, IBM Corporation (1968).

(9) Lathwell, R.H., APL/360: System Generation and Library Maintenance, IBM Corporation (1968).

(10) Lorie, R.A., Symonds, A.J., A schema for describing a relational data-base, Proceedings ACM - SIGFIDET Workshop (November 1970), 230-295.

(11) Pakin, S., APL/360 Reference Manual, Science Research Associates (1968).

(12) Strnad, A.J., The relational approach to the management of data-base, Project MAC, Massachusetts Institute of Technology, Cambridge (November 1970).

The following is a ccmplete list of Cambridge Scientific Center Technical Reports. Reports that are not available through an outside journal (as noted below) can be requested from IBM Corporation, Cambridge Scientific Center, 545 Technology Square, Cambridge, Massachusetts 02139.

320-2000 Production Sequencing by Combinatorial Programming
J. F. Pierce and D. J. Hatfield
March, 1966 ; Revised November, 1967
Appeared in TAPPI Special Technical Association Publication No. 4: Operations Research and the Design of Management Information Systems, chapter 17, Technical Association of the Pulp and Paper Industry, New York, 1967
Obsolete 10/30/72

320-2001 On the Solution of Integer Cutting Stock Problems by Combinatorial Programming - Part 1
J. F. Pierce
May, 1966; Revised June, 1968

320-2002 Application of Combinatorial Programming to a Class of All-Zero-One Integer Programming Problems
J. F. Pierce
July, 1966

320-2003 A Multi-Item Version of the Economic Lot Size Model J. F. Pierce
Appeared in IBM Systems Journal, Volume 7, No. 1, 1968

320-2004 String Processing on the System/360: Techniques and Example S. E. Madnick
Appeared in Communications of the ACM, Vol. 10, No. 7, July 1967

320-2005 SPL/I: A String Processing Language
S. E. Madnick
June, 1966

320-2006 An Approach to the Two Dimensional, Irregular Cutting Stock Problem
R. C. Art, Jr.
September, 1966
Appeared in Journal of Apparel Research Foundation, Volume 2, No. 2

320-2007 A Virtual Machine System for the 360/40
R. J. Adair, R. U. Bayles, L. W. Comeau, R. J. Creasy
May, 1966

320-2008 Applications of Time-Shared Computers in a Statistics Curriculum
M. Schatzoff
Appeared in Journal of the American Statistical Association, March 1968 Volume 63, pp 192-208

320-2009 Efficient Calculation of All Possible Regressions S. Fienberg, M. Schatzoff, R. Tsao
January, 1967; Revised August, 1967
Appeared in Technometrics, Volume 10, No. 4, November 1968

320-2010 An Experimental Comparison of Time-Sharing and Batch-Processing
M. Schatzoff, R. Tsao, R. Wiig
Appeared in Communications of the ACM, Vol. 10, No. 5, May 1967

320-2011 Computer Diagnosis: A Review and Discussion
E. Hoffer
May, 1967

320-2012 URBAN5: An On-Line Urban Design Partner
N. Negroponte, L. Groisser
Appeared in "The Architecture Machine," by N. Negroponte, MIT Press, MIT, Cambridge, Massachusetts
Obsolete 10/30/72

320-2013 A Study of the Effect of User Program Optimization in a Paging System
L. W. Comeau
Appeared in ACM Symposium on Operating System Principles 10/1/-4/67, Gatlinburg, Tennessee
Obsolete 10/30/72

320-2014 A Second Order Exponential Model for Multidimensional Dichotomous Contingency Tables, with Applications in Medical Diagnosis
R. F. Tsao
August, 1967

320-2015 CP-67/CMS User's Guide
Available from IBM Mechanicsburg, Form Number GH20-0859

320-2016 CCMB User's Guide
M. Schatzoff
Obsolete

320-2017 Design and Implementation of Cosmos
M. Schatzoff, R. Tsao, T. Burhoe
Obsolete

320-2018 On the Truck Dispatching Problem - Part 1
J. F. Pierce
Appeared in Transportation Research, Vo. 3, April 1969, pp. 1-42
Obsolete 10/30/72

320-2019 Computational Probability
U. Grenander, R. Tsao
March, 1968

320-2020 A Conversational Partition Monitor for OS/360 MFT
C. I. Johnson, R. M. Mitchell
Superseded by 320-2036

320-2022 Linguistic Tendencies in Pattern Analysis
U. Grenander
April, 1968

320-2023 SCRIPT: An Online Manuscript Processing System
S. E. Madnick, A. Moulton
Appeared in IEEE Transactions on Engineering Writing Systems, 8/68
IEEE, 345 East 47th Street, New York, New York 10017

320-2024 Mass Storage Software Simulated Associative Memory for PL/I Graphics
A. J. Symonds
Appeared in IBM Systems Journal, Vol. 7, No. 3 & 4, pp. 229-245, 1968

320-2026 A Feature Logic for Clusters
U. Grenander
June, 1968

320-2027 Multi-Processor Software Lockout
S. E. Madnick
April, 1968

320-2028 The Scattering of Sound by a Gas Bubble in an Elastic, Viscous Medium
J. W. Horton
July, 1968

320-2029 A Formulation of the Carotid-Artery Baroceptor Transducer Problem
J. W. Horton
June, 1968

320-2031 Pipe Network Analysis in Integrated Civil Engineering Systems (ICES)
K.T.H. Liu
Contact MIT, Mr. Kaalstad, 1-290, Cambridge, Massachusetts
Obsolete 10/30/72

320-2032 An Introduction to CP-67/CMS
L. H. Seawright, J. A. Kelch
Superseded by CP-67/CMS Description Manual, Available from IBM Mechanicsburg, Form No. GH20-0802

320-2033 Multi Access Systems - The Virtual Machine Approach
M. S. Field
September, 1968

320-2034 Computational Probability 3: Simulation
J. Grenander
December, 1968

320-2035 Operation of OS/360 in a Virtual Machine
C. I. Johnson, E. C. Hendricks
March, 1969

320-2036 Introduction to Online/OS
E. C. Hendricks, C. I. Johnson, R. D. Seawright, D. B. Tuttle
March, 1969

320-2037 Online/OS User's Guide
E. C. Hendricks, C. I. Johnson, R. D. Seawright, D. B. Tuttle
March 1969

320-2038 Design Strategies for File Systems: A Working Model
S. E. Madnick
Obsolete
Updated version can be ordered from: National Technical Information Service (NTIS), Operations Division, Springfield, Virginia 22151 (Microfilm, $0.95; Positive Copy, $3.00) REFER TO AD-714-269.

320-2039 Computational Probability 2: Randomness
U. Grenander
December, 1968

320-2040 Computational Probability 4: Regression in Time Series
U. Grenander, R. Vitale
February, 1969

320-2041 A Conversational Context-Directed Editor
the CSC Staff
March, 1969

320-2042 Use of Relational Programming to Manipulate a Structure in a Software Associative Memory (SAM)
A. J. Symonds
April, 1969
OUT OF PRINT

320-2043 An Experimental Graphic Input and Table Interface for the IBM 2250 Display Models 1 & 4
T. E. Johnson, F. W. Giesin, C. I. Johnson
March, 1969

320-2044 Interactive Graphics in Data Processing: Principles of Interactive Systems
C. I. Johnson
Appeared in IBM Systems Journal, Volume 7, No. 3 & 4, pp. 147-173, 1968

320-2045 Parametric Relaxation Techniques for Decomposing Quadratic Programs
K. M. Chandy
October, 1969

320-2048 Optimal Paths in a Two Parameter Cost Network
P. E. O'Neil, D. Coppersmith
July, 1969

320-2049 Storing a Data Base Using a Software Associative Memory - A Feasibility Study
M. F. C. Crick, A. J. Symonds
August, 1969
Obsolete, April 1972

320-2051 On Quadratic Optimization in Distributed Parameter Systems
S. G. Greenberg
November, 1969

320-2052 Pointwise Regulation of Distributed Parameter Systems S. G. Greenberg
November, 1969

320-2053 SCRIPT User's Manual
E. H. Levey
Available from IBM Mechanicsburg, Form No. GH20-0860

320-2054 Improved Combinatorial Programming Algorithms for a Class of All-Zero-One Integer Programming Problems
J. F. Pierce, J. S. Lasky
June, 1970

320-2055 On the Computation of Eigenvectors of a Symmetric Matrix Y. Bard
February, 1970

G320-2056 Automatic Page Fitting of Program Modules: A Means of Improving the Performance of Paging Systems
D. J. Hatfield, J. A. Gerald
Appeared in IBM Systems Journal, Vol. 10, No. 3, pp. 168-192, 1971
Title: Program restructuring for virtual memory

G320-2057 Non-Orthogonal Main-Effect Designs for Asymmetrical Factorial Experiments
B. H. Margolin
May, 1970

G320-2058 A Data-Base System for Interactive Applications
M. F. C. Crick, R. A. Lorie, E. J. Mosher, A. J. Symonds
July, 1970

G320-2059 A Schema for Describing a Relational Data Base
R. A. Lorie, A. J. Symonds
July, 1970

G320-2060 A Software Associative Memory for Complex Data Structures
M. F. C. Crick, A. J. Symonds
August, 1970

G320-2061 CP-67 Measurement and Analysis I: Regression Studies Y. Bard, B. Margolin, T. Peterson, M. Schatzoff
June, 1970

G320-2062 CP-67 Measurement and Analysis II: Overhead and Throughput
Y. Bard
September, 1970

G320-2063 Solution of the Algebraic Matrix Riccati Equation for Single-Input Systems in Standard Controllable Form
Y. Bard, S. G. Greenberg
October, 1970

G320-2064 An Improved Method for Designing Optimal Linear Compensators
Y. Bard, S. G. Greenberg
November, 1970

G320-2065 Integrated Text Processing for Publishing and Information Retrieval
C. F. Goldfarb, E. J. Mosher, T. I. Peterson
February, 1971; Revised April, 1971

G320-2066 AIMS - Applied Information & Management Simulation A General Business Simulation in APL
P. Wahi
April, 1971

G320-2067 APL as a Language for Handling a Relational Data Base
R. Lorie
March, 1971

G320-2068 Preferred Virtual Machines for CP-67
R. Parmelee
Unpublished as of May 1972

G320-2069 A Comparison of Computational Methods for Solving the Algebraic Matrix Riccati Equation
Stuart G. Greenberg and Yonathan Bard

G320-2070 Task Queuing in Auxiliary Storage Devices with Rotational Position Sensing
Yonathan Bard
May, 1971

G320-2071 Use of a Relational Access Method under APL
R. Lorie, A. J. Symonds
April, 1971

G320-2072 CP-67 Measurement Method
R. Adair, Y. Bard
May, 1971

G320-2073 The Ecology Decision Game Introduction
T. I. Peterson
September, 1971

G320-2074 Programming of Professional Society Meetings with an Integrated Text Processing System
Theodore I. Peterson, Edward J. Mosher, Charles F. Goldfarb
September, 1971

G320-2075 Analysis of Algorithms for CP-67 Free Storage Management
B. Margolin, R. Parmelee, M. Schatzoff
July, 1971

G320-2076 The Ecology Decision Game Management Science and Gaming
Pran N. Wahi and Theodore I. Peterson
October, 1971

G320-2077 The Ecology Decision Game - Game Authoring
Theodore I. Peterson and Rosemary Shields
October, 1971

G320-2078 An Eclectic Approach to Nonlinear Programming
Yonathan Bard
October, 1971

G320-2079 Interactive Statistical Data Analysis - APL Style
M. Schatzoff
April, 1972

G320-2080 Experimental Evaluation of System Performance
Y. Bard
April 1972

G320-2081 A Technique for Performance Comparison of System Software Features
Y. Bard
April 1972

G320-2082 A Note on an Implementation of an Experimental Preferred Virtual Machine System
R. P. Parmelee
June 1972

IBM

IBM Cambridge Scientific Center
545 Technology Square
Cambridge, Massachusetts 02139

IBM Houston Scientific Center
6900 Fannin Street
Houston, Texas 77025

IBM Los Angeles Scientific Center
1930 Century Park W.
Los Angeles, California 90067

IBM Palo Alto Scientific Center
2670 Hanover Street
Palo Alto, California 94304

IBM Philadelphia Scientific Center
3401 Market Street
Philadelphia, Pennsylvania 19104