NBSIR 81-2302

The Positional Set Processor: A Tool for Data Modeling

W. Terry Hardgrave
Sandra B. Salazar

National Bureau of Standards
Institute for Computer Sciences and Technology
Washington, D. C.

30 November 1981

Abstract

The Positional Set Processor (PSP) is a software tool for manipulating mathematical objects such as sets, sequences, ordered pairs, etc. The PSP serves as the underpinning for a Data Model Processor (DMP), an experimental system for emulating commercial and prototype database management systems. The PSP also provides a mathematical basis for semantic specification and interpretation of database operations. While its powerful query facilities make the PSP itself appear to be a database management system, it has no explicit concept of data definition, no access control, no integrity control, etc. The authors prefer to view the PSP as a tool for specifying and building database management systems. This paper reviews the mathematical formalism, Positional Set Notation, and describes the design of the Positional Set Processor.

1. INTRODUCTION

Data models may be viewed as the theoretical underpinnings for database management systems (DBMS). The Abstract Database Models Project at the National Bureau of Standards (NBS) is developing a structured approach to the study of data models. One objective of this project is to design and implement a Data Model Processor (DMP), a software system that will accept a description of a data model and emulate the behavior of an implementation of that data model (that is, a particular DBMS). While a precise definition of the term "data model" is still evolving, we believe that a specification for a data model should have these characteristics:

* It should include specifications for data structures (e.g. trees, networks, relations, etc.) plus specifications for operations (e.g. find-record, join, etc.) on those data structures.

* It should account for both structures used for definitional purposes and structures that appear as occurrences in the database. It may also include structures used for secondary indexing if the fundamental operations depend on those structures.

* It should be documented in the open literature, independently of any vendor's literature.

* It should be stipulated in a formal or mathematical framework that requires rigorous specification of semantics as well as syntax.

An important concept in the DMP framework (which is consistent with denotational semantics [JONE78]) is that the concepts of a data model can be expressed as set-theoretic objects. Thus, we have chosen to express the data structures as "positional sets" and the operations as operations on positional sets. This paper deals with the mathematical formalism for expressing positional sets, called Positional Set Notation (PSN), and a major component of the DMP, the Positional Set Processor (PSP), a sophisticated tool for storing and retrieving positional sets.

2. SUMMARY OF POSITIONAL SET NOTATION (PSN)

The philosophy behind Positional Set Notation is similar to the philosophy behind the Vienna Definition Method (VDM)--see [JONE78]. That is, the semantics of operations (i.e. functions, processes, etc.) programmed in computer languages should be defined in terms of abstractions that are divorced from physical implementations. These abstractions are, in turn, defined in a mathematical notation. The VDM approach uses (with a few minor exceptions) traditional mathematical notation and list structures. In this work, we use PSN.

PSN provides sufficient expressive power to define a wide range of data models. One salient feature is the availability of the "labeled tuple" which is the mathematical counterpart of an ADP "record" and simplifies the definition of most data models. This section provides an informal summary of PSN; more detail may be found in [HARD81a].

The essence of PSN is the recursive definition of the positional set, S:

S := [x(1)@p(1), x(2)@p(2), ..., x(n)@p(n)]

where

* the p(i) are called position identifiers and must be atoms, that is, either numbers or character strings. The p(i)'s need not be unique.

* the x(i) are elements and may be either atoms or themselves positional sets.

The mathematical object S, above, is a mathematical set of ordered pairs. We surround it with square brackets rather than braces for reasons discussed below. For convenience, the construction "x(i)@p(i)" is called a "duplex". Duplexes may appear in any order in the written representation of a positional set.

An example of a positional set (abbreviated as "p-set") in which all of the elements are atoms is:

[JONES@NAME,25@AGE,UMD@SCHOOL,BA@DEGREE]

Here the elements are JONES, 25, UMD, and BA. The position identifiers (pids) are NAME, AGE, SCHOOL, and DEGREE. An example of a p-set in which some of the elements are themselves p-sets is the p-set "STUDENT":

```
[[[[J@MI,JONES@LAST,JOHN@FIRST]@#]@NAME,25@AGE,
  [[UMD@SCHOOL,BA@DEGREE,
    [[CMSC41Ø@COURSE,B@GRADE]@#,
     [CMSC42Ø@COURSE,A@GRADE]@#]@CURRIC]@#]@ED_HIST]@#,
[[S@MI,SMITH@LAST,SCOTT@FIRST]@NAME,29@AGE,
  [[UNC@SCHOOL,PHD@DEGREE,
    [[CMSC62Ø@COURSE,A@GRADE]@#,
     [CMSC63Ø@COURSE,A@GRADE]@#,
     [CMSC798@COURSE,A@GRADE]@#,
     [CMSC72Ø@COURSE,B@GRADE]@#]@CURRIC]@#]@ED_HIST]@#]
```

The symbol # denotes the null position identifier. Given in tabular form with the position identifiers as column headers, the p-set above looks like:

STUDENT

NAME			AGE	ED_HIST			
FIRST	MI	LAST		SCHOOL	DEGREE	CURRIC	
						COURSE	GRADE
JOHN	J	JONES	25	UMD	BA	CMSC41Ø	B
						CMSC42Ø	A
SCOTT	S	SMITH	29	UNC	PHD	CMSC62Ø	A
						CMSC63Ø	A
						CMSC798	A
						CMSC72Ø	B

The PSP described here has print operations that display p-sets in tabular form as well as the formal bracketed notation; both forms are shown above.

PSN allows the data modeler to use and manipulate a number of objects similar to traditional mathematical objects. The p-sets representing the classical set, the ordered pair, the sequence, and the labeled tuple are shown in the table below. As mentioned above, the square brackets are used to surround p-sets so the braces can be used to surround classical sets (i.e., sets in which all the pids are "#"). The PSP will accept mathematical objects in either the traditional notation or PSN. All traditional expressions are converted to PSN. The mathematical underpinning is explained in detail in [HARD81a].

OBJECT	TRADITIONAL REPRESENTATION	POSITIONAL SET NOTATION
CLASSICAL SET	{a(1),a(2),...,a(n)}	[a(1)@#,a(2)@#,...,a(n)@#]
SEQUENCE	<a(1),a(2),...,a(n)>	[a(1)@1,a(2)@2,...,a(n)@n]
ORDERED PAIR	(a,b)	[a@1,b@2]
2-ELEMENT SEQUENCE	<a,b>	[a@1,b@2]

The labeled tuple is an important concept found in most data models and the basis for the relational model:

C(1)	C(2)	...	C(n)
a(1)	a(2)	...	a(n)

The C(i) are column-headings and a(j) are atomic elements; this is represented in PSN as:

[a(1)@C(1),a(2)@C(2),...,a(n)@C(n)]

Following is an example of a relational database expressed first in the tabular form of [ZLOO75], then in PSN using the relational formulation given in [HARD78].

Tabular form:

SALES	DEPARTMENT	ITEM
	STATIONERY	DISH
	HOUSEHOLD	PEN
	STATIONERY	PENCIL
	COSMETICS	LIPSTICK
	TOY	PEN
	TOY	PENCIL
	TOY	INK
	COSMETICS	PERFUME
	STATIONERY	PEN
	HARDWARE	INK

TYPE	ITEM	COLOR	SIZE
	DISH	WHITE	M
	LIPSTICK	RED	L
	PERFUME	WHITE	L
	PEN	GREEN	S
	PENCIL	BLUE	M
	INK	GREEN	L
	INK	BLUE	S
	PENCIL	RED	L
	PENCIL	BLUE	L

In PSN form, the set of relation occurrences, REL-OCC, is:

```
  [[SALES@REL_NAME,[[STATIONERY@DEPARTMENT,DISH@ITEM]@#,
                    [HOUSEHOLD@DEPARTMENT,PEN@ITEM]@#,
                                    .
                                    .
                                    .
                    [HARDWARE@DEPARTMENT,INK@ITEM]@#]@RELATION]@#,

   [TYPE@REL_NAME,[[DISH@ITEM,WHITE@COLOR,M@SIZE]@#,
                   [LIPSTICK@ITEM,RED@COLOR,L@SIZE]@#,
                                    .
                                    .
                                    .
                   [PENCIL@ITEM,BLUE@COLOR,L@SIZE]@#]@RELATION]@#]
```

3. AN OVERVIEW OF THE POSITIONAL SET PROCESSOR

The Positional Set Processor (PSP) is a collection of 37 commands (see Table 1). These commands are operations on positional sets; most commands take p-sets as input and produce p-sets as output. They are not intended to comprise a convenient high level query language.

The development of the PSP is part of an experimental project and the collection of commands has been modified as the Data Model Processor (DMP) design has proceeded. We began with a basic set of Classical Set Operations (such as union, intersection) that, when applied to positional sets, are analogous to the traditional set operations. The Utility Operations (such as print, copy) provide additional capabilities needed for use in an interactive environment.

The Positional Set Operations provide the user with the basic database functions--retrieving, updating, adding, and deleting. The operation "CREATE" is the most important operation and is analogous to the relational DBMS operations "Retrieve" in QUEL or "SELECT" in SQL. There are also operations to return the (classical) set of elements, return the (classical) set of position_identifiers for a p-set, and distribute a position_identifier over a classical set. TEMPLATE, POPULATE, and CONFORM were designed specifically for use by the various DMP roles.

Since some data models (e.g., CODASYL) require the manipulation of sequences, the Sequence Operations were added to allow manipulation of sequences as a special form of p-sets.

We will not attempt to fully describe all the PSP commands in this paper, but will concentrate on the central group, the Positional Set Operations. A complete description of the commands (including both syntax and semantics) can be found in "The Positional Set Processor User's Manual" [HARD81b] and examples of the use of the PSP by the DMP are given in [KOLL81] and [KOLL82].

TABLE 1 - PSP COMMANDS

* Classical Set Operations	
IN	Intersection
UN	Union
RC	Relative Complement
SY	Symmetric Difference
CD	Cardinality
* Positional Set Operations	
CR	Create
RG	Range
FR	Freeze
ST	Start
RL	Release
TH	Thaw
* Extended Positional Set Operations	
BR	Break
RK	Recurse
TMP	Template
POP	Populate
CNF	Conform
* Sequence Operations	
SQ_CAT	Concatenate sequences
SQ_EXC	Extract by contents
SQ_EXI	Extract by index
SQ_FLC	Flush by contents
SQ_IAC	Insert after contents
SQ_IAI	Index after index
SQ_IBC	Insert before contents
SQ_RPC	Replace element
* Utility Operations - Non-predicates	
CP	Copy
DE	Delete
NS	Null set
PT	Put
EN	Enter
PN	Print names
PS	Print set
* Utility Operations - Predicates	
CS	Classical set
EQ	Equal
ISIN	Is in
NL	Null
NN	Not null
SB	Subset

4. POSITIONAL SET OPERATIONS

The Positional Set Operations (CREATE, RANGE, FREEZE, START, RELEASE, and THAW) allow traversal of p-sets and generation of new p-sets based on a specified condition. In this section we will discuss "range variables" and the "dot operator" and will then describe the commands.

Range Variables

Range variables (rv) allow access to p-sets. The name "range variable" is taken from ALPHA [CODD71] and QUEL [STON76], but the concept is more general here. Unlike (normalized) relations, p-sets may be nested to any depth. The range variables provide a mechanism for accessing nested structures as well as first normal form relations.

Each rv is associated with a particular p-set. That is, the declaration of a range variable enables that variable to take on, one-at-a-time and in no particular order, the values of the elements of a p-set. If X is a variable ranging over the p-set P, then "print all X" would print all the elements of P. The qualification, "X = <value>" is TRUE iff $\exists$X (X $\in$ P AND X = <value>). Thus, "print all X where <condition>" would print those elements of P which satisfy <condition>.

The association between the p-set and the rv is established with the RANGE command and (unlike QUEL) remains in effect until cleared by the RL or ST command (see below).

Dot Operator

The dot operator, often used without formal definition, has been a mainstay for several relational languages (e.g. QUEL, SEQUEL2), as it is quite useful in designing query languages with substantial expressive power. It is also a notational device to simultaneously allow traversal of a set and access to (elements of) tuples within the set. Since both the tuples and the sets can be represented as p-sets, a more precise definition of the dot operator expression can be given in PSN.

Let X range over the p-set P. At any particular time, X is a pseudonym for a tuple T that is an element of P. The construction: "X.ATT" refers to the element at pid ATT contained in the tuple T. "Print all X.ATT" would print all the values, Y, such that Y@ATT is an element of an element of P. That is:

"X.ATT = <value>" is TRUE iff

$\exists Y \exists T$ (Y@ATT $\in$ T AND T $\in$ P AND Y = <value>).

For example, consider the <condition>:

X.AGE = '21'

as evaluated for a particular value of X (ie., some element of P). The <condition> is TRUE if "21@AGE" is an element of X (ie., an element of an element of P). Similarly, the <condition>:

X.AGE > '30'

returns TRUE if there exists an M such that "M@AGE" is an element of X and M > '30'.

Commands on Range Variables

The START (ST) command initializes the global range variable by deleting all range variable associations.

The RANGE (RG) command declares and associates a range variable with a p-set. It has two forms:

RG <rv> IS <p-set name>
or
RG <rv1> IS <rv2> . <pid>

For example, the command:

RG X IS STUDENT

declares "X" a range variable and associates it with the p-set named "STUDENT". The rv "X" ranges over the duplexes of the p-set "STUDENT".

The command form to declare and associate an rv with a p-set element (that is itself a p-set) is somewhat more complex. The dot operator "." indicates that the <pid> is at the next level of nesting within the p-set or p-set element associated with <rv2>. For example, the command:

RG Y IS X.ED_HIST

declares "Y" a range variable and associates it with the element X which has pid "ED_HIST". "ED_HIST" is at the next level of nesting within the p-set over which "X" ranges, that is, "STUDENT".

Declaring one range variable in terms of another implies a relationship between range variables that is important for the semantics of the CREATE command; this is an ancestor relationship. If X is an rv and appears in an RG statement:

```
RG X IS <p-set name>
```

then X has no ancestors. If Z is an rv and appears in a statement:

```
RG Z IS Y.<pid>
```

then Y is both an ancestor and the parent of Z. All of Y's ancestors are also ancestors of Z. In the example:

```
RG X IS STUDENT
RG Y IS X.ED_HIST
RG Z IS Y.CURRIC
```

"X" has no ancestors; "X" is an ancestor of "Y" and "Z" and the parent of "Y"; "Y" is an ancestor and the parent of "Z".

The RELEASE (RL) command, which has the form:

```
RL <rv>
```

removes the association between the specified rv and a p-set.

The FREEZE (FR) command binds the rv to one duplex of the p-set for which the specified <condition> is true. This means that the rv no longer ranges over the entire p-set, but indicates only one duplex. The form of the FREEZE command is:

```
FR <rv> WHERE <condition>
```

The rv must have been previously associated with a p-set using an RG command. The <condition> is a predicate--an expression involving comparison operators, boolean operators, and parentheses, that returns "true" or "false". For example, the command:

```
FR X WHERE "X.AGE = '29'"
```

indicates that the rv "X" ranges over the duplexes in the p-set with which it is associated. When a duplex that satisfies the condition (that the value of the element associated with the pid "AGE" is '29') is found, the value of the rv "X" is then frozen at (i.e. bound to) that duplex. If the condition is never satisfied, the rv remains free.

If an rv is frozen with the FREEZE command, all of its ancestors will be frozen also. For example, if the unfrozen rv's "X", "Y", and "Z" are associated with the p-sets "STUDENT", "X.ED_HIST", and "Y.CURRIC", respectively, the command:

```
FR Z WHERE "Z.COURSE='CMSC798'"
```

will not only freeze "Z" at a duplex in the p-set element "Y.CURRIC" which has 'CMSC798' as the value of the element associated with the pid "COURSE", but will also freeze "Y" and "X" at the duplexes which are their values when the condition on "Z" is satisfied.

Note that when more than one duplex satisfies the condition, the choice of a duplex is implementation-dependent and cannot be anticipated or controlled by the user. The capability to traverse the p-set, successively freezing on each duplex that satisfies the condition, is an option.

The THAW (TH) command, which removes the FREEZE binding, has the form:

```
TH <rv>
```

When an rv is thawed, any frozen descendants are also thawed; ancestors are not affected. Thus in general, the "TH X" command is not the reciprocal of the "FR X" command.

The CREATE Command

The PSP CREATE (CR) command is analogous to SELECT in the SEQUEL language of SYSTEM R [ASTR76] or RETRIEVE in the QUEL language of INGRES [STON76]. However, unlike SEQUEL and QUEL, the PSP allows nested relations. In relational jargon, this means that first normal form (1NF) is not required. The CREATE command builds a new p-set having (1) the specified attributes from the original p-set(s) and (2) the duplexes which satisfy a given condition. It has the form:

```
CR P WITH <attribute-list> WHERE <condition>
```

The <condition> has the same form and meaning for the CREATE command as for the FREEZE command. The <attribute-list> defines the structure of the new p-set by identifying its pids. Syntactically, it is a list of <term>'s and <assignment>'s. A <term> specifies which pids are to be taken from which p-sets; the basic form is:

```
"<rv>.<pid>"
```

The <assignment> has the form:

"<pid> := <phrase>"

This allows the user to specify a pid for the new p-set which is not related to a previously defined p-set and to assign a value for its element. The <phrase> may be an arithmetic expression, a value, or a basic <term>. To rename the pid in the new p-set, this form of the <assignment> is used:

"<pid> := <rv>.<pid>"

An example of the CR command using our sample database is:

```
RG X IS STUDENT
RG Y IS X.ED_HIST
RG Z IS Y.CURRIC
CR NEWSET WITH "X.NAME,X.AGE,CLASS := '1981',
      TRANSCRIPT := Y.CURRIC" WHERE "Y.DEGREE = 'PHD' |
      (Z.COURSE = 'CMSC498' & Z.GRADE = 'A')"
```

In the "WHERE" clause, "|" is the logical "OR" operator and "&" is the logical "AND" operator. The p-set "NEWSET" will have the pids "NAME", "AGE", "CLASS", and "TRANSCRIPT". "NAME" and"AGE" will be the same as in "STUDENT", and "TRANSCRIPT" will be the same as "CURRIC" in "STUDENT" (only the pid name is different). Each duplex will have the new pid "CLASS" which has the element value '1981'. The duplexes chosen from "STUDENT" are those which have 'PHD' as the value of "DEGREE" or that have a duplex in "CURRIC" with 'CMSC498' as the value of "COURSE" and 'A' as the value of "GRADE". The answer to this query, given the p-set "STUDENT" of section 2, would be:

PSN form:

```
[[[S@MI,SMITH@LAST,SCOTT@FIRST]@NAME,29@AGE,1981@CLASS,
  [[CMSC620@COURSE,A@GRADE]@#,
   [CMSC630@COURSE,A@GRADE]@#,
   [CMSC798@COURSE,A@GRADE]@#,
   [CMSC720@COURSE,B@GRADE]@#]@TRANSCRIPT]@#]
```

Tabular form:

NEWSET

NAME			AGE	CLASS	TRANSCRIPT	
LAST	FIRST	MI			COURSE	GRADE
SMITH	SCOTT	S	29	1981	CMSC62Ø	A
					CMSC63Ø	A
					CMSC798	A
					CMSC72Ø	B

Notice the similarity in the QUEL equivalent of this CREATE statement:

```
RETRIEVE INTO NEWSET
(AGE=X.AGE,NAME=X.NAME,TRANSCRIPT=Y.CURRIC,CLASS="1981")
WHERE Y.DEGREE="PHD" OR
      (Z.COURSE="CMSC498" AND Z.GRADE="A")
```

Since the PSP was designed to contain the functionality of the relational operators, it is appropriate to discuss how the relational operators (i.e. PROJECT, SELECT, and JOIN) as well as p-set operators can be expressed in terms of the CREATE command. An informal discussion and some examples are given here.

A PROJECT can be represented as a CR with no <condition>, with the effect that each duplex of the original p-set is selected, but only the pids specified in the attribute-list are copied. For example,

```
RG X IS STUDENT
CR AGES WITH "X.AGE"
```

creates the p-set "AGES" which is a classical set containing all the elements with pid "AGE" in the p-set "STUDENT".

PSN form:

```
[[25@AGE]@#,[29@AGE]@#]
```

Tabular form:

```
                AGES
--------------------------------
        AGE
---------------
        25
        29
```

A slightly more complicated example is a PROJECT involving a nested pid:

```
  RG X IS STUDENT
  RG Y IS X.ED_HIST
  CR SCHOOLS WITH "Y.SCHOOL"
```

This creates the p-set "SCHOOLS", a classical set containing all elements with the pid "SCHOOL" at the "ED_HIST" level of the p-set "STUDENT". Any duplicate duplexes are, of course, eliminated, since a p-set is a mathematical set.

PSN form:

```
  [[UMD@SCHOOL]@#,[UNC@SCHOOL]@#]
```

Tabular form:

```
                SCHOOLS
--------------------------------
        SCHOOL
---------------
        UMD
        UNC
```

A SELECT can be expressed as a CR which copies each duplex of a p-set that satisfies a condition. For example,

```
  RG X IS STUDENT
  RG Y IS X.ED_HIST
  CR DOCTORS WITH "X.NAME,X.AGE,X.ED_HIST" WHERE
      "Y.DEGREE='PHD'"
```

This creates a p-set "DOCTORS" having the same structure as the original p-set "STUDENT" but a subset of the original duplexes--those that satisfy the condition that 'PHD' is the value of the element with pid "DEGREE" at the "ED_HIST" level.

PSN form:

```
  [[[S@MI,SMITH@LAST,SCOTT@FIRST]@NAME,29@AGE,
    [[UNC@SCHOOL,PHD@DEGREE,
      [[CMSC620@COURSE,A@GRADE]@#,
       [CMSC630@COURSE,A@GRADE]@#,
       [CMSC798@COURSE,A@GRADE]@#,
       [CMSC720@COURSE,B@GRADE]@#]@CURRIC]@#]@ED_HIST]@#]
```

Tabular form:

DOCTORS

NAME			AGE	ED_HIST			
FIRST	MI	LAST		SCHOOL	DEGREE	CURRIC	
						COURSE	GRADE
SCOTT	S	SMITH	29	UNC	PHD	CMSC620	A
						CMSC630	A
						CMSC798	A
						CMSC720	B

A JOIN can be expressed as a CR which concatenates duplexes from two p-sets, based on a condition that links the p-sets through some pids that presumably have the same domain. For example, given the p-set "ADDRESS":

PSN form:

```
  [[[S@MI,SMITH@LAST,SCOTT@FIRST]@PERSON,RALEIGH@CITY,NC@STATE]@#,
   [[J@MI,JONES@LAST,JOHN@FIRST]@PERSON,PITTSBURGH@CITY,PA@STATE]@#]
```

Tabular form:

ADDRESS

PERSON			CITY	STATE
FIRST	MI	LAST		
SCOTT	S	SMITH	RALEIGH	NC
JOHN	J	JONES	PITTSBURGH	PA

The PSP commands:

```
RG X IS STUDENT
RG S IS ADDRESS
CR COMPLETE WITH "X.NAME,X.AGE,X.ED_HIST,S.CITY,S.STATE"
    WHERE "X.NAME=S.PERSON"
```

create the p-set "COMPLETE" which consists of all concatenations of a duplex from "STUDENT" and a duplex from "ADDRESS" containing the same value of the elements corresponding to pid "NAME" in "STUDENT" and pid "PERSON" in "ADDRESS". The answer to this query, given the example "STUDENT" database, is:

PSN form:

```
[[[J@MI,JONES@LAST,JOHN@FIRST]@NAME,25@AGE,
  [[UMD@SCHOOL,BA@DEGREE,
    [[CMSC410@COURSE,B@GRADE]@#,
     [CMSC420@COURSE,A@GRADE]@#]@CURRIC]@#]@ED_HIST,
  PITTSBURGH@CITY,PA@STATE]@#,
[[S@MI,SMITH@LAST,SCOTT@FIRST]@NAME,29@AGE,
  [[UNC@SCHOOL,PHD@DEGREE,
    [[CMSC620@COURSE,A@GRADE]@#,
     [CMSC630@COURSE,A@GRADE]@#,
     [CMSC798@COURSE,A@GRADE]@#,
     [CMSC720@COURSE,B@GRADE]@#]@CURRIC]@#]@ED_HIST,
  RALEIGH@CITY,NC@STATE]@#]
```

Tabular form:

COMPLETE

NAME			AGE	ED_HIST				CITY	STATE
FIRST	MI	LAST		SCHOOL	DEGREE	CURRIC			
						COURSE	GRADE		
JOHN	J	JONES	25	UMD	BA	CMSC410	B	PITTSBURGH	PA
						CMSC420	A		
SCOTT	S	SMITH	29	UNC	PHD	CMSC620	A	RALEIGH	NC
						CMSC630	A		
						CMSC798	A		
						CMSC720	B		

5. CONCLUDING REMARKS

The first version of the PSP is operational in an experimental environment. The system is quite slow and only works with very small databases; so far, little work has been done on performance improvement. This section discusses the success of the design and our plans for the future.

The PSP and Data Models

As mentioned earlier, the PSP is intended to be used as a tool for studying DBMS's, not as a DBMS itself. It has no data definition facility, nor any restrictions on database design, as would exist in a commercial DBMS that implements a particular data model. However, unlike most tools, the PSP has a tremendously powerful query facility. Consequently, it is useful to compare the expressive capabilities of the PSP to those of various data models.

The query language of the PSP may be viewed as an extension of relational query languages. Range variables, similar to those in QUEL, enable referencing of attributes in a relation. The basic data retrieval and manipulation command, CREATE, is analogous to SELECT in SYSTEM R. However, the PSP has some important features that extend its expressive power beyond that of current relational systems.

* It allows nesting of all kinds of mathematical objects within one another: sets, sequences, relations, or any other structures definable in PSN.

* There are no intrinsic normal forms. Normal forms are restrictions on the original mathematical concept of a relation. The proliferation of normal forms as anomalies or limitations are discovered in each new one suggests that normalization is not worth the sacrifice of the power of the mathematical definition of "relation".

* The database may contain nested hierarchical structures that can be referenced with relational-like commands.

* Nodes from different subtrees can be specified in the same query.

Thus, the PSP combines concepts from several data models and serves as a powerful underpinning for the Data Model Processor and as a basis for the study of future database management systems.

Future Directions

Project efforts in the next year are directed toward applications of the Data Model Processor to such areas as data model mappings, commercial DBMS validation, evaluation of data model specifications, and query language translation. A paper on the DMP [KOLL82] is forthcoming. Work will also be done on improving the efficiency and expressive power of the PSP, so that it will be a more effective and efficient tool. Some of our plans are discussed below.

Another version of the PSP, based on the same users manual, but using an integer set processor (ISP), may be implemented. This would be similar to a previous implementation described in [HARD76]. The ISP has the potential to perform set operations rapidly on rather large classical sets of positive integers represented by bit strings. A revised bit-string technique has been developed by Gallagher [GALL81]. By mapping p-sets onto the integers (using an element table and the Cauchy-Cantor technique), the power of an ISP can be used to implement p-sets.

Measurement and predictive performance modeling techniques may be used to direct the further development of the prototype PSP. A new predictive model that handles both recursive routines and multiple concurrent processes may be designed, implemented, and used to gather data on the execution of the major routines of the PSP. Based on the results, a model, similar to [DEUT78], that predicts performance for the prototype may be attempted. The model would be validated by checking predictions against actual performance in test cases, then the model would be used to forecast performance in an operational environment. Modification and optimization efforts would concentrate on the PSP routines that the model shows to be problem areas.

The expressive power of the PSP will be increased by modifications to some of the commands and operators. Additional features for creating and accessing nested structures will be specified.

We will be examining the relationship between the PSP and other work on semantics, abstract data modeling, and functional languages. We are trying to incorporate some ideas from functional languages [BACK80] but this is just beginning.

6. ACKNOWLEDGEMENTS

A major part of the design work on the PSP and DMP has been done by Drs. M. Koll and G. Sockut. The implementation (and some design) has been carried out by these part-time and student programmers: Edwin J. Beller, Charles Haas, Jerald Herstein, Thomasin Kirkendall, Anthony Marriott, Joseph Naputi, Steven Norman, Sabrina Saunders, Michael Shon, Cristine Shuey, and George Skillman.

7. REFERENCES

ASTR76 Astrahan, M. M., et al., "System R: Relational Approach to Database Management", ACM Transactions on Database Systems, Vol. 1, No. 2, June 1976, pp. 97-137.

BACK80 Backus, J., "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs", Communications of the ACM, Vol. 21, No. 8, August 1978, pp. 613-641.

CODD71 Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus", Proceedings of the 1971 ACM-SIGFIDET Workshop on Data Description, Access, and Control, Nov. 1971, ACM, New York, 1971, pp. 35-68.

DEUT78 Deutsch, D. R., Modeling and Measurement Techniques for Evaluation of Design Alternatives in the Implementation of Database Management Software, NBS Special Publication 500-49, 1978.

GALL81 Gallagher, L., "Computer Implementation of an Integer Set Algebra", NBS Special Publication (in preparation), 1981.

HARD76 Hardgrave, W. T., "A Technique for Implementing a Set Processor", Conference on Data: Abstraction, Definition, and Structure, ACM, N.Y., 1976, pp. 8-94.

HARD78 Hardgrave, W. T., "The Relational Model: A Reformulation of Some Mathematical Aspects Using Positional Set Notation", IFSM T.R. No. 25, University of Maryland, 1978.

HARD81a Hardgrave, W. T., "Positional Set Notation", to appear in Advances in Database Management, Vol. 2, Heyden and Son, New York, 1981.

HARD81b Hardgrave, W. T., et al., "The Positional Set Processor User's Manual", NBSIR (in preparation), 1981.

JONE78 Jones, C. B. and D. Bjorner (eds.), The Vienna Development Method: The Meta-Language, Springer-Verlag, New York, 1978.

KOLL81 Koll, Matthew, "Data Model Processor User's Manual", NBSIR (in preparation), National Bureau of Standards, Washington, D. C., 1981.

KOLL82 Koll, Matthew B., Hardgrave, W. Terry, and Sandra B. Salazar, "Data Model Processing", submitted to NCC, Houston, Texas, May, 1982.

KURA76 Kuratowski, K. and A. Mostowski, Set Theory, North-Holland Publishers, New York, 1976.

STON76 Stonebraker, M., E. Wong, et. al., "The Design and Implementation of INGRES", ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976, pp. 189-222.

ZLOO75 Zloof, M.M., "Query-By-Example", Proceedings of the AFIPS National Computer Conference Vol. 44, AFIPS Press, Montvale, N.J. (May 1975) pp. 431-438.

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBS-IR-81-2302	2. Performing Organ. Report No.	3. Publication Date Nov. 30, 1981

4. TITLE AND SUBTITLE

The Positional Set Processor: A Tool for Data Modeling

5. AUTHOR(S)

W. Terry Hardgrave, Sandra B. Salazar

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

Institute for Computer Sciences & Technology
Data Management & Programming Languages Division
National Bureau of Standards/DoC/Bldg. 225, Room A265
Washington, DC 20234

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

The Positional Set Processor (PSP) is a software tool for manipulating mathematical objects such as sets, sequences, ordered pairs, etc. The PSP serves as the underpinning for a Data Model Processor (DMP), an experimental system for emulating commercial and prototype database management systems. The PSP also provides a mathematical basis for semantic specification and interpretation of database operations. While its powerful query facilities make the PSP itself appear to be a database management system, it has no explicit concept of data definition, no access control, no integrity control, etc. The authors prefer to view the PSP as a tool for specifying and building database management systems. This paper reviews the mathematical formalism, Positional Set Notation, and describes the design of the Positional Set Processor.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

databases, software tool, DBMS, database management systems

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

22

15. Price

USCOMM-DC 6043-P80