A PRECISE DEFINITION OF BASIC RELATIONAL NOTIONS

AND OF THE RELATIONAL ALGEBRA

Alain Pirotte

Computer Corporation of America
575 Technology Square
Cambridge, Massachusetts 02139
USA

ABSTRACT

This paper presents a precise definition of basic relational notions as well as a precise and general definition of the relational algebra. The paper also illustrates a method for developing semi-formal definitions of data models. A nearly formal definition of relations with unordered attributes is given, and algebraic operations are described as operating on and producing relations thus defined. The definition of algebraic operations contains, as a special case, the "classical" definition of the algebra (see e.g. [CODD72]). But, in addition, several operations are generalized. The closure of the algebra is characterized precisely, with the help of an operation for renaming relation attributes.

This work was partially supported by the National Bureau of Standards under Contracts NB79SBCA0088 and NB79SBCA0086.

CONTENTS

1. INTRODUCTION

This paper presents a precise definition of basic relational notions and a precise and general definition of the relational algebra.

The definitions are precise, in that there should be no ambiguity in the details and the scope of what is described. Definitions are expressed with a semi-formal notation that provides enough detail to permit an easy construction of a formal definition. Thus, the description method does not give up anything essential in precision with respect to a completely formal definition (see e.g., [BJOR80, LAME80, RTG82]). We believe that the degree of formality of a definition is a relatively secondary issue, and that any formal definition is not automatically interesting simply because it is formal. For the approach used in this paper, it was thought that being completely formal would obscure things instead of clarifying them.

A nearly formal definition of relations with unordered attributes is given. In that definition, a relation is characterized by a name, a structure, and a value.

The definition of the relational algebra is general. The result of every algebraic operation is defined to be a relation with well defined and unordered attributes. The definition of algebraic operations contains, as a special case, a semi-formal equivalent of the "classical" definition of the algebra (see e.g., [CODD72]). In addition, several algebraic operations are given a more general definition. Thus, for example, natural join is defined so that several attributes of each operand relation can be tested for equality in a single join operation. In addition to the usual operation, a new version of division is defined that corresponds closely to typical patterns of universal quantification. New algebraic operations, called bordered union, intersection, and difference are introduced: they generalize the corresponding set-theoretic operations to relation operands that are not union-compatible. Algebraic operations that return a truth value are also defined. The definition of algebraic operations is not limited to specifying a minimum set. As a result, some operations can be defined in terms of combinations of others. Such equivalences, where they exist, are mentioned.

The algebra enjoys a "closure" property: algebraic expressions can be used as operands of algebraic operations to any depth of nesting. Accounting for that property with precision and generality requires a specification of the correspondence of attributes between the operand(s) and the result of an algebraic operation, and also between the operands of binary algebraic operations. This paper defines a renaming operation for attributes of a relation and requires that this operation be invoked when necessary in algebraic expressions in order to realize special correspondences of attributes before algebraic operations are applied. The relation result of an algebraic operation directly inherits attributes from the (possibly renamed) operands. As any relation in an algebraic expression can undergo attribute renaming, attribute correspondence is thus specified in full generality.

The structure of the paper is as follows. Section 2 introduces basic mathematical concepts necessary to construct a precise definition of relations and operations on relations. Section 3 gives a consistent definition of the basic relational concepts, and discusses some questions usually not specified precisely. Sections 4 and 5 define the algebraic operations. Most definitions of operations are presented in two parts. First, an informal definition in English briefly presents the general idea. It is followed by a precise definition in a semi-formal notation. Section 6 briefly presents concrete forms of algebraic languages, and Section 7 sketches how the algebra can be used to express constraints.

Currently, there does not exist a definition that would be recognized as a precise reference for THE relational model. This paper gives a nearly formal definition for the bases of the "usual" relational model: the fundamental structural concepts (relation, domain, attribute, database, etc.), and the relational algebra. It is not, however, a complete definition of the relational model. Additional concepts and operations can be described using the method employed in this paper, such as the tuple and domain relational calculi, update operations, functional dependencies and keys, multi-valued and other kinds of dependencies, etc. The difficult issue of precisely which data structures, operations, and general integrity rules belong to THE relational model is being addressed by standards committees [RTG82].

2. BASIC CONCEPTS

The language used in this paper to define the semantics of the relational model is a mathematical language based on set-theoretical operations and on the predicate calculus. For brevity, it is not defined explicitly; its notations should be sufficiently familiar. This section recalls some mathematical concepts that are useful in a precise definition of the relational model.

2.1 Set theory

- Set-theoretic operations
  - The union S1 ∪ S2 of two sets S1 and S2 comprises all the elements that are either in S1, or in S2, or in both S1 and S2.
  - The intersection S1 ∩ S2 of two sets S1 and S2 comprises the elements that are in both S1 and S2.
  - The difference S1 - S2 of two sets S1 and S2 comprises those elements of S1 which are not in S2.
  - The Cartesian product S1 x S2 of two sets S1 and S2 is the set of ordered pairs or 2-tuples (s1,s2), where s1 is an element of S1 and s2 is an element of S2.
  - The power set P(S) of a set S is the set of all subsets of S.
- Set comparators
  - Equality and inequality of two sets S1 and S2: S1 = S2, and S1 ≠ S2.
  - Inclusion. S1 is a subset of S2 (S1 ⊂ S2) if all the elements of S1 are also elements of S2.
  - Element of. s∈S if s belongs to S; otherwise s∉S.

- Cardinality of a set S. #S is the number of elements in S.

- Set-valued expressions. {x | P(x)}, (where P is a predicate of the predicate calculus with x as a free variable), is the set of values for which the predicate P is true.

2.2 First-Order Predicate Calculus

An applied, first-order predicate calculus is used in this report. We assume that the reader is familiar with this notation. Predicates in this calculus can be any expressions having a truth value, including set comparisons.

2.3 Functional Mapping

A functional mapping (or function) is defined as a set of ordered pairs {(a1,b1),...,(an,bn)} that does not contain two pairs having the same first element (we will also write {(a1->b1),..., (an->bn)}).

The domain of a function consists of all the first elements of the ordered pairs. The range of a function consists of all the second elements of the ordered pairs. More formally, if M is a function, then its domain (dom) and range (range) are defined as:

$$dom(M) = \{a \mid \nexists\, b \quad (a,b) \in M\}$$
$$range(M) = \{b \mid \nexists\, a \quad (a,b) \in M\}$$

If a is in dom(M), then

M(a) is the element b such that $(a,b) \in M$.

The union of two functions with disjoint domains is the ordinary set union of the associated sets of pairs.

2.4 Generalized or Indexed Cartesian Product

A generalized Cartesian product of a collection of sets can be defined as follows. Consider a collection of n sets, (D1,...,Dn) where the sets are not necessarily distinct, that is, where some of the sets in the collection may be the same. Consider next a set A of n distinct indexes. The collection of sets is "put into a correspondence with" the set of indexes by associating each index with one set in the collection. The set associated with index a will be called Da.

The indexed Cartesian product of n sets indexed by the set A of n distinct indexes is defined as:

$$\{f : A \to \bigcup_{a \in A} Da \mid f \text{ is total and } \forall a \in A : f(a) \in Da\}$$

that is, the set of all functions from A to the union of the sets Da (for $a \in A$), where each function is defined for each index in A and has as its value an element in the associated set.

The indexed Cartesian product will be denoted: Ai:Di x ... x An:Dn or, for brevity, X(A:D).

Note that if t is an element of the indexed Cartesian product (i.e., t is a particular function), then dom(t) = A, the set of indexes. The definition of dom is extended to:

$$dom(X(A:D)) = A$$

The elements of an indexed Cartesian product will also be called labeled n-tuples, that is, unordered n-tuples of values where each value is explicitly associated or labeled with an index.

The concept of the indexed Cartesian product is more general than that of the "ordinary" Cartesian product in two ways. First, it is defined for any number of argument sets, while the ordinary Cartesian product is defined for only two sets. Second, the argument sets are not ordered in the indexed product; instead, they are distinguished from one another by unique indexes.

3. DEFINITION OF BASIC RELATIONAL NOTIONS

This section gives a consistent definition of some basic relational concepts and goes into some details usually not discussed explicitly. A relation is defined as having a fixed name, a fixed structure, and a time-varying value. Terminology and the role of domains are discussed in detail.

3.1 Domains

Elementary values in relations belong to specified sets associated with each relational database. These sets are called domains. A database domain, or simply a domain, is a finite set of atomic elements, or values. Each domain has a name that uniquely identifies it among all the domains of a database. All domains of a relational database are disjoint. We do not consider domains that overlap one another but are not identical.

A domain has associated with it:

1. A collection of binary comparison operators. This collection includes, for each domain, at least an operation that tests whether two elements are equal and an operation that tests whether two elements are different.

2. A collection of operations for elements of the domain. This collection includes, for example, the usual arithmetic operations for domains of numeric elements. The collection can be empty.

An order can be defined for a domain. If so, there exists an operation that, given any pair of domain elements, determines the first or smaller element of the pair.

Domains can be general, application-independent sets (for example, the positive integers smaller than a given value,) or they can be specific to a particular database (for example, a domain of employee names).

Domains can be fixed during the lifetime of a database (like the set of positive integers smaller than 100), or they can vary with update operations

(like the set of student names in a university).

Domain elements have several different representations, including internal machine representations and external user-oriented representations. The latter are typically numbers and character strings. Domains of "surrogates" have also been included in several definitions of the relational model. They are domains whose elements have no external, user-oriented representation.

3.2 Relations

3.2.1 Name, Structure and Value

A relation has three fundamental attributes:

- a name
- a structure
- a value

The name of a relation uniquely identifies the relation among all the relations of a database.

The structure of a relation is specified by a set of attribute-domain pairs. All the attributes of a relation must be different. An attribute uniquely identifies the role played by a domain in a relation. The domains referenced in a relation are not necessarily all different. The collection of attribute-domain pairs of a relation is not ordered.

A relation named R with n attribute-domain pairs is noted as R(A1:D1,...,An:Dn), where the Ai's are the attributes and the Di's are the domains (more precisely, domain names). Formally, the correspondence between attributes and domains in a relation can be noted as a function {(A1->D1),...,(Ai->Di),...,(An->Dn)}.

At every moment, the relation value (or value) of a relation R(A1:D1,...,An:Dn) is a set of n-tuples <d1,...,dn> of elementary values, where di belongs to Di for all i between 1 and n. More exactly, the value of R is a set of labeled n-tuples <A1:d1,...,An:dn> of values, where each value is explicitly associated with an attribute. Thus, a labeled n-tuple is an unordered set of attribute-value pairs.

Formally, at every moment, the value of a relation R is a subset of the generalized Cartesian product of its domains indexed by its attributes. The indexing is specified by the correspondence between attributes and domains, described as a function in the relation structure.

If A = {A1,...,An}, then the relation value of R(A1:D1,...,An:Dn) is a subset of:

$$X(A:D) = A1:D1 \times \ldots \times An:Dn =$$

$$\{t : A \rightarrow \bigcup_{a \in A} Da \mid t \text{ is total and } \forall a \in A : t(a) \in Da\}$$

Thus, each relation tuple t of a relation value RV with attributes A is described formally as a total function on the set of attributes, having values in the associated domains. Therefore:

$$dom(t) = A$$

and, by extension

$$dom(RV) = A$$

This formal description of relation values is similar to what would be a description in terms of the "positional sets" of [HARD81].

The cardinality #RV of a relation value RV at a given moment is the number of tuples in RV. The degree (sometimes called arity) of a relation, relation structure, relation value, or labeled tuple is the number of attributes in it. An n-ary relation is a relation with n attributes.

We will sometimes speak of the "concatenation" of two tuples. Formally, the concatenation of two tuples with different sets of attributes is a tuple (i.e., a total function from attributes to a union of domains) equal to the set-theoretic union of the two argument tuples. The concatenation of t and t' will be noted t+t'.

3.2.2 Kinds of Relations in RDBMSs

Three kinds of relations are manipulated by relational database management systems (RDBMSs): database relations, derived relations, and relations denoted by queries.

1. Database relations are maintained permanently by the RDBMS. Their structure is declared in a relational database schema. The specification of the name and the structure of a database relation is sometimes called a relation schema (or scheme). The RDBMS permanently maintains a relation value for each database relation. The relation value is said to be an instance of the associated relation schema. The value of database relations can be modified by update operations.

2. Relations denoted by queries are, as their denomination suggests, temporary relations specified by expressions in query languages. They are not declared in a database schema or subschema and are uniquely associated with a query. Their structure and value are specified in terms of the structure and value of database relations by the syntax and semantics of query languages. The structure of such a relation can be derived from the form of the expression denoting it and from the structure of relations referenced in the expression. The value of the relation is obtained by evaluating the expression. Unlike database relations, relations denoted by queries do not have a simple name. In a sense, the expression itself specifying such a relation can be viewed as a name for the relation. Also, relations denoted by queries cannot be updated.

3. Derived relations or views are virtual relations that do not exist independently of database relations. A view is defined in a relational (sub)schema by giving a name, a structure, and an expression denoting a relation in terms of database relations (and, possibly, other views). Operations on views translate into operations on the underlying database relations: the relation value of a view is determined by evaluating the expression defining the view, and update operations on a view are translated into updates on the underlying database relations.

Different kinds of relations coexist only in specific operations of data manipulation languages. For example, an operation like store can be viewed as an assignment of the value of a relation denoted by a query to the name of a database relation.

Constraints may be imposed on relations and on operations on relations (see next section), and the particular constraints may vary according to the kind of relation. For example, the notion of primary key and restrictions attached to it might only be defined for database relations; also, update operations on views may be constrained so that their effect on underlying database relations can be uniquely determined.

3.3 Consistency (or Integrity) Constraints

A relational database schema can include constraints. In general, a consistency constraint is any assertion about the relation schemes of a relational database. A constraint is a time-independent statement that must be satisfied by all possible database instances of the relational schema for which the constraint is defined.

Some constraints apply only to one state of a database at a time; for example, the definition of a unique key. Other constraints apply to several states of a database considered together; for example, "salaries are nondecreasing". While constraints that refer to a single relation (e.g., keys and various kinds of dependencies) have been most thoroughly studied, in general, constraints can refer to several relations.

A language for expressing constraints can be any sufficiently powerful assertion language that can address a relational schema. Such a language can be derived from the relational algebra or from a relational calculus.

3.4 Relational Databases

A relational database consists of a time-independent, relational database schema and a time-dependent relation value for each database relation.

A relational database schema (or scheme) defines the time-independent properties of a database. It comprises the specifications of:

- A collection of domains
- A collection of relation schemes (or schemata)
- A collection of consistency constraints

3.5 Comments

3.5.1 Terminology

Relational terms are not used consistently throughout the literature, and therefore we had to make some terminology choices.

In this paper, a relation is defined as consisting of a fixed name, a fixed structure, and a time-varying value. Thus, our definition of relation embodies the concepts of "type" and "instance". With the concept of type of a relation is associated the set of potential instances or, in the terminology used here, the set of potential relation values: this set is defined by the relation structure and is the set of all subsets of the generalized Cartesian product of the relation domains indexed by the relation attributes. Thus, a relation value is an instance of the relation type.

Many authors call simply "relation" what we have called "relation value". Our goal was to have a special term for each separate notion. Note that our "relation value" contains all the information about attributes and domains, that is, all the information specified in the "relation structure". Other authors use "relation state" instead of "relation value".

We have reserved "relation schema" (or "relation scheme") for "database relations". Thus, a "relation schema" is that part of a database schema that concerns a particular relation. Other authors use "relation schema" to characterize the time-independent properties of every kind of relations (i.e., the relation name and the attribute-domain information). However, "relations denoted by queries" usually do not have a simple name, and their "schema" essentially reduces to what we call "structure" (that is, the set of attribute-domain pairs). That is why we reserve "relation schema" to database relations. Still other authors call "schema" what we call "structure", that is, the attribute-domain information.

Sometimes, a meaning of "attribute" is used that is slightly different from the one of this paper. Let A be an attribute of a relation R (as defined above) and t a tuple in the value of R. Then t(A) is an atomic value. Sometimes the pair (A,t(A)) is called an attribute, A is called the attribute name, and the element t(A) the attribute value. Other authors call "attribute name" what we call "attribute", and reserve "attribute" to refer to an attribute name and a domain (or domain name).

In this paper, all properties of tuples are subsumed by properties of the relations to which they belong, and no separate definitions are attached to the notion of tuple. In some other definitions, a tuple is defined as having a value and a structure. A type can thus also be attached to a tuple, but this is a different type from that attached to a relation. The set of potential instances attached to the tuple type is the indexed Cartesian product of domains indexed by attributes (instead of the set of all subsets of the indexed Cartesian product, as for the type attached to a relation). A relation value is then a time-varying set of instances of the tuple type. Thus, with that definition, a relation would be characterized by a name, a tuple type, and a "population" of tuple instances.

3.5.2 Table View of Relations

One of the advantages of the relational model that is sometimes emphasized heavily is that its data structures can be described informally as tables. Similar descriptions are also characterized as the "record view" or the "flat file view" of relations. The following is an informal definition of the table view of relations [SAND81]:

> "... The data are structured in the form of tables consisting of columns and rows, with the

rows corresponding to records or segments, and the columns representing fields within the records... The following rules must be followed:

- Each table contains only one record type.
- Each record (row) has a fixed number of fields, all of which are explicitly named.
- Fields are distinct (atomic) so that repeating groups are not allowed.
- Each record is unique - duplicates are not allowed.
- Records may come in any order; there is no predetermined sequence.
- Fields take their value from a domain of possible field values.
- The same domain may be used for many different field types, thus becoming the source of field values in different columns in the same or different tables."

Note that this definition can be made compatible with the terminology discussed above, if the following correspondence of terms is introduced:

- relation value = table
- type of relation tuples = record type (or table type)
- attribute = column name
- tuple = row = record
- N-ary relation = table with N columns
- etc.

The "rules" of the informal definition automatically hold for the quasi-formal definition (i.e., they are built-in); making them explicit would be redundant.

3.5.3 The Role of Domains

The choice of domains is an important decision in the design of a relational database. The definition of domains: (1) expresses some interrelation, structural information, and (2) constrains operations in such a way that any two atomic values can be compared to one another (e.g., for equality) only if they belong to the same domain. In programming language terms, the latter aspect is equivalent to associating a type to each domain [LACR81].

Compared to other presentations of the fundamental concepts of the relational model, the definitions given here may appear to have a "domain-oriented" flavor, in the sense that domains play an important role in the definitions. In many specifications of the relational model, domains are not explicitly defined; sometimes, they are not mentioned at all. This is not tenable in a precise definition, which must include some characterization of where individual elements of relation tuples come from.

One of two things is intended by specifications that do not mention domains:

1. Values for elements in relation tuples are taken from a single set D.
2. More often, the domains of the database are, by default, considered to be the predefined types of the underlying operating system (typically numbers and strings of characters).

A "one-domain" definition is a special case of a "many-domain" definition. In this case, all relations have a structure of the form R(A1:D,...,An:D), where all attributes are associated with the same domain. Formally, the indexed Cartesian product of domains becomes simply the set of total functions:

$$X(A:D) = \{f : A \rightarrow D\}$$

As above, the value of a relation R(A1:D,...,An:D) is a subset of the indexed Cartesian product X(A:D) = A1:Dx...xAn:D.

Formally, the definitions of algebraic operations are not very different in their one-domain version. Essentially, all conditions on compatibility of domains imposed in the many-domain definitions are relaxed in the one-domain definitions.

3.5.4 Representation of Domain Elements

The external representation of domain elements is typically some form of numbers or strings of characters. As indicated above, some definitions consider that domains <u>are</u> numbers or strings of characters. However, that is only one possibility, and the relational model can also support database-dependent domains.

In this paper, domains are considered to be disjoint. This considerably simplifies a precise definition of algebraic operations and is faithful to the usual, informal definitions. Supporting nondisjoint domains would require refinements of the definition of algebraic operations beyond what is usually understood as the relational model. Again in programming language terms, such refinements would be roughly equivalent to introducing types and subtypes into the relational model.

Even if domains are disjoint, their external representations usually are not, since they are expressed in terms of numbers and character strings. Thus, a domain value alone usually does not unambiguously determine a domain. However, in algebraic and calculus expressions, elementary values occur in contexts where they are associated with a domain via a relation attribute, so there is no real ambiguity. For example, suppose that both social security numbers and bank account numbers are represented externally as strings of digits. The query "List employees whose social security number is equal to their bank account number" is illegal, unless social security numbers and bank account numbers are defined to be the same domain.

4. DEFINITION OF THE RELATIONAL ALGEBRA

4.1 Introduction

This section presents a definition of algebraic operations for relations with unordered attributes as they are defined in Section 3. An algebraic operation has one or two operands which are relations. The operation denotes another relation, with a well defined structure (i.e., attributes and domains), and with a well defined value.

However, unlike a database relation, the relation denoted by an algebraic operation has no simple name.

The definition given in this section is general and precise. Derive a completely formal definition from the definition given here is straightforward.

4.2 Notation

For all the definitions which follow in this section, the following notation will be used.

R1(A:DA), R2(A:DA,B:DB) and so on, are shorthand notations respectively for R1(A1:D1,...,An:Dn) and R2(A1:D1,...,An:Dn,B1:D'1,...,Bm:D'm), with A = {A1,...,An} and B = {B1,...,Bm} and with A and B disjoint. Da (resp. DB) will also be used to denote the union of all domains associated with attributes A (resp. B) in R. Here, specifically, DA = {D1,...,Dn} and DB = {D'1,...,D'm}.

Sometimes, as a shorthand notation, domains will not be explicitly indicated when they are not explicitly referenced in the surrounding text. Thus, for example, sometimes R(A,B) will be written instead of R(A:DA,B:DB).

The language used to describe the semantics of the algebra is a mathematical language based on set-theoretical operations and on the predicate calculus. It is not defined explicitly because its notations are sufficiently familiar.

Typically, the value of a relation R(A:DA) will be specified by an expression of the form:

$$\{t : A \rightarrow DA \mid P(t)\}$$

where P(t) is a predicate with the tuple variable t as a free variable. If P is completely explicit about the set of attributes of the relation, then we will also write:

$$\{t \mid P(t)\}$$

4.3 Closure of the Algebra

4.3.1 Nesting

Algebraic operations take relations as operands and produce relations as results. Provided correspondence rules for attributes are defined, the algebra is closed under composition; that is, any algebraic expression can be used as operand of any other algebraic operation to any level of nesting. Note that restrictions on the generality of this composition rule may jeopardize relational completeness. This is sometimes overlooked in proofs of relational completeness of query languages.

Of the algebraic operations defined in this paper, only the "outer" operations defined in section 4.11 are to be excluded from this composition rule: they cannot be used as operands of other algebraic operations, because the definition of the other operations is not specified for operands with null values.

4.3.2 Renaming of Relation Attributes

An operation that renames the attributes of a relation can be presented as a function "rename(R,M)": its arguments are an expression denoting a relation R and a correspondence of attributes M, and its value is the relation with renamed attributes.

Formally, let A = {A1,...,An} be the attributes of R and B = {B1,...,Bn} be the desired attributes for the renamed relation. M can be described as a functional mapping {(Ai->Bj)} which actually is bijective or one-to-one, that is, functional in both directions. M can be a partial renaming which affects only a subset of the attributes of R. In particular, the renaming operation leaves its relation argument unchanged when this argument has no attribute subject to renaming.

Let A = A' U A", with A' and A" disjoint. A' is the list of attributes in dom(M), i.e., those attributes in A which undergo renaming. Thus, attributes A" are not affected by the renaming; if A" is empty, then all attributes of R are renamed. Let B be the new attributes and M = {(A'i->Bj)} be the correspondence of attributes. The renamed relation will have attributes A" U B, and each attribute Bj in B will be associated with the database domain of the corresponding attribute Ai, with Bj = M(Ai). Then, the formal definition of rename is as follows:

rename (R,M) =

$$\{t : A'' \cup B \rightarrow \bigcup_{x \in A''UB} Dx \mid \exists s \in R\ ((\forall a \in A''\ t(a)=s(a)) \text{ and } (\forall d \in A'\ t(M(d))=s(d)))\}$$

4.3.3 Generality of Attribute Correspondences

A general specification of attribute correspondences simply involves the possibility of renaming the attributes of any relation before applying a binary algebraic operation. Then the definition proper of binary operations can, without loss of generality, take advantage of special correspondences among attributes. The result of any algebraic operation directly inherits attributes from (possibly renamed) operands, and that result can, in turn, undergo renaming, if it appears as an operand of another binary operation.

For example, the definition of the natural join will suppose that (1) attributes that are the same in both operands will be put in correspondence in the join (i.e., the corresponding values will be tested for equality); (2) attributes from both operands that are not in correspondence are all distinct; (3) the set of attributes of the result is the union of the attributes of the operands. Renaming is available before and after the operation thus defined in order to ensure full generality.

The approach just outlined is followed in the definitions of the next sections. Enough information is provided informally to enable an easy construction of a formal definition.

A fully general specification could be characterized as one in which no condition is imposed on the attributes of the operands and no automatic inheritance of attributes between operands and result is imposed (although all attributes of the result must be distinct as for all "legal" rela-

tions). The information needed to put attributes of the operands into correspondence and to insure uniqueness of the attributes of the result then must be provided as extra arguments of the algebraic operations. In such a description, the natural join, for example, could be defined as an operation with four operands: two relations, a mapping of attributes to specify which attribute(s) of the operands are put into correspondence and tested for equality in the join, and a mapping of attributes to insure uniqueness of the attributes of the result. When actually worked out, such a description of algebraic operations looks much more complex than the description given in this paper, although both descriptions have the same generality. In effect, the description of this paper (which, in general, requires renaming of relation attributes before algebraic operations are applied) is equivalent to a description without renaming where algebraic operations have extra arguments to specify attributes correspondences: the extra arguments of the latter description are used to specify attribute renaming in the former.

Another typical description of algebraic operations addresses the problem of attribute correspondence by ordering them for each relation. Attributes essentially become indexes with consecutive integer values. Although no power of expression is lost, this kind of description is not really faithful to the abstract concept of relation that is independent of storage considerations.

4.4 Projection

Informally, the projection R[A1,...,An] of a relation R(A1,...,An,B1,...,Bm) is a relation obtained by retaining only the (A1,...,An) part of each tuple of R, and by suppressing all redundant duplicate tuples in the resulting relation.

Formally, given a tuple t:A->D of a relation R(A,B), the projection of t on attributes B is a tuple t′ defined as follows:

$$t' : B \to D \quad \text{such that } \forall b \in B \quad t'(b) = t(b)$$

The tuple t′ thus obtained will also be noted t[B].

Given a relation R(A:DA,B:DB), the projection of R on attributes B is a relation R1(B:DB) denoted R[B] (i.e., R[B1,...,Bm] if B = {B1,...,Bm}) whose value is given by:

$$\{s : B \to DB \mid \nexists t \in R \quad \forall b \in B \quad t(b) = s(b)\}$$

or, equivalently by:

$$\{t[B] \mid t \in R\}$$

4.5 Cartesian Product

The Cartesian product of two relations R1 and R2 is the set of all tuples t such that t is the concatenation of a tuple t1 belonging to R1 and a tuple t2 belonging to R2.

Formally, given R1(A:DA) and R2(B:DB) with A and B disjoint, the Cartesian product R1 x R2 of R1 and R2 is a relation R(A:DA,B:DB) whose value is given by:

$$\{t : A \cup B \to DA \cup DB \mid t[A] \in R1 \text{ and } t[B] \in R2\}$$

or, equivalently:

$$\{t1 + t2 \mid t1 \in R1 \text{ and } t2 \in R2\}$$

4.6 Union, Intersection, and Difference

4.6.1 Union-Compatibility

Given two relations with the same number of attributes and a one-to-one correspondence of attributes between the relations, the relations are union-compatible if corresponding attributes are associated with the same domain. Note that, in general, two relations can be union-compatible in more than one way, with different correspondences of attributes.

Formally, given two relations R1(A:DA) and R2(B:DB), and a one-to-one correspondence j between attributes in A = {A1,...,An} and attributes in B = {B1,...,Bn}, then R1 and R2 are union-compatible if for each a ∈ A and each b ∈ B such that b=j(a), it is true that Da=Db (or equivalently for each a ∈ A, it is true that Da=Dj(a)).

In the one-domain version of the relational model, two relations are union-compatible if they have the same number of attributes.

4.6.2 Ordinary Union, Intersection and Difference

Before defining union, intersection and difference of two union-compatible relations R1(A:DA) and R2(B:DB), suppose that the attributes B of R2 have been renamed as attributes A of R1, in such a way that attributes with the same name in both relations are those which are put into correspondence in the operations.

Union, intersection, and difference could be defined for relation operands with different attribute sets, but then a correspondence of attributes would have to be supplied as an extra operand. Thus this definition supposes that the attribute correspondence is applied as a renaming operation before the operations are executed.

Given two union-compatible relations R1 and R2, the union, intersection, and difference of R1 and R2 are defined as follows:

a. The union of R1 and R2 is the set of all tuples belonging to either R1 or R2 (or both).

b. The intersection of R1 and R2 is the set of all tuples belonging to both R1 and R2.

c. The difference of R1 and R2 is the set of all tuples belonging to R1 and not to R2.

Formally, given two union-compatible relations R1(A:DA) and R2(A:DA), then the union R1 U R2, the intersection R1 ∩ R2 and the difference R1 - R2 of R1 and R2 are relations R(A:DA) whose values are,

respectively:

$\{t \mid t \in R1 \text{ or } t \in R2\}$

$\{t \mid t \in R1 \text{ and } t \in R2\}$

$\{t \mid t \in R1 \text{ and } t \notin R2\}$

4.6.3 Bordered Union, Intersection and Difference

Bordered operations generalize union, intersection, and difference to relation operands that are not union-compatible.

The bordered union of R1 and R2 is the ordinary union of two union-compatible relations: one is obtained by completing or "bordering" each tuple of R1 by all possible values for the attributes of R2 that are not attributes of R1; the other is obtained similarly by bordering each tuple of R2 with all possible values for the attributes of R1 that are not attributes of R2. Bordered intersection and difference are similar.

Formally, let R1(A:DA,B:DB) and R2(B:DB,C:DC) be two relations which are such that, after suitable renamings, (1) each attribute in B is associated with the same domain in both R1 and R2, and (2) attributes in A are all different from attributes in C.

Then the bordered union R1 $\underline{\cup}$ R2 of R1 and R2 is a relation R(A:DA,B:DB,C:DC) whose value is defined as follows:

$$(R1 \times X(C:D)) \cup (R2 \times X(C:D))$$

where "U" is the ordinary union operator, and R1 x X(C:D) is a relation with attributes A U B U C whose value is obtained by concatenating every tuple of R1 with every tuple of the indexed Cartesian product of domains X(C:D). R2 x X(A:D) has a similar structure.

Similarly, the bordered intersection R1 $\underline{\cap}$ R2 and the bordered difference R1 $\underline{-}$ R2 of R1(A:DA) and R2(A:DA) are relations R(A:DA) whose values are given respectively by:

$$(R1 \times X(C:D)) \cap (R2 \times X(A:D))$$

$$(R1 \times X(C:D)) - (R2 \times X(A:D))$$

where "$\cap$" and "-" are respectively the ordinary intersection and difference defined in Section 4.6.2.

The operations include the ordinary union, intersection, and difference as special cases (when both A and C are empty). They are useful, for example, to describe the value of domain relational calculus expressions in terms of algebraic operations [LOUI82].

It is interesting to note that bordered intersection is equivalent to the natural join defined in section 4.8.2.

4.6.4 Another Generalization of Difference

Let R1(A:DA,B:DB) and R2(B:DB,C:DC) be two relations such that, after suitable renamings, (1) each attribute Bi in B is associated with the same domain in both R1 and R2, and (2) attributes in A are all different from attributes in C.

The following operation generalizes the difference operation and reduces to it when A and C are empty. The generalized difference of R1 and R2 is a relation R(A:DA,B:DB) whose value is at all times defined as:

$$\{t \mid t \in R1 \text{ and } t[B] \notin R2[B]\}$$

or equivalently

$$R1 - (R1[A] \times R2[B])$$

where "-" is the "ordinary" difference operation defined in section 4.6.2 and "x" is the Cartesian product defined in section 4.5. This generalization of difference is similar to the "generalized difference" of [HALL75].

4.7 Restriction (or Selection)

Given a relation R, the restriction of R is the subset of tuples of R which satisfy a specified condition.

Formally, let R(A:DA) be a relation, with A = {A1,...,An}. Then R[Ai θ c], where Ai is an attribute in A, is a restriction of R, if c is a constant of the domain Di associated with Ai and θ is any binary comparison operation (such as EQ,NE,GT,GE,LT,LE) defined for values of that domain.

The relation value R1 of R[Ai θ c] is given by:

$$\{t \mid t \in R \text{ and } t(Ai)\ \theta\ c\}$$

Another form of restriction is written R[Ai θ Aj], where Ai and Aj must be associated with the same domain in relation R, and θ is a binary comparison operation defined for values of that domain. The value of R[Ai θ Aj] is given by:

$$\{t \mid t \in R \text{ and } t(Ai)\ \theta\ t(Aj)\}$$

The generalization to an operation with several elementary comparisons is straightforward:

$$R\ [Ai\ \theta i\ ci,\ldots,Ak\ \theta k\ ck,Al\ \theta l\ Am,\ldots,Ap\ \theta p\ Aq]$$

It is equivalent to the composition of an arbitrary number of elementary restrictions with constants and an arbitrary number of elementary restrictions that compare attribute values. Restriction can be further generalized as in [PECH75] or [MERR78] so that the selection criterion is a Boolean combination of elementary comparisons. Then, that form of restriction is in general equivalent to a combination of set operations (union, intersection, and difference) with operand relations that are obtained as values of elementary restrictions defined above.

A more general view of restriction would consist in defining the selection criterion to be any predicate that selects some of the tuples of the restricted relation. Thus, for example, the semijoin operation of [BERN81] or the QT-selector of [MERR78] could be viewed as a form of restriction. The QT-selector of [MERR78], together with the

Cartesian product, is general enough to express θ-join, natural join, and division.

4.8 Join

4.8.1 θ-Join

Given two relations R1 and R2, and a condition A1 θ B1 where A1 is an attribute of R1 and B1 is an attribute of R2, the θ-join of R1 and R2 is the set of all tuples t such that t is the concatenation of a tuple t1 of R1 and a tuple t2 of R2 such that the condition t1(A1) θ t2(B1) is satisfied.

Formally, let R1(A:DA) and R2(B:DB) be two relations and assume that their attributes have been renamed so that they have no attribute in common. Then "R1[A1 θ B1]R2" is a θ-join of R1 and R2, if A1 is an attribute of R1 and B1 is an attribute of R2, which are associated in R1 and R2 with the same domain, and if θ is a binary operation (like EQ,NE,GT,GE,LT,LE) defined for values of that domain.

The relation value of R1[A1 θ B1]R2 is defined to be:

```
{t : A U B -> DA U DB |
  t[A] ∈ R1 and t[B] ∈ R2 and t(A1) θ t(B1)}
```

or, equivalently:

```
{t1 + t2 | t1 ∈ R1 and t2 ∈ R2 and t1(A1) θ t2(B1)}
```

It is clear from that definition that the operation is equivalent to the Cartesian product of R1 and R2 restricted to those tuples where the condition "A1 θ B1" holds.

The generalization to an operation where several elementary restrictions are imposed is straightforward. Thus, the value of R1[A1 θ1 B1,...,Ak θk Bk]R2 is defined to be:

```
{t1 + t2 | t1 ∈ R1 and t2 ∈ R2
       and t1(A1) θ1 t2(B1)
       and  ...
       and t1(Ak) θk t2(Bk)}
```

The operation is equivalent to the Cartesian product of R1 and R2 restricted to those tuples t where the conditions "t(Ai) θi t(Bi)" hold for $1 \leq i \leq k$.

Each attribute A1,...,Ak of R1 that participates in the join can participate more than once (and similarly for B1,...,Bk), but the same pair (Ai,Bi) can only participate with one operator. Formally, the correspondence of attributes in the join can be represented by a functional mapping {((Ai,Bi)->opi)} from pairs of attributes to operators.

Just as restriction, θ-join can be further generalized so that the selection criterion is a Boolean combination of elementary restrictions.

4.8.2 Natural Join

Informally, the natural join of two relations R1 and R2 is the equi-join (θ-join with "θ" being "=") of the two relations on their corresponding attributes, with only one of each pair of corresponding attributes being retained in the result.

Formally, let R1(A:DA,B:DB) and R2(B:DB,C:DC) be two relations such that, after suitable renamings, (1) each attribute in B is associated with the same domain in both R1 and R2, and (2) attributes in A are all different from attributes in C.

The natural join R1*R2 of R1 and R2 is a relation R(A:DA,B:DB,C:DC) whose value is given by:

```
{t : A U B U C -> DA U DB U DC  |
          t[A U B] ∈ R1  and  t[B U C] ∈ R2}
```

or, equivalently:

```
{t1 + t2[C] |
  t1 ∈ R1  and  t2 ∈ R2 and t1[B] = t2[B]}
```

A number of special cases of natural join are worth pointing out:

- when B is the empty collection of attributes, the definition of natural join reduces to that of the Cartesian product (see section 4.5);
- when A and C are empty, the definition reduces to that of intersection (see section 4.6.2);
- when A or C is empty (but not both), the operation is called a semi-join [BERN81]. The relation value of the result is a subset of the value of one of the operand relation, and, in some sense, the semi-join is similar to a restriction. For example:

```
R1(A:DA,B:DB) * R2(B:DB) =

        {t | t ∈ R1 and t[B] ∈ R2}
```

The natural join given here is equivalent to the bordered intersection defined in section 4.6.3. The definition is similar to that of "generalized intersection" in [HALL75].

The preceding definition generalizes the usual definition of the natural join in that more than one attribute in general (i.e., attributes B in the definition above) can be common to the operand relations and can participate in the equality test of the join. The natural join on several common attributes is equivalent to a natural join on one common attribute, followed by a restriction that tests equality of each pair of the remaining common attributes, itself followed by a projection that retains only one attribute of each pair of common attributes.

4.9 Division

Three cases of division will be defined. Intuitively, in terms of predicate calculus formulas, the structure of the operations corresponds

respectively to:

$\{X \mid \forall Y\ P(X,Y)\}$

$\{X \mid \forall Y\ Q(Y) \rightarrow P(X,Y)\}$

$\{(X,Z) \mid (\exists Y\ Q(Y,Z)) \text{ and } \forall Y'\ Q(Y',Z) \rightarrow P(X,Y')\}$

The second case corresponds to the division operation usually given in definitions of the relational algebra. The other two are new generalizations.

The usual definition of division involves a comparison of two sets of values constructed from the operand relations. The three definitions given here generalize the usual definitions in a manner similar to the generalization of the natural join: they involve in general a comparison of two sets of tuples of values constructed from projections of the operand relations. In other words, in terms of the calculus notation above, Y can stand for more than one variable ("$\forall$ y1, $\forall$ y2, ..."), and, in the definitions below, B need not be a single attribute as in the usual definition of division, but it can instead stand for several attributes.

4.9.1 The Divisor Is a Domain or a Cartesian Product of Domains

The division of a relation R(A1:D1,A2:D2) by the domain D2 is a relation that contains those elements of the projection R[A1] each of which is associated in R with all the elements of the domain D2.

Formally, the definition generalizes to several domains as follows. The division of a relation R(A:DA,B:DB) by the domain(s) DB is a relation R1(A:DA), where the domain associated with each attribute in R1 is the same as the domain associated with the same attribute in R. The value of R1 is given by:

$$R(A,B) / DB = \{t \mid t \in R[A] \text{ and } \{t\} \times X(B:DB) \subseteq R\}$$

where "{t}" is the singleton comprising t as its only element (note that "{t}" is a relation value), "x" is the Cartesian product operation for relation operands defined in Section 4.5 and "X(B:DB)" is the Cartesian product of domain(s) DB indexed by the attribute(s) B. Note that, in this formula, "$t \in R[A]$" is implied by "$\{t\} \times X(B:DB) \subseteq R$", and is thus redundant.

Note that if the relation value of R(A,B) is empty, then the division R(A,B)/DB also has an empty relation value.

4.9.2 The Divisor Is a Relation All of Whose Attributes Participate in the Division

The division of a relation R1(A,B) by a relation R2(B) is a relation that contains those elements of the projection R1[A] each of which is associated in R1 with all the elements of R2.

Formally, let R1(A:DA,B:DB) and R2(B:DB) be relations where each attribute in B is associated with the same domain in both R1 and R2.

The division of R1 by R2 is a relation R(A:DA), where the domain associated with each attribute in R is the same as the domain associated with the same attribute in R1. The value of R is given by:

$$R1(A,B) / R2(B) = R(A) = \{t \mid t \in R1[A] \text{ and } \{t\} \times R2 \subseteq R1\}$$

Note that if R(A,B) has an empty relation value, then the division R1(A,B)/R2(B) also has an empty relation value. If the value of R2(B) is empty, then R1(A,B)/R2(B) is equal to the projection R1[A]. Another sensible definition of division could rule out the latter case by requiring that the division R1(A,B)/R2(B) be defined only when R2(B) does not have an empty relation value.

4.9.3 The Divisor Is any Relation Which Has at Least a Domain in Common With the Dividend

The third case of division is a new operation that was not included in previous definitions of the relational algebra. It produces a relation built from attributes of both relation operands. The division of R1(A1,A2) by a relation R2(A2,A3) is a set of labeled tuples t = <A1:a1,A3:a3>, each of which is such that the sets of values a2 associated in R2 with the value a3 (=t(A3)) is contained in the set of values a2 associated in R1 with the value a1 (=t(A1)).

Formally, the definition generalizes to several attributes as follows. Let R1(A:DA,B:DB) and R2(B:DB,C:DC) be two relations such that, after suitable renamings, (1) each attribute in B is associated with the same domain in both R1 and R2, (2) attributes in A are all different from attributes in C, and (3) R2 does not have an empty relation value.

The division of R1 by R2 on their common attributes B is a relation R(A:DA,C:DC), where the domain associated with each attribute in R is the same as the domain associated with the same attribute in R1 or R2. The value of R is given by:

$$R1(A,B) / R2(B,C) = R(A,C) = \{t : A \cup C \rightarrow DA \cup DC \mid t[A] \in R1[A] \text{ and } t[C] \in R2[C] \text{ and } (R2[C=t[C]][B]) \subseteq R1[A=t[A]][B])\}$$

"R2[C=t[C]][B]" expresses a restriction "R2[C=t[C]]" followed by a projection on attributes B. If C = {C1,...,Cn}, then "R2[C=t[C]]" is a shorthand notation for "R2[C1=t(C1),...,Cn=t(Cn)]". "R1[A=t[A]][B]" has a similar structure.

An equivalent, simpler notation is the following:

$$R1(A,B) / R2(B,C) = R(A,C) = \{t1 + t2 \mid t1 \in R1[A] \text{ and } t2 \in R2[C] \text{ and } (R2[C=t2][B]) \subseteq (R1[A=t1][B])\}$$

When R1(A,B) or R2(B,C) has an empty relation value then so does the division. Another sensible definition could define this general case of division only for relations R2 that do not have an empty relation value.

Note that, with this notation, the second case of division in the preceding section can also be

defined as:

$$R1(A,B) \,/\, R2(B) = \{t \mid t \in R1[A] \text{ and } R2 \subseteq R1[A=t][B]\}$$

Recently, an operation very similar to our general case of division has been studied by Demolombe [DEMO82].

4.10 Tuple Constructor and Literal Relation

Relations are needed that are defined in-line in the text of a query. The value of such a relation is specified as a set of tuples, and each tuple is built by a tuple constructor. Thus, a literal relation with n attributes appears as follows:

```
{<A1:u1 , ... , An:un>,
 <A1:v1 , ... , An:vn>,
 ...
 <A1:w1 , ... , An:wn>}
```

where "<A1:u1,...,An:vn>" is a tuple constructor; A1, ..., An are n distinct attributes; u1, ..., un, v1, ..., vn, w1, ..., wn are elementary values. All the tuples in a literal relation must have the same set of attributes.

The elementary values ui, vi, ..., wi associated with the same attribute Ai in all tuples belong to one domain Di of the database. However (see section 3.5.4), the external representation of an elementary value usually does not uniquely identify one domain.

A general solution to this problem is for a literal relation to make explicit the domains associated with the attributes. Thus, a literal relation would appear as something like:

```
[ <A1:D1 , ... , An:Dn> :
  {<A1:u1 , ... , An:un>,
   <A1:v1 , ... , An:vn>,
   ...
   <A1:w1 , ... , An:wn>} ]
```

A shorter equivalent notation leaves out the attributes in the n-tuples of data values, since they are supplied by the header n-tuple specifying the structure information. The data n-tuples thus become, for this operation only, ordered n-tuples of values:

```
[ <A1:D1 , ... , An:Dn> :
  {<u1 , ... , un>,
   <v1 , ... , vn>,
   ...
   <w1 , ... , wn>} ]
```

Another solution is to rely on the context in which a literal relation appears in a query. A literal relation is combined with other relations in algebraic operations that impose conditions on the correspondence of domains of their relation arguments. These correspondence rules enable the determination of domains for the literal relation. In this second solution where domains are not mentioned explicitly in the literal relation itself, the same conditions must be satisfied by the elementary values u1,v1,... as in the general case: their external representation must be consistent with the external representation of elementary values of the domains to which they belong.

4.11 Outer Join, Union, Intersection, and Difference

4.11.1 Manipulation of null values

The description method used in this paper extends naturally to the manipulation of null values. This section defines algebraic operations that create relations with null values from operand relations without null values. A version of algebraic operations, generalized to handle operand relations that contain null values, for example as in [CODD79], could also be defined easily with the specification method of this paper. The difficulty with null values is not how to describe algebraic operations that manipulate them. The difficult problem is to decide exactly what is to be described, that is, the possible meanings of null values and how much of these meanings has to be or can be embedded in the definition of algebraic operations. Specifying operations that faithfully handle different shades of meanings of "null", and that at the same time produce definitions of reasonable size and complexity, remains an unsolved problem.

4.11.2 Outer Natural Join

Let @ be a "null" value whose meaning is not otherwise specified here, and suppose, for this definition only, that @ belongs to each database domain. Or, more precisely, as domains are disjoint, suppose that there is one different null value for each domain. Let $\emptyset = \{@\}$.

As an extension of notations, $X(A:\emptyset)$ will indicate, for some collection of attributes A, that @ is associated with all attributes in A. Strictly speaking, for each attribute in A, the null value concerned is that of the domain associated with the attribute in question. Thus $X(A:\emptyset)$ denotes a labeled tuple of null values <A1:@,...,An:@> if A = {A1,...,An} and the null value associated with attribute Ai is considered to belong to the corresponding database domain Di associated with Ai.

For joining R1(A:DA,B:DB) and R2(B:DB,C:DC), suppose as for the ordinary natural join that, after suitable renamings, (1) each attribute in B is associated with the same domain in both R1 and R2, and (2) attributes in A are all different from attributes in C.

Let R(A:DA,B:DB,C:DC) be the ordinary natural join as defined in section 4.8.2. The symmetric outer natural join R′ of R1 and R2 is a relation R′(A:DA,B:DB,C:DC) whose value is defined as follows:

$$R' = R \cup ((R1-R[A,B]) \times X(C:\emptyset)) \cup ((R2-R[B,C]) \times X(A:\emptyset))$$

This operation is defined to generalize the join operation in such a way that projections of the outer join are always equal to the operands

(the symmetric outer join is a "lossless" join). With the notations above:

$$R'[A] = R1 \quad \text{and} \quad R'[B] = R2$$

Outer joins were proposed by a number of researchers (e.g., [CODD79] and [LACR76]). Outer θ-joins, and asymmetric or unidirectional outer joins also were defined.

4.11.3 Outer Union, Intersection, and Difference

The definition of outer union, intersection, and difference has the same structure as that of the corresponding bordered operations defined in section 4.6.3. The outer union of R1 and R2 is the union of two other relations: one is obtained by bordering each tuple of R1 with a null value for each of the attributes of R2 that is not an attribute of R1; the other one is obtained similarly by bordering each tuple of R2 with a null value for each of the attributes of R1 that is not an attribute of R2. Outer intersection and difference are similar.

Formally, let R1(A:DA,B:DB) and R2(B:DB,C:DC) be two relations such that, after suitable renamings, (1) each attribute in B is associated with the same domain in both R1 and R2, and (2) attributes in A are all different from attributes in C. The conventions and notations for null values are the same as for the definition of outer join.

The outer union, intersection, and difference of R1 and R2 are relations R(A:DA,B:DB,C:DC) whose value is defined, respectively, as:

$$(R1 \times X(C:\emptyset)) \cup (R2 \times X(A:\emptyset))$$

$$(R1 \times X(C:\emptyset)) \cap (R2 \times X(A:\emptyset))$$

$$(R1 \times X(C:\emptyset)) - (R2 \times X(A:\emptyset))$$

4.12 A Word About Empty Relation Values

Except possibly for the second operand of division, the definitions of algebraic operations is valid for operands with empty relation values. Specifically:

- restriction, intersection, difference, join and division can produce results with empty relation values from operands with nonempty values, whereas projection, Cartesian product, and union cannot produce empty results from nonempty operands;
- projection, Cartesian product, intersection, restriction, and join produce an empty result if they have an empty operand; the same is true for difference and division when their left (first) operand is empty;
- union can have an empty operand and a nonempty result; the same is true of difference with its right (second) operand empty;
- the only case where an operand could be forbidden to have an empty value is that of the second (right) operand of division.

4.13 Completeness of the Relational Algebra

Relational completeness is a measure of the selective power of a query language. In [CODD72], a relational algebra and the tuple relational calculus are proposed as equivalent measures of relational completeness.

In the algebra of [CODD72], union, difference, Cartesian product, projection, and restriction form a relationally complete subset of algebraic operations from which intersection, division, θ-join and natural join can be derived. In effect, it is obvious from their definitions that θ-join is equivalent to a Cartesian product followed by a restriction, and that natural join is equivalent to a θ-join followed by a projection. Intersection is equivalent to a natural join on one common attribute, followed by restrictions to test equality of the other common attributes and by projections to keep only one copy of each common attribute. The division of [CODD72] can be expressed as follows in terms of other operations:

$$R1(A,B)/R2(B) = R1[A] - ((R1[A] \times R2) - R1)[A]$$

The algebraic operations defined in this paper relate as follows to the operations of [CODD72]. Projection, Cartesian product, and ordinary union, intersection, and difference are essentially the same operations. Bordered intersection contains as special cases, and is not more general than, Cartesian product, ordinary intersection and natural join. Bordered union and difference contain as special cases ordinary union and difference respectively, and are strictly more general than the corresponding ordinary operations. The general restriction with a Boolean selection formula of section 4.7 is equivalent to a combination of unions (corresponding to the Boolean "or") and simple restrictions (expressing the equivalent of the Boolean "and" and "not"). The natural join and θ-join of section 4.8 are equivalent to a Cartesian product followed by a restriction and a projection.

The three cases of division can be expressed in terms of other operations as follows:

$$R(A,B)/DB = R[A] - ((R[A] \times X(B:DB)) - R)[A]$$

$$R1(A,B)/R2(B) = R1[A] - ((R1[A] \times R2) - R1)[A]$$

$$R1(A,B)/R2(B,C) = (R1[A] \times R2[C]) - (((R1[A] \times R2) - (R1 \times R2[C]))[A,C])$$

Outer join, union, intersection, and difference of Section 4.11 cannot be nested as operands of other algebraic operations, because the behavior of the other operations has not been defined when their operands contain null values. Thus the concept of relational completeness as usually defined does not include the capabilities of these "outer" operations.

In summary, leaving aside the "outer" operations, the only algebraic operations defined in this paper that are not equivalent to a combination of operations of the algebra of [CODD72] are the first case of division, which explicitly references a database domain (or a Cartesian product of domains), and the bordered union and difference.

5. ALGEBRAIC OPERATIONS FOR BOOLEAN VALUES

In query languages based on the predicate calculus, it is natural to interpret calculus formulas without free variables (closed formulas) as denoting truth values ('true' or 'false'). Thus "yes-no" queries can be expressed as closed formulas.

Algebraic expressions can be written whose value is a degenerate relation without attributes. Such degenerate relations can be interpreted as denoting truth values, and thus as representing "yes-no" queries within an extended algebra. The semi-formal definitions of section 3 are extended to degenerate relations in [LOUI82].

Specifically, the algebraic operations fall into three categories concerning their interactions with degenerate relations:

1. Projection and division can produce degenerate relations from ordinary ones. Their definition can be generalized as follows.

   Projection:

   For A empty : R[A] = 'false' if R is empty
   = 'true' if R is not empty

   Division:

   Division produces a relation without attributes if the set of attributes of the dividend is the same as the set of attributes of the divisor. Only the first two cases of division defined in section 4.9 are relevant.

   R(B:DB) / DB = 'true' if R = X(B:DB)

   = 'false' otherwise

   R1(B:DB) / R2(B:DB) = 'true' if R2 $\subseteq$ R1

   = 'false' otherwise

2. Union, difference, and intersection do not produce degenerate relations. They are extended to operands that are degenerate, and have the same semantics as the corresponding Boolean operations, namely, respectively, disjunction, complement, and conjunction.

3. Cartesian product, restriction, and join do not produce degenerate results and are not extended to degenerate operands.

6. CONCRETE SYNTAXES FOR THE RELATIONAL ALGEBRA

The following is a BNF syntax of algebraic expressions without "outer" operations. It defines a purely applicative notation, where every algebraic expression can be used as argument of any algebraic operation.

Note that, as discussed in section 4.3, renaming appears as an operation of the language: it controls the attributes of operands of other operations so that they obey the conditions on attribute correspondence introduced in the definition of those operations. Parentheses are freely available to avoid ambiguities.

```
R ::
 R"["atlist"]"              | -projection-
 R"x"R                      | -Cartesian product-
 R"U"R                      | -union-
 R"⊎"R                      | -bordered union-
 R"⋂"R                      | -intersection-
 R"⊓"R                      | -bordered intersection-
 R"-"R                      | -difference-
 R"⊟"R                      | - bordered difference-
 R"["comp{","comp}*"]"      | -restriction-
 R"["comp{","comp}*"]"R     | -θ-join-
 R"*"R                      | -natural join-
 R"/("dom{","dom}*")"       | -division-
 R"/"R                      | -division-
 "rename("R",("corat"))"    | -renamed attributes-
 rname                      | -database relation-
 "["struct":{"tuple{","tuple}*"}]"
                              -literal relation-

struct :: "<"at":"dom{","at":"dom}*">"
atlist :: at | at "," atlist
comp   :: at op at | at op constant
corat  :: cor | cor "," corat
cor    :: at "->" at
op     :: "=", "≠", "<", ">", "<=", ">=", etc.
tuple  :: "<"[at":"]constant{","[at":"]constant}*">"
rname  :: <...>
            -(sub)schema relation names-
dom    :: <...>
            -(sub)schema domain names-
at     :: <...>
            -(subschema) attributes or
             attributes chosen by user-
constant :: <...>
            -external representations of domain
             elements-
```

All terminal symbols of the context-free grammar appear between double quotation marks. "{A}*" means any number, including zero, of instances of A, while "[A]" means an optional instance of A (i.e., zero or one instance of A). "<...>" stands for a nonterminal not defined in this syntax. Such are (sub)schema objects or names chosen by users for which different external representations can be chosen for each implementation. In the syntax rules, comments appear between hyphens.

This paper focuses on the semantics or functionality of the relational model, as recommended in [RTG82]. The design of a user interface to a relational DBMS must also take into account human factors, which should guide the selection of a particular syntax and of groupings of operations, in a manner adequate for the intended users of the interface.

For example, a more procedural notation could present an algebraic expression as a sequence of statements, where each statement assigns the result of an algebraic operation to an intermediate relation variable. Intermediate notations between the purely functional and the purely procedural can also be defined.

Also some specific syntactic sugar can be added to specialize the various algebraic operations. For example:

PROJECT <algebraic expression> ON (A,B)

or PROJECT OUT C FROM <algebraic expression>

where A,B and C are attributes.

All such syntactic variations define languages with the same functionality.

7. THE RELATIONAL ALGEBRA AS AN ASSERTION LANGUAGE

The relational algebra can be used to express assertions about a relational database. In a context where an assertion is expected, the idea is to interpret as assertions the algebraic expressions having a truth value. This is straightforward and requires only some syntax (e.g., a keyword like "assert" to indicate that an assertion follows). Thus, the algebra can be used to express consistency constraints about a relational schema.

8. ACKNOWLEDGEMENTS

This work was initiated during stimulating discussions within the ANSI/X3/SPARC DBS-SG Relational Database Task Group [RTG82], particularly with Michael Brodie, Dan Ries, and Joachim Schmidt. I also benefited from fruitful exchanges of ideas with Georges Louis, of Philips, and Frank Manola and Ronni Rosenberg, of Computer Corporation of America.

9. REFERENCES

[BERN81]
Bernstein, P.A., D.W. Chiu, "Using Semi-Joins to Solve Relational Queries", J.ACM, Vol.28 No 1, January 1981.

[BJOR80]
Bjorner, D., "Formalization of Data Base Models", IN: Abstract Software Specifications, Bjorner Ed., Springer Lecture Notes in Computer Science 86, 1980.

[CODD72]
Codd, E.F., "Relational Completeness of Database Sublanguages", In: Database Systems, Courant Computer Science Symposium 6, Prentice-Hall (1972).

[CODD79]
Codd, E.F., "Extending the Database Relational Model to Capture More Meaning", ACM Trans. on Database Systems, Vol.4 No 4, December 1979.

[DEMO82]
Demolombe, R., "Generalized Division for Relational Algebraic Language", to appear in Information Processing Letters.

[HALL75]
Hall, P.A.V., P. Hitchcock, and S.J.P. Todd, "An Algebra of Relations for Machine Computation", Proc. 2nd ACM Symposium on Principles of Programming Languages, Palo Alto (1975).

[HARD81]
Hardgrave, T.W., "Positional Set Notation", to appear in Advances in Database Management, Vol.2, Heyden and Son, 1981.

[LACR76]
Lacroix, M., and A. Pirotte, "Generalized Joins", ACM Sigmod Record, Vol.8 No 3, September 1976.

[LACR81]
Lacroix, M., and A. Pirotte, "Associating Types with Domains of Relational Databases", Proc. ACM-NBS Workshop on Data Abstraction, Databases, and Conceptual Modeling, Sigmod Record Vol.11 No 2, January 1981.

[LAME80]
Lamersdorf, W., and J.W. Schmidt, "Semantic Definition of PASCAL/R", Berichte Nr. 73 and 74, University of Hamburg, 1980.

[LOUI82]
Louis, G., and A. Pirotte, "A Denotational Definition of the Semantics of DRC, the Domain Relational Calculus", Proc. VLDB Conference, Mexico, 1982.

[MERR78]
Merrett, T.H., "The Extended Relational Algebra, a Basis for Query Languages", In Databases: Improving Usability and Responsiveness, Shneiderman Ed., Academic Press, 1978.

[PECH75]
Pecherer, R.M., "Efficient Evaluation of Expressions in a Relational Algebra", Proc. ACM Pacific Regional Conference, April 1975.

[RTG82]
ANSI/X3/SPARC DBSSG Relational Task Group, Final Report, M.Brodie and J.Schmidt Eds., to appear in ACM SIGMOD Record.

[SAND81]
Sandberg, G., "A Primer on Relational Database Concepts", IBM Systems Journal, Vol.20 No 1, 1981.