A Denotational Definition of the Semantics of DRC, A Domain Relational Calculus

Georges Louis
Philips Research Laboratory
Bruxelles, Belgium

and

Alain Pirotte
Computer Corporation of America
Cambridge, Massachusetts, USA

ABSTRACT

This paper presents a semi-formal denotational definition of the semantics of a version of domain relational calculus called DRC. A single basic design principle governs the semantic definition: each predicate (or formula) of DRC denotes a relation.

The definition obtained is precise, short, and systematic. Generalizations of operations of the relational algebra are suggested, which correspond very directly with the semantics of DRC formulas.

This work also suggests a more active role of semantic considerations in the design process of a query language, in order to simplify the specification of the language and, eventually, the language itself.

1. INTRODUCTION

A relatively recent development in the evolution of programming languages has been the use of formal specifications of language semantics, in addition to the more classical formal specification of syntax. This formal specification of semantics has slowly begun to replace the usual informal description in English. Thus, for example, the classical disciplines of logic and linguistics have given rise to the denotational approach to programming language semantics [STOY77].

In contrast to the programming language field, the field of database query languages does not have a tradition of precise (let alone formal) definition of semantics. This has created a number of problems. It is not uncommon, for example, to see an initial language design based on an intuitively appealing idea, but with the semantics left intuitive as well. Then, when more advanced investigations reveal that the language is not powerful enough, the initial design is patched with new constructs. Such a design strategy in successive stages leads to layered languages lacking conceptual unity. Other problems are created when implementors are not provided with completely precise specifications. It is not uncommon to see the limits of a query language actually defined by language processors rather than by a reference document. This situation reduces portability and results in a more difficult learning process for users.

There is no fundamental reason, however, for database query languages to be defined less precisely than programming languages. Indeed, if anything, the converse could be true, since typical query languages are simpler than typical programming languages. For example, the mathematical foundations of the denotational definition of programming languages involve sophisticated mathematical concepts. Yet, precise definitions of high-level relational query languages can be produced with very little formalism. These definitions can be remarkably concise and easy to understand.

This paper illustrates the latter point by presenting a semi-formal denotational definition of a version of domain relational calculus called DRC [LACR77, ULLM80, DATE81]. For the clarity of this paper, we preferred a semi-formal definition to a formal one. It will be clear however that constructing one from the other is easy.

A denotational definition of a language describes the semantics of language constructs as functions from syntactic structures to mathematical objects, in such a way that the semantics or "denotation" of a composite construct is expressed as a combination of primitive objects and of denotations of the immediate constituents of the construct.

A single basic design principle governs the semantic definition of DRC given in this paper: each formula (or predicate) denotes a relation. All the details of the definition follow from that principle. The semantics of DRC is strictly "bottom-up": for each formula F of DRC, the structure and value of the relation denoted by F do not depend on the context in which F occurs in a DRC query. In fact, the only formal structure

A.Pirotte's work was partially supported by the National Bureau of Standards under Contracts NB79SBCA0088 and NB79SBCA0086.

Authors' present address: Philips Research Laboratory, 2 avenue Van Becelaere, 1170 Bruxelles, Belgium.

needed is the definition of relations as sets of tuples with selectors (or attributes).

The rest of this paper is organized as follows. Section 2 contains an informal definition of DRC. Section 3 is a formal definition of relations. Section 4 gives the precise semi-formal definition of DRC, consisting of a syntactic definition based on a BNF grammar, and a set of semantic rules. Section 5 relates the semantic definition of DRC to a version of relational algebra. Section 6 summarizes the results of this paper, and advocates a more active role for semantic specifications in the query language design process.

2. INFORMAL DEFINITION OF DRC

DRC is a version of domain relational calculus [LACR77, ULLM80, DATE81]. It has the same power as the tuple relational calculus or the usual relational algebra [CODD72].

This section gives an intuitive definition of the semantics of DRC. The definition proceeds from the root to the leaves in the syntactic structure of queries. The definition is very similar to a usual way of understanding the structure of the predicate calculus, where quantifiers translate into conjunctions and disjunctions, and predicates have a truth value. The definition accounts for both "list" queries and "yes-no" queries.

DRC variables are called domain variables: each variable ranges on the elements of a domain of elementary values of the database. The association of a variable with a domain is specified by some form of explicit or implicit declaration.

A query in DRC has one of two possible forms:

- an open query (or list query) has the form

  {(x1,...,xn) | P (x1,...,xn)}

  where P(x1,...,xn) is a formula of DRC with x1,...,xn as free variables. Its value is the set of labeled tuples <x1:a1,...,xn:an> such that P(a1,...,an) is "true". For each i, ai is an element of the domain associated with xi. Note that the target specification (x1,...,xn) in the prefix is redundant, if it is assumed that the target is specified by the set of free variables of P.

- a closed query (or yes-no query) is simply a DRC formula without free variables. The value of the query is the truth value (or Boolean value) of the formula, "true" (or "yes") or "false" (or "no").

The atomic formulas (or atomic predicates) of DRC are of two kinds:

- usual binary comparison predicates like =, ≠, <, >, etc., whose arguments are constants and domain variables. Constants are notations for domain elements. For each comparison predicate, both arguments must be associated with the same domain.

- 2^n-1 relation predicates are defined for each relation with n attributes. They have between 1 and n arguments and correspond to all possible ways of selecting a non-empty subset in the set of attributes of the relation. Each predicate is in fact a membership predicate for a projection of the relation and its arguments are constants or variables of DRC.

  The value of a predicate is "true" if the tuple made of its arguments belongs to the associated relation projection. It is "false" otherwise. For a relation R(A1:D1,...,An:Dn), the associated predicates are written R(Ai:ai,...,Aj:aj) where Ai,...,Aj are attributes of R (i.e., {Ai,...,Aj} is a subset of {A1,...,An}) and ai,...,aj are the arguments of the predicate (i.e., constants or variables). The "name" of the predicate can therefore be seen as "R(Ai: ,...,Aj:)" with the attributes Ai,...,Aj unordered just as in the associated relation. The position or arguments for the predicate is indicated by blanks. Thus:

  R(Ai:ai,...,Aj:aj) = "true" <=>

  <Ai:ai,...,Aj:aj> ∈ R[Ai,...,Aj]

The formulas of DRC are defined as follows:

- Atomic formulas are formulas.

- If A and B are formulas, so are (not A), (A or B), (A and B), (A->B), and (A<->B). Instead of a fully parenthesized form, usual priorities can be used for the connectives. The meaning of these propositional formulas is computed from the meaning of A and B using the usual rules of Boolean algebra.

- If x is a variable, then ∀x and ∃x are quantifiers containing x. If P(x) is a formula that contains x but no quantifier containing x, then (∀x P(x)) and (∃x P(x)) are formulas. Their meaning is defined as follows:

  ∀x P(x) ≡ P(a1) and ... and P(an)

  ∃x P(x) ≡ P(a1) or ... or P(an)

  where a1,...,an are the elements of the domain associated with x.

3. FORMAL DEFINITION OF RELATIONS

A relational database comprises a collection of finite domains of elementary values and a collection of relations.

A relation has a fixed name, a fixed structure, and a time-varying value.

The structure of a relation (sometimes called relation scheme) is specified by a set of attribute-domain pairs. All the attributes of a relation must be different. The domains referenced in a relation are not necessarily all different. The collection of attribute-domain pairs of a relation is not ordered.

Formally, the structure of a relation is a functional mapping from the attributes of the relation to domains (more precisely, domain names). This mapping is noted as a set of pairs {(A1->D1),...,(Ai->Di), ...,(An->Dn)}, where the Ai's are the attributes and the Di's are the domain names.

A relation R with n attribute-domain pairs is denoted as R(A1:D1,...,An:Dn). Sometimes, R(A:D) or R(A) will be used as a shorthand notation.

The value of a relation R(A1:D1,...,An:Dn) (sometimes called state of relation, or simply, relation) is a set of n-tuples <d1,...,dn> of elementary values, where di belongs to Di for all i such that $1 \leq i \leq n$. More exactly, the value of R is a set of "labeled n-tuples" <A1:d1,...,An:dn> of values. The order of values is not significant, since each value is associated with an attribute of R.

Formally, the value of a relation R(A1:D1,...,An:Dn) is a subset of the generalized Cartesian product of its domains D = {D1,...,Dn} indexed by its attributes A = {A1,...,An} defined as:

$$A1:D1 \times \ldots \times An:Dn =$$

$$\{t:A \rightarrow D1 \cup \ldots \cup Dn \mid t \text{ is total and } t(Ai) \in Di \text{ for } 1 \leq i \leq n\}$$

The indexed Cartesian product will also be noted X(A1:D1,...,An:Dn) or, for brevity, X(A:D).

The indexed Cartesian product generalizes the ordinary Cartesian product in that the component sets are unordered in the product and each set is distinguished from the others by a unique index. Thus, each relation tuple t of a relation R with attributes A is described formally as a total function on the set of attributes, having values in the associated domains.

The operations of the relational algebra [CODD72] can be defined precisely with this formal description of relation values [PIRO82]. For example, projection and Cartesian product, which are used in this paper, are defined as follows.

The projection of a tuple t on its B attributes is written t[B] and is the tuple t':

$$t':B \rightarrow DB \text{ such that } \forall b \in B\ t'(b) = t(b)$$

Similarly, the projection R[B] of a relation R(A), with $B \subseteq A$, on its B attributes is the set of all such tuples:

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

$$= \{t':B \rightarrow DB \mid \exists t \in R\ \forall b \in B\ t'(b) = t(b)\}$$

The Cartesian product of two relations R1(A:DA) and R2(B:DB) with A and B disjoint is a relation whose value is:

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

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

4. DENOTATIONAL SEMANTICS OF DRC

A denotational definition of a language essentially consists of a definition of the syntax of the language, and of the definition of functions which map syntactic constructions to mathematical objects representing the objects actually manipulated by the language.

4.1 Syntax of DRC

Denotational definitions start from an "abstract" syntactic definition, similar to descriptions with classical context-free (or BNF) grammars. The abstract syntax of a semantic definition is not necessarily the syntax used for syntactic analysis. The abstract syntax simply provides a way of manipulating the syntactically analyzed programs. It may well be ambiguous, if all the syntactic analyses of a program lead to the same semantic interpretation.

An abstract syntax for DRC is shown in Figure 1.

The syntactic categories (or nonterminals) are: V (variables or, more precisely, variable symbols), R (names of database relations used as predicate symbols), C (constants), Q (queries), F (formulas), T (terms), OP (operators like =, ≠, <, >, etc.). V, R, C, and OP are not further specified. They essentially behave like terminal symbols of usual BNF grammars. Rules 1, 3, 7, 8, and 9 are rule schemas for $n \geq 1$. Indexed occurrences of nonterminal symbols are used to represent distinct instances of the corresponding class of objects.

An abstract syntax is not necessarily a complete syntactic specification of a language. Thus, in the case of DRC, acceptable queries must also verify the following additional syntax rules, where numbers refer to rules in Figure 1:

```
(1)  Q  ::= {(V1,...,Vn) | F}
(2)      |  {F}
(3)  F  ::= R (A1:T1,..., An:Tn)
(4)      |  not F
(5)      |  F1 and F2
(6)      |  F1 or F2
(7)      |  ∀ V1,...,Vn F
(8)      |  ∀ V1,...,Vn F1 -> F2
(9)      |  ∃ V1,...,Vn F
(10)     |  (F)
(11)     |  V OP T
(12) T   ::= V
(13)     |  C
```

Figure 1. Syntax of DRC

(1) the free variables of F must be exactly the variables (V1,...,Vn); all variables in (V1,...,Vn) must be distinct; there must be at least one variable in (V1,...,Vn);

(2) F may not have free variables;

(3) each variable or constant used as an argument in a relation predicate must be associated with the same domain as the domain associated with its position as an argument in the predicate;

(5,6) either F1 and F2 have no free variables or both have free variables;

(7,9) the variables in (V1,...,Vn) must all be distinct; they must all appear as free variables of F; there must be at least one variable in (V1,...,Vn);

(8) the variables in (V1,...,Vn) must all be distinct; each must appear as a free variable of F1, F2 or both; there must be at least one variable in (V1,...,Vn);

(11) V and T must be associated with the same domain; the operation OP must be defined for the domain associated with V and T.

4.2 Semantic Functions

4.2.1 Relations Denoted by Formulas

DRC is typed. Each domain of the database defines a type for variables and constants. Subtypes are not supported and domains are supposed to be disjoint. Each variable is associated with one domain. Several mechanisms are possible to specify the association (such as an explicit prefixed declaration); they are equivalent for what concerns the semantic structure of the language, and, therefore, they are not discussed in this paper.

The central design principle of the semantic definition is to represent the value of each formula or predicate (F in the syntax) by a relation value whose attributes are the free variables of the formula. (We also say that a formula "denotes" a relation). Thus, if a formula F has free variables V1,...,Vk with types corresponding respectively to domains D1,...,Dk, then the denotation of F will be a relation which is a subset of the indexed Cartesian product

$$V1:D1 \times \ldots \times Vk:Dk$$

also noted X({V1,...,Vk}:D). That relation depends on F alone and not on the context in which F occurs in a query.

The same holds for "open" or "list" queries, whose value can be described as a relation.

Section 4.2.3 shows that the same formalism extends to the description of the value of "closed" or "yes-no" queries (and formulas) without free variables, which denote a truth value.

4.2.2 Semantics of DRC

The meaning of a query Q addressed to a database schema is a function from database instances of the schema to relations whose attributes are the target variables of the query:

$$mngQ : DB \to 2^{X(V:D)}$$

where V is the set of target variables of Q if Q is an open query, and the empty set if Q is a closed query, and D is the collection of domains defining the types of the variables in V. 2^X is the power set of X, that is, the set of all subsets of X. DB is the set of possible database instances of a database schema. Formally, if the schema has n relations R1,...,Rn with sets of attributes A1,...,An respectively, then DB has the form:

$$2^{X(A1:D)} \times \ldots \times 2^{X(An:D)}$$

A generic meaning function has the following form:

$$m : Q \to (DB \to Relations)$$

or more precisely

$$m : Q \to (DB \to 2^{X(V:D)})$$

To each query Q, it associates a function (mngQ with the notation above) from database instances to relations.

An equivalent function has the form:

$$m' : DB \to (Q \to 2^{X(V:D)})$$

To each database DB, it associates a function, that we will call mng, from queries to relations. The latter function is described in Figure 2. Thus, mng in Figure 2 is defined for a given database. This particular form of definition is chosen to simplify the notations. Making explicit the dependency on the database DB is immediate but complicates the notations. The database is referenced only through "rel(R)", which is the relation value of the database relation R (in rule 3), and through the domains D.

Numbers in Figure 2 refer to syntax rules in Figure 1. Rules 1 and 2 are in a sense superfluous: they define a special function mng' that presents the meaning of a list query as a relation and the meaning of a yes-no query as either "yes" or "no". Rules 3 to 11 describe the mng function. Only rules 3, 5, and 8 will be discussed in some detail. Rules 1 and 10 are obvious. Rule 2 is covered in section 4.2.3, which deals with truth-valued formulas. Rules 4 and 6 are similar to rule 5: rule 4 expresses negation as a set difference between an indexed Cartesian product of domains and the relation denoted by the formula on which negation bears; rule 6 expresses disjunction as "bordered" union. Rule 7 is a case of universal quantification simpler than rule 8; "$\forall x$" in "$\forall x$ F(x)" means "for all values in the domain associated with variable x". Rule 9 expresses the usual equivalence between existential quantification and projection. Rule

11 describes the value of a comparison as a subset of an indexed domain, or of an indexed Cartesian product of two domains.

Semantics of rule 3: f ::= R (A1:T1,...,An:Tn)

R is the name of a database relation, and rel(R) is its value in the database. {A1,...,An} is a subset of the attributes of rel(R). {T1,...,Tn} is a set of terms, that is, of variables or constants. V is the set of variables in {T1,...,Tn}. Variables serve as attributes of the value (a relation) of the formula. Constants denote elements of database domains. A function from constants to the union of domains is assumed. Here, to simplify, we have done as if constants were domain elements.

In algebraic terms, the relation denoted by R(A1:T1,...,An:Tn) is obtained by (1) a restriction of rel(R) corresponding to the constants in {T1,...,Tn}, followed by (2) a projection on the attributes associated with the variables V in {T1,...,Tn}, followed by (3) a renaming of attributes, where each remaining attribute Ai is replaced by the variable Ti (in V) with which it is associated.

In Figure 2, when a term Ti is a constant, the denotation of F involves a restriction of rel(R) to those tuples (formally: functions) where the value of attribute Ai equals Ti.

When Ti is a variable, say Vi, the denotation of F involves a renaming of Ai to Vi, which thus becomes an attribute of the relation meaning of F. When two (or possibly more) attributes Ai and Aj are associated with the same variable Vk, the operation is no longer a mere renaming of attributes. Instead, it becomes an "equi-restriction" of rel(R) to those tuples that have the same value for attributes Ai and Aj.

(1) $mng'(\{(V1,\ldots,Vn):F\}) = mng(F)$

(2) $mng'(F) = \text{"no"}$ if $mng(F) = \emptyset$
"yes" if $mng(F) = \{\emptyset\}$.

mng : F -> X(V:D)

(3) $mng(R(A1:T1,\ldots,An:Tn)) =$
$\{t \in X(V:D) \mid \exists t1 \in rel(R)\ \forall i$ such that $1 \le i \le n$
$t1(Ai) = Ti$ if Ti is a constant
$t1(Ai) = t(Ti)$ if Ti is a variable $\}$

(4) $mng(\text{not } F) = X(fr(F):D) - mng(F)$

(5) $mng(F1 \text{ and } F2) =$
$mng(F1) \times X((fr(F)-fr(F1)):D) \cap$
$mng(F2) \times X((fr(F)-fr(F2)):D)$

(6) $mng(F1 \text{ or } F2) =$
$mng(F1) \times X((fr(F)-fr(F1)):D) \cup$
$mng(F2) \times X((fr(F)-fr(F2)):D)$

(7) $mng(\forall V1,\ldots,Vn\ F) =$
$\{t \in X((fr(F)-\{V1,\ldots,Vn\}):D) \mid$
$\{t\} \times X(\{V1,\ldots,Vn\}:D) \subseteq mng(F)\}$.

(8) $mng(\forall V1,\ldots,Vn\ F1 \to F2) =$
$\{t \in X((fr(F1) \cup fr(F2)-\{V1,\ldots,Vn\}):D) \mid$
$\{s[\{V1,\ldots,Vn\}] \mid s \in mng(F1)$
and $s[fr(F1)-\{V1,\ldots,Vn\}]$
$= t[fr(F1)-\{V1,\ldots,Vn\}]\}$
$\subseteq \{r[\{V1,\ldots,Vn\}] \mid r \in mng(F2)$
and $r[fr(F2)-\{V1,\ldots,Vn\}]$
$= t[fr(F2)-\{V1,\ldots,Vn\}]\}\}$

(9) $mng(\exists V1,\ldots,Vn\ F) =$
$mng(F)[fr(F)-\{V1,\ldots,Vn\}]$

(10) $mng((F)) = mng(F)$

(11) $mng(V\ OP\ T) =$
$\{t \in X(V:D) \mid t(V)\ OP\ T\}$ if T is a constant
$\{t \in X(\{V,T\}:D) \mid t(V)\ OP\ t(T)\}$ if T is a variable

Figure 2. Denotational Semantics of DRC

Semantics of rule 5: F ::= F1 and F2

Let fr(F) be the set of variables occurring free in F. The attributes of the meaning of F are the variables in

$$fr(F) = fr(F1) \cup fr(F2).$$

It would be interesting that, as usual, conjunction correspond to set intersection. However, mng(F1) and mng(F2) cannot be intersected directly, since in general $fr(F1) \neq fr(F2)$ and the intersection is empty. What is needed is an interpretation of both F1 and F2 as sets of tuples in X(fr(F):D). Since F1, for example, does not constrain variables in fr(F)-fr(F1), the interpretation of F1 as a subset of X(fr(F):D) is obtained by completing or "bordering" each tuple of mng(F1) by all possible values for the attributes fr(F)-fr(F1). Thus, this interpretation amounts to interpreting both F1 and F2, in the context of the conjunction, as formulas with free variables fr(F).

In algebraic terms, the meaning of "F1 and F2" is thus the "bordered intersection" [PIRO82] of the meanings of F1 and F2. This operation has a number of special cases. It reduces to ordinary intersection if F1 and F2 have all their free variables in common. If F1 and F2 have no free variables in common, then the meaning of "F1 and F2" is equivalent to the Cartesian product of the meanings of F1 and F2. If F1 and F2 have some but not all of their free variables in common, then the meaning of "F1 and F2" is equivalent to the natural join of the meanings of F1 and F2. If F1 and F2 do not have free variables, then "F1 and F2" is the Boolean conjunction of truth values represented by $\{\emptyset\}$ (true) and $\emptyset$ (false).

Because of a well-formedness rule, either neither F1 nor F2 has free variables or both F1 and F2 have free variables.

Semantics of rule 8: F ::= ∀V1,...,Vn F1 -> F2

Rule 8 is a special case of rule 7 (∀ V1,...,Vn F1 -> F2 is equivalent to ∀ V1,...,Vn (not F1 or F2)), and its semantics can be deduced from the semantics of rule 7. The special form of quantification described by rule 8

is made available in DRC because it corresponds to the most frequent use of universal quantification.

If fr(F1) and fr(F2) are the free variables of F1 and F2 respectively, then the formula denotes a relation with attributes

$$fr(F1) \cup fr(F2) - \{V1,...,Vn\}$$

that is, in general, a relation whose tuples are constructed from values of the relations denoted by both F1 and F2.

If fr(F1), fr(F2), and {V1,...,Vn} each reduce to a single variable, then an example of quantification described by rule 8 is:

$$\forall x\ F1(x,y) \rightarrow F2(x,z)$$

In a set notation, the value denoted by that formula is the set of pairs (2-tuples):

$$\{\langle y,z\rangle |\ \{x|F1(x,y)\} \subseteq \{x|F2(x,z)\}\ \}$$

The usual division of [CODD72] describes the relation values of a special case of universal quantification, where $fr(F1) \subseteq \{V1,...,Vn\}$. In that case, if the relation denoted by F1 is not empty, then the relation denoted by the formula is made of projections of some tuples of F2, without contributions from F1.

This suggests a generalization of division, which corresponds to the form of universal quantification defined by rule 8. This new division operation contains the usual division of [CODD72] as a special case. It is defined in [PIRO82].

4.2.3 Boolean-valued Formulas

DRC can express both list queries, which denote a relation, and yes-no queries, whose value is a truth value. The informal definition of section 2 describes both kinds of queries. The same is true for the semantic equations of the preceding section, if degenerate relation values without attributes are interpreted as denoting truth values.

For an empty collection of domains and an empty set of attributes $\emptyset$, the definition of section 3 of the Cartesian product of the domains indexed by the attributes becomes:

$$\begin{aligned} X(\emptyset:\Lambda) &= \{t : \emptyset \rightarrow \cup\Lambda\} \\ &= \{t : \emptyset \rightarrow \emptyset\} \\ &= \{\emptyset\} \end{aligned}$$

In effect, $\{t : \emptyset \rightarrow \emptyset\}$ is the set of functions from $\emptyset$ to $\emptyset$. There is only one such function, the empty function (the empty set of pairs of values, if functions are viewed as sets of pairs).

Thus, there are two degenerate relation values of structure $(\emptyset:\Lambda)$:

$$\emptyset = \text{"0"}$$

$$\{\emptyset\} = \text{"1"}$$

which we interpret respectively as "false" and "true".

Applying the definition of section 3 for the Cartesian product of two relations, we obtain:

$$\text{"0"} \times \text{"0"} = \text{"0"}$$

$$\text{"0"} \times \text{"1"} = \text{"1"} \times \text{"0"} = \text{"0"}$$

$$\text{"1"} \times \text{"1"} = \text{"1"}$$

For the projection of a relation on an empty set of attributes:

$$R[\emptyset] = \emptyset \quad \text{if R is the empty set of n-tuples}$$

$$= \{\emptyset\} \quad \text{if R is not empty}$$

Similarly, the set-theoretic union, intersection, and difference of degenerate relations express respectively the Boolean disjunction, conjunction, and negation.

This is sufficient to cover all the cases of truth-valued formulas in DRC. Those cases are as follows: (1) relation predicates can produce a truth value from an ordinary relation; (2) so can universal and existential quantifications; (3) conjunction, disjunction, and negation in DRC do not produce truth values from ordinary relations: instead, when their operands are degenerate relations, their effect is to apply the usual Boolean operations with the same name on the truth values represented by the degenerate relations.

In summary, the semantic rules of Figure 4.2 extend to queries and formulas without free variables by interpreting the empty set $\emptyset$ as "false" and $\{\emptyset\}$ as "true". This result increases our confidence in the adequacy of the semantic objects and operations chosen to define DRC. We do not suggest however that truth-valued formulas be presented to users as denoting degenerate relations.

5. DRC, RELATIONAL ALGEBRA AND LOGIC

The definition of DRC in the preceding section establishes the following connections between logical connectives and a version of algebraic operations:

negation	: complement
conjunction	: bordered intersection
disjunction	: bordered union
universal quantification with implication	: division
existential quantification	: projection
predicate	: projection, equi-restriction

The denotational form of the definitions suggests interesting generalizations of some algebraic operations. Thus operations called "bordered union", "bordered intersection", and also a generalization of division have been defined. A complete definition of a relational algebra with the new operations is given in [PIRO82].

Bordered intersection contains as special cases natural join, ordinary intersection and Cartesian product, as they are defined e.g., in [CODD72]. Bordered union and the generalized division contain respectively ordinary union and division, but they are more general. For example, all cases of universal quantification of the calculus can be expressed as a generalized division.

The generalized intersection and generalized union of [HALL75] are similar the corresponding bordered operations.

The version of algebraic operations sketched in this section (and defined in [PIRO82]) and the definition of DRC in the preceding section show in a striking manner the fundamental unity of relational algebra and calculus. In that respect, the present work has similarities with that described in [MERR78].

DRC has the same syntax as the first order predicate logic, and the semantics have a similar structure. Many equivalence rules from logic are preserved in DRC. For example:

mng(F1 and F2) = mng(not((not F1) or (not F2)))

or

mng(∀V1,...,Vn F) = mng(not (∃V1,...,Vn (not F)))

However, it is not true in general that if logic allows to deduce F1<->F2 for formulas F1 and F2, then mng(F1)=mng(F2) in DRC. For example, in logic

F1 or (F1 and F2) <-> F1

but, in DRC

mng(F1 or (F1 and F2)) ≠ mng(F1)

if fr(F1) ≠ fr(F2).

6. SUMMARY AND CONCLUSIONS

6.1 Language Description

The work reported in this paper is an experiment with a precise (nearly formal) method for specifying the semantics of relational query languages. The language chosen, DRC, is a version of domain relational calculus. It has the same power of expression as the relational algebra. This experiment produced several interesting technical results.

First, we gave a purely "bottom-up" denotational specification of the semantics of DRC, based on the single principle that every formula (or predicate) of DRC denotes a relation. This is an interesting result in itself, as it was not obvious initially that such a definition was possible. The definition is precise, short, and systematic.

Second, the denotational definition of DRC formulas involving logical connectives suggested interesting generalizations of operations of the relational algebra. The main result of this paper is to exhibit a version of relational calculus and a version of relational algebra which correspond very directly to one another.

Third, a formal definition is given for relations with unordered attributes where domains determine the comparability of elementary values. Algebraic operations operate on and produce relations thus defined. The denotational definition of the semantics of DRC is such that the same syntactic and semantic rules describe the meaning of both "list queries", whose value is a relation, and of "yes-no" queries, whose value is a truth value.

A precise semantic definition enables fine analyses of the structure of query languages. Thus, several continuations of the present work have been or are being investigated. One of them investigates denotational definitions of other query languages. For example, we have already established that the application to relations of a fairly general version of aggregate functions can be described with basically the same formalism as the one used in this paper. This will be reported in another paper. Another continuation of this work consists in further theoretical investigations of the relational algebra suggested by DRC, and of the relationships of DRC with predicate logic. An interesting result will be to characterize precisely the equivalence rules of the predicate calculus that are not preserved in DRC. Another interesting subject has been the characterization in query languages of anomalies linked to the unrestricted use of negation, universal quantification, or disjunction (see e.g. [DEMO82, PIRO76, ULLM80]). The association of types with the variables of DRC automatically solves the most serious problems linked to negation and quantification in languages without types. In addition, the regular semantic structure of DRC enables a fine analysis of "sensible" uses of negation, disjunction, and conjunction. Results will be reported elsewhere.

6.2 Language Design

This work also suggests that judicious semantic decisions made early in the design process of a query language can simplify the specification of the language, and eventually, the language itself.

We clearly realized in writing the denotational definition of this paper that the central idea in the original design of DRC [LACR77] was more than anything else a decision about a uniform semantics for its constructs. The terse and precise definition produced for DRC can be relatively easily translated into ordinary language, while essentially preserving its terseness and precision.

More generally, this work suggests a "semantics-directed" method of language design instead of what seems to be the typical "syntax-directed" strategies of conventional query language design. Thus, for example, the history of the design of the SQUARE, SEQUEL and SQL languages [CHAM76] could be summarized as investigations of how much functionality can be expressed with the basic syntax of the "select block". The successive versions of the languages

describe attempts to accommodate the limitations of that syntax, for example, the fact that only a few patterns of universal quantification or of calls to aggregate functions fit in a straightforward way into that syntax mold. Similarly, the successive versions of Query-By-Example [ZLOO75] describe how much can be done with the basic syntactic idea of filling in examples in a table. For the simplest patterns of query (equivalent, say, to projection and restriction in the algebra), that syntax nicely expresses the intended semantics. But, this is much less true when quantifiers or negation are involved, and not at all when a "condition box" must be introduced to express comparisons of values.

We find it interesting that no precise definition of the semantics of SQL or Query-By-Example exists, which would be short and would somehow reflect the impression of user-friendliness that an initial contact with those languages communicates. We conjecture that a semantic definition with a few primitive operations like that of this paper is not possible for SQL or Query-By-Example. We believe that precise definitions would be large and complex, and that they would probably suggest a redesign of parts of those languages.

By contrast, a semantics-directed method of language design specifies the basic semantics and the exact limits of validity and of legal utilization of a construct before (or, at least, at the same time as) its syntactic appearance in queries. This strategy has the advantage that, whatever design decisions are made eventually, it guarantees that a precise definition of the syntax and semantics of the language is manageable, that is, that it is of reasonable size and involves semantic objects and operations chosen by the language designers and manageable, at least for them.

In other words, we believe that if the designers of a language give a precise (maybe formal) semantic definition of their language, then the chances of having a "good" design are increased, where precise semantic specifications are not hopelessly complex, and match intuitive perceptions of language constructs by users. This is to be contrasted with situations where a precise semantic definition is done, by implementors or formal language specialists, after the "design" phase. It is interesting to note that some programming language designers reached similar conclusions and expressed similar recommendations [ASHC82,LOND78].

As an example of semantics-directed design, the design of aggregate function calls could be integrated in the definition of DRC of this paper as follows. First, the decision is made that a new construct is made available to express function application. It expresses the application of a function to a relation, and the repetition of function applications to classes of a horizontal partition of a relation. It returns a relation (possibly reduced to a value) as a result. Then, a choice is made of the particular functions that are to be made available, and a precise semantic definition is specified for calls to each of them, including computation of the result, duplicate control, repeated applications, etc. Then only, syntactic decisions have to be made, in the best case on the basis of human factor studies, about the exact form or forms of the new operation.

Note that we do not advocate semantic formalisms for the sake of using formalisms. We believe that any formal definition is not automatically interesting, and that simplicity and economy of concepts (although we don't suggest that they are easily measurable) are equally important.

7. ACKNOWLEDGEMENTS

We are grateful for valuable comments made by Frank Manola, Michel Sintzoff, and Robert Demolombe about previous versions of this paper.

8. REFERENCES

[ASHC82]
Ashcroft, E.A., and Wadge, W.W., "R for Semantics," ACM Trans. on Programming Languages and Systems, Vol. 4, No. 2, April 1982.

[CHAM76]
Chamberlin, D.D., et al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control," IBM Journal of Research and Development, November 1976.

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

[DATE81]
Date, C.J., An Introduction to Database Systems, Third Edition, Addison-Wesley, 1981.

[DEMO82]
Demolombe, R., "Utilisation du Calcul des Prédicats comme Langage d'Interrogation des Bases de Données," Thèse de Doctorat d'Etat, Univ. Toulouse, February 1982.

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

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

[LACR77]
Lacroix, M. and A. Pirotte, "Domain Oriented Relational Languages," Proc. VLDB Conference, Tokyo, 1977.

[LOND78]
London, R.L., et al., "Proof Rules for the Programming Language Euclid," Acta Informatica, Vol. 10, Fasc. 1, 1978.

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

[PIRO76]
Pirotte, A., "Explicit Description of Entities and their Manipulation in Languages for the Relational Data Base Model," Doctoral Thesis, Univ. Bruxelles, December 1976.

[PIRO82]
Pirotte, A., "A Precise Definition of Basic Relational Notions and of the Relational Algebra," to appear in ACM SIGMOD Record, 1982.

[STOY77]
Stoy, J., Denotational Semantics : The Scott-Strachey Approach to Programming Language Theory, The MIT Press, 1977.

[ULLM80]
Ullman, J.D., Principles of Database Systems, Computer Science Press, 1980.

[ZLOO75]
Zloof, M.M., "Query-By-Example," AFIPS Conference Proceedings, Vol.44, 1975.