EQUIVALENCE OF RELATIONAL ALGEBRA AND RELATIONAL CALCULUS QUERY LANGUAGES HAVING AGGREGATE FUNCTIONS

by

Anthony Klug

Computer Sciences Technical Report #389

June 1980

Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions

Anthony Klug

University of Wisconsin

Abstract

Aggregate functions in relational query languages allow intricate reports to be written. In this paper aggregate functions are precisely defined. The definition does not use the notion of "duplicates". Relational algebra and relational calculus are extended in a general and natural fashion to include aggregate functions. It is shown that the languages so extended have equivalent expressive power.

Keywords and Phrases: aggregate function, relational algebra, relational calculus, query equivalent, relational model

CR Categories: 4.33, 5.21

Author's address: Anthony Klug, Computer Sciences Department, University of Wisconsin-Madison, Madison, WI 53706
This work was supported in part through contract NB79SBCA0191 with the National Bureau of Standards.

Contents

some symbols used in this paper

symbol	name	meaning
$\mathbb{U}$	English U	set union, union operator
$\mathbb{v}$	English v	logical OR
$\mathbb{\sqcap}$	intersect	set intersection
$\mathbb{\exists}$	backwards E	existential quantifier
$\mathbb{N}$	bold N	natural numbers
$\mathbb{V}$	bold V	set of variables
$\mathbb{T}$	bold T	set of terms
$\mathbb{F}$	bold F	set of formulas
$\mathbb{A}$	bold A	set of alpha expressions
$\mathbb{E}$	bold E	set of algebra expressions
$\mathbb{I}$	bold I	set of instances
$\mathbb{\epsilon}$	small epsilon	set membership
$\mathbb{\alpha}$	small alpha	calculus expressions
$\mathbb{\psi}$	small psi	a formula
$\mathbb{\pi}$	small pi	a formula
$\mathbb{\rho}$	small rho	a restriction or join
$\mathbb{\sigma}$	small sigma	a selection
$\mathbb{\times}$	cross	Cartesian product
$\mathbb{\equiv}$	four bars	equi-selection
$\mathbb{\equiv}$	three bars	equi-restriction
$\mathbb{\{\ \}}$	bold { }	restriction/selection operator

1. Introduction

Report writing, generating what is colloquially known as "IBM printouts", is an important function of database systems. A report is generated by the application of aggregate or statistical functions such as sum, average, minimum, etc. to data files or database relations. While relational database theory has provided a sound mathematical basis for studying many database problems, the use of aggregate functions in relational query languages is not well understood. Precise and general definitions are lacking, and their embedding into query languages is not well defined.

1.1. An Example

Consider the relational schema of Figure 1 describing a university database. A typical query, in English, involving aggregate functions would be:

> For every department in the Letters & Science college, compute the total grad student support for each of the department's faculty members and print the department name and the average support for the department.

```
dept(name, head, college)
faculty(name, dname)
grad(name, majorprof, grantamt)
```

Figure 1. Example Schema

1.2. Purpose of Paper

The purposes of this paper are:

- We give precise definitions for aggregate functions.
- We extend relational calculus to include aggregate functions in a natural manner.
- We extend relational algebra to include aggregate functions in a natural manner.
- We prove that the algebra and calculus so extended have the same expressive power.

There are several reasons the above work is needed:

As stated, report writers are an important part of real-life database applications. A theoretical foundation is needed for these systems. With such a foundation, system specifications can be made much more precise, and new languages can more easily be defined.

Previous treatments of aggregate functions in relational languages have not been general and have not been well defined. Two examples are System-R ([ABCE], [CAEG], [ChBo]) and Ingres ([HeSW], [SWKH]). These formulations do not apply to more general languages, for example, to languages having explicit quantifiers. Their definitions of aggregate functions also unnecessarily rely on sets of tuples having duplicate members (a contradiction). This goes completely outside the set-theoretic definition of the relational model. A precise definition is therefore needed which does

not use the notion of "duplicate". In this paper we give definitions of aggregate functions for the calculus and the algebra which are natural, general and precise.

Having extended the calculus and algebra to include aggregate functions, it is important to know if they have equivalent expressive power. This will allow designers to base new languages on either the calculus or the algebra without fear of losing expressive power.

The proof by Codd [Codd72a] that relational calculus and relational algebra (without aggregate functions) are equivalent is incorrect. (See Section 4.) The literature should record a correct proof.

1.3. Outline of Paper

In the next section we formally define relations, aggregate functions, relational algebra and relational calculus. Some comparisons between the two languages are made. In Section 3 an algorithm is given showing that every algebra query can be expressed as a calculus expression. In Section 4 we show that every calculus query has an equivalent algebra expression. Finally, in Section 5 we provide a summary and some directions for future work.

2. Formal Definitions

The relational model is first defined. We follow Codd ([Codd70], [Codd72a], [Codd72b]) in treating all data domains as integers and in referring to attributes by column number. This simplifies the treatment although some arithmetic computations on column numbers is then necessary.

2.1. The Relational Model

A relation scheme is a pair $\langle R,k\rangle$. R is a symbol (the relation name), and k is a positive integer (R's degree) which is denoted deg(R). If $\langle R,k\rangle$ is a relation scheme, the domains of R, doms(R), is the set $\{1,2,\ldots,k\}$ of natural numbers.

A schema is a sequence $\langle\langle R_1,k_1\rangle,\ldots,\langle R_s,k_s\rangle\rangle$ of relation schemes. It is generally written simply $\langle R_1,\ldots,R_s\rangle$. Throughout this paper, one fixed schema $\langle R_1,\ldots,R_s\rangle$ is assumed.

An instance I of schema $\langle R_1,\ldots,R_s\rangle$ is a sequence $\langle I_1,\ldots,I_s\rangle$ where for each $i=1,\ldots,s$, $I_i \subseteq \mathbb{N}^{deg(R_i)}$. All domains are taken without loss of generality to range over the set $\mathbb{N}$ of natural numbers, and $\mathbb{N}^m$ is the set of all m-tuples over $\mathbb{N}$. We let I be the set of all instances over our fixed schema of s relations.

2.2. Aggregate Functions

The concept of an aggregate function is quite simple. An aggregate function takes a set of tuples as an argument and produces a single simple value as a result.

Both SQL ([ABCE], [CAEG], [ChBo]) and QUEL ([HeSW], [SWKH]), the query languages of System-R and Ingres, respectively, require that aggregate functions be able to accept arguments with duplicates. For example, to sum the salaries in an employee relation, the relation would be projected on the salary column, duplicates would be retained, and the projection would be sent to the sum function. Besides being unnecessary as we will see, the notion of "duplicates" is not well defined[1]. For example, the number of duplicate tuples in an expression:

$$R[X] \cup S[Y]$$

could depend on the order in which the system choses to perform the projections and union.

Instead of providing one sum function (or average, max etc.), we provide a family of sum functions:

$$sum_1, sum_2, sum_3, \ldots, sum_i, \ldots$$

The function sum_i sums the numbers in the i-th column of its

[1] Although bags have been formally defined [RoLe], they do not solve the problem.

input. Now there is no need for the vague notion of "duplicates". For example to determine the total of column 3 (salaries) in the tuple set:

R =

Joe	25	12000
Nancy	21	13000
Sue	28	13000
Pete	35	14000
John	30	12000

we would write $sum_3(R)$, not sum'(R[3]), where sum' is somehow supposed to tell its input not to be a set but a bag!

Formally, we hypothesize a countable set

$$F = \{f_1, f_2, f_3, \ldots\}$$

of aggregate function symbols. For each $f_i \in F$, the meaning of f_i is a function

$$\overline{f}_i : U_i \, Fin(\mathbb{N}^i) \rightarrow \mathbb{N}.$$

(U_i denotes the union over $i \in \mathbb{N}$; Fin denotes the finite subset operator.) That is, it is a function whose range is the natural numbers (our universe) and whose domain is the set of all finite homogeneous tuple sets[2].

[2] We do not bother to define the meaning of a function symbol for each database instance as is the case for function symbols in logic; all we need is one standard interpretation for each symbol.

We also do not consider details of which kinds of

The set F cannot be completely arbitrary. It must have the following uniformness property: For every $f \in F$ and tuple set S, if S' is a "constant expansion" of S, i.e., if there is a projection X such that S'[X] = S and S[~X] contains just one tuple, then there is an $f' \in F$ such that f(S) = f'(S'). S and S' are always isomorphic, and we will not even distinguish between f and f'. All normal sets of aggregate functions have this property. A more concise statement of this property will be given when we have defined the relational algebra.

2.3. Relational Algebra

The set $\mathbb{E}$ of relational algebra expressions over our fixed schema and the associated functions deg (degree) and doms (domains) for expressions are defined inductively as follows:

If $e \in \mathbb{E}$ has degree k, then doms(e) = {1,...,k}.

(Ø) Literals: For any constant $c \in \mathbb{N}$, $\{c\} \in \mathbb{E}$ and has degree 1.

(1) Base Relations: $R_i \in \mathbb{E}$ for each R_i in the schema, and $deg(R_i)$ is as defined in the schema.

(2) Projection: If $e \in \mathbb{E}$ and deg(e) is k, then $e[X] \in \mathbb{E}$ where X is a sublist of doms(e), and deg(e[X]) = number of elements in X.

(3) Cross Product: If $e_1, e_2 \in \mathbb{E}$ and $deg(e_1) = d_1$, $deg(e_2)$

---

tuple sets aggregate functions might not be defined on.

$= d_2$, then $(e_1 \times e_2) \in \mathbb{E}$, and $deg(e_1 \times e_2) = d_1 + d_2$.

(4) Restriction: If $e \in \mathbb{E}$ and $X,Y \in doms(e)$, then $e\{\!|X\theta Y|\!\} \in \mathbb{E}$, where θ is $\equiv$, $>$ or $<$, and $deg(e\{\!|X\theta Y|\!\}) = deg(e)$.

(5) Union, Difference: If e_1 and e_2 are in $\mathbb{E}$ and both have degree n, then $e_1 \cup e_2$ and $e_1 - e_2$ are in $\mathbb{E}$ and have degree n.

(6) Aggregate Formation: If $e \in \mathbb{E}$ and X is a sublist of doms(e), then $e\langle X,f\rangle \in \mathbb{E}$, and has degree len(X) + 1.

Other traditional operators can be defined in terms of the above operators. The most common are the following:

(7) Selection: If $e \in \mathbb{E}$, $V \in \mathbb{N}$, and $X \in doms(e)$, then $e\{\!|X\theta'V'|\!\}$ is $(e \times \{V\})\{\!|X\theta k|\!\}[doms(e)]$, where $k = deg(e)+1$. We will write $e\{\!|X\equiv'V'|\!\}$ as $e\{\!|X\equiv V|\!\}$.

(8) Restriction Lists: If $e \in \mathbb{E}$ and $X_1,\ldots,X_k,Y_1,\ldots,Y_k$ are in doms(e), then $e\{\!|X_1,\ldots,X_k \equiv Y_1,\ldots,Y_k|\!\}$ is $e\{\!|X_1\equiv Y_1|\!\}\{\!|X_2\equiv Y_2|\!\}\ldots\{\!|X_k\equiv Y_k|\!\}$.

(9) Join: If e_1, e_2 are in $\mathbb{E}$, X is a domain list of e_1 of length k, and Y is a domain list of e_2 of length k, then $e_1\{\!|X\theta Y|\!\}e_2$ is $(e_1 \times e_2)\{\!|X\theta Y'|\!\}$, where Y' is $Y+deg(e_1)$.

(10) Intersection: If e_1, e_2 are in $\mathbb{E}$ and have the same degree, then $e_1 \sqcap e_2$ is $(e_1 \times e_2)\{\!|D_1\equiv D_2|\!\}[D_1]$, where D_1 is $1,\ldots,deg(e_1)$ and D_2 is $1+deg(e_1),\ldots,deg(e_2)+deg(e_1)$.

(11) Division: If e_1, e_2 are in $\mathbb{E}$, X is a domain list of e_1 of length k, and Y is a domain list of e_2 of length k, then $e_1[X \div Y]e_2$ is $e_1[\tilde{}X] - ((e[\tilde{}X] \times e_2[Y]) - e_1)[\tilde{}X]$, where $\tilde{}X$ are the domains of e_1 not in X.

For each $e \in \mathbb{E}$ of degree k and for each $I \in \mathbb{I}$, the value of e on I, denoted e(I), is a subset of $\mathbb{N}^k$. The formal definition is as follows:

(Ø) $\{c\}(I) = \{c\}$.

(1) $R_i(I) = I_i$.

(2) $e[X](I) = \{t : \exists t' \in e(I)\ t'[X]=t\}$, where $t'[X]$ is the tuple whose i-th component is the j-th component of t', where j is the i-th element of X.

(3) $(e_1 \times e_2)(I) = \{t_1 \hat{} t_2 : t_1 \in e_1(I) \ \& \ t_2 \in e_2(I)\}$, where '^' denotes concatenation.

(4) $e\{X \theta Y\}(I) = \{t : t \in e(I) \ \& \ t[X] \theta t[Y]\}$.

(5) $(e_1 \cup e_2)(I) = \{t : t \in e_1(I) \vee t \in e_2(I)\}$

$(e_1 - e_2)(I) = \{t : t \in e_1(I) \ \& \ t \notin e_2(I)\}$.

(6) $e\langle X,f\rangle(I) = \{t[X] \hat{} y : t \in e(I) \ \& \ y = f(\{t' : t' \in e(I) \ \& \ t'[X]=t[X]\})\}$.

2.4. Relational Calculus

Six classes of objects are used to define calculus expressions: variables, terms, formulas, range formulas, alpha expressions (called simply alphas) and closed alpha expressions.

The set $\mathbb{V}$ of variables is $\{v_1, v_2, v_3, \ldots\}$.

The set $\mathbb{T}$ of terms is defined as follows: Every element of $\mathbb{N}$ is in $\mathbb{T}$. (These are the constants.) For every variable v_i and column number A, $v_i[A]$ is a term. If α is an alpha,

and $f \in F$, then $f(\alpha)$ is a term.

The set $\mathbb{F}$ of formulas is defined as follows: If α is an alpha and $v_i \in \mathbb{V}$, then $\alpha(v_i) \in \mathbb{F}$. If t_1, $t_2 \in \mathbb{T}$, and θ is '$\equiv$'[3], If ψ, $\pi \in \mathbb{F}$, then so are $\sim\psi$ and $(\psi \vee \pi)$. If $\psi \in \mathbb{F}$, $v_i \in \mathbb{V}$ and r is a range formula, then $(\exists r v_i)\psi \in \mathbb{F}$[4].

The set R$\mathbb{F}$ of range formulas is defined as follows: If $\alpha_1,\ldots,\alpha_k$ are closed (defined below) alphas, and $v_i \in \mathbb{V}$, then

$$\alpha_1(v_i) \vee \ldots \vee \alpha_k(v_i)$$

in in R$\mathbb{F}$ and is called a range formula for v_i. If r is a range formula for v_i we will often write $r(v_i)$.

The set $\mathbb{A}$ of alphas is defined as follows: If R_i is in the schema, then R_i is an alpha of degree $deg(R_i)$. If $t_1,\ldots,t_n \in \mathbb{T}$, if $r_1,\ldots,r_m$ are range formulas for $v_{i_1},\ldots,v_{i_m}$, if the free variables (defined below) of $t_1,\ldots,t_n$ are $v_{i_1},\ldots,v_{i_m}$, and if $\psi \in \mathbb{F}$, then

$$(t_1,\ldots,t_n) : r_1,\ldots,r_m : \psi$$

is an alpha of degree n.

---

[3] We have tried in this paper to keep multiple usages of symbols to a minimum. Thus '$\equiv$' denotes the formal equality symbol, while '=' denotes real set equality. Some symbols such as 'U' still serve double purposes.

[4] For completeness, we only need to have $\exists$-quantified variables coupled to simple alpha expressions, but allowing range formulas makes the proofs less cumbersome.

Free variables, bound variables and closed objects (terms, formulas, alphas) are now defined. An occurrence of a variable v_i is <u>free</u> if it is not within the scope of a v_i-quantifier. Otherwise, the occurrence is bound. A v_i-<u>quantifier</u> is a fragment $(\exists r v_i)$ of a formula or the range fragment $r_1,\ldots,r_j(v_i),\ldots,r_m$ of an alpha. In the following, X and Y denote (possibly empty) strings. The <u>scope</u> of $(\exists r v_i)$ in $X(\exists r v_i)\Psi Y$ is Ψ. The <u>scope</u> of $r_1,\ldots,r_m$ in $X(t_1,\ldots,t_n):r_1,\ldots,r_m:\Psi Y$ is $(t_1,\ldots,t_n)$ and Ψ. An object (term, formula, alpha) is <u>closed</u> if it has no free occurrences of any variable.

2.5. Interpretations of Alphas

In order to define the value of an alpha on an instance, we must, as in Predicate Calculus, define valuations which give values to free variables. Since a variable may occur in an alpha in several scopes with different ranges, it simplifies matters to rename variables so that each one occurs in only one scope. That this can be done follows from the same arguments as for the analogous property in Predicate Calculus ([Shoe], [BeSl]). We will assume this renaming has been done.

If α is atomic, say R_i, then no valuation is needed and we simply define $\alpha(I)$ to be I_i.

Let α be an alpha with variables $v_1,\ldots,v_n$ whose respective ranges are $r_1,\ldots,r_n$. Let $I \in \mathbb{I}$. If r_i has the form

$$\alpha_1(v_i) \vee \ldots \vee \alpha_k(v_i),$$

then the range of v_i in I is $r_i(I) = \alpha_1(I) \cup \ldots \cup \alpha_k(I)$ which is defined by induction. A valuation for α on I is a partial function

$$x : \{1,\ldots,n\} \to \cup_j r_j(I)$$

(a sequence with "holes") such that for each i, $x_i \in r_i(I)$. Given a valuation x, x[i/a] denotes the valuation which is identical to x except that x_i is always defined and $x_i = a$.

Given instance I and valuation x for α on I, we define interpretations of the terms, formulas and alphas in α, and α itself as follows:

Interpretations of terms: If $c \in \mathbb{N}$ is a constant, $c(I,x) = c$ (itself). If x is defined on v_i (on i to be precise), then $v_i[A](I,x)$ is defined and equals $x_i[A]$ (the A-th component of x_i). If $\alpha(I,x)$ is defined, then $f(\alpha)(I,x)$ is defined and equals $\overline{f}(\alpha(I,x))$.

Interpretations of formulas: For an atomic formula of the form $\alpha'(v_i)$, we have $\alpha'(I)$ defined by induction. Then $\alpha'(v_i)(I,x)$ is defined if x_i is, and $\alpha'(v_i)(I,x) = 1$ if $x_i \in \alpha'(I)$ and is otherwise $\emptyset$. For terms t_1 and t_2, if $t_1(I,x)$ and $t_2(I,x)$ are defined, then $(t_1 \Theta t_2)(I,x)$ is defined and equals 1 if $t_1(I,x)$ is Θ to $t_2(I,x)$, and otherwise equals $\emptyset$. If $\Psi(I,x)$ is defined, then $(\sim\Psi)(I,x)$ is defined and equals 1 if $\Psi(I,x)=\emptyset$, and otherwise equals $\emptyset$.

If $\psi(I,x)$ and $\pi(I,x)$ are defined, then $(\psi \vee \pi)(I,x)$ is defined and equals the maximum of $\psi(I,x)$ and $\pi(I,x)$. If for some $a \in r_i(I)$, $\psi(I,x[i/a])$ is defined and equals 1, then $(\exists r_i v_i)\psi(I,x)$ is defined and equals 1. Otherwise $(\exists r_i v_i)\psi(I,x)$ equals $\emptyset$.

Interpretations of alphas: $R_i(I,x)$ is always defined and equals I_i. If α' is $(t_1,\ldots,t_n):r_{j_1},\ldots,r_{j_m}:\psi$, then $\alpha'(I,x)$ is defined and empty if some $r_{j_i}(I)$ is empty. Otherwise it is defined and equal to the set of all tuples $(t_1(I,x'),\ldots,t_n(I,x'))$ such that x' is a valuation of the form $x[j_1/a_1][j_2/a_2]\ldots[j_m/a_m]$ for some $a_i \in r_i(I)$, and such that each $t_i(I,x')$ is defined and $\psi(I,x')$ equals 1.

As in Predicate Calculus, it is easy to show that for any alpha α, the value of $\alpha(I,x)$ depends only on the components of x corresponding to free variables of α. In particular, if α is closed, $\alpha(I,x)$ is always defined and is independent of x, and in this case we simply write $\alpha(I)$.

2.6. Comments and Comparisons

In this section we illustrate the different features of the algebra and the calculus.

The syntactic (and semantic) structures of the two languages are quite different. The algebra has one syntactic class: expressions. The calculus has six: variables, terms, formulas, range formulas, alphas and closed alphas, the last

being the only one corresponding to expressions. In Figure 2 we depict the syntactic structures of the algebra and the calculus where arrows → to denote the relation "is used to define". Any equivalence proof must reconcile these very different structures.

Among the tradeoffs in using the two languages are the following two: The algebra has a much simpler structure. This could be useful in theoretical work where, for example, inductive proofs are used. On the other hand, the calculus allows queries to be expressed more naturally (see below), and this may be useful, not only for end-user purposes, but also to make language specifications by translations to the calculus simpler.

Some examples are now given

Given a student relation scheme:

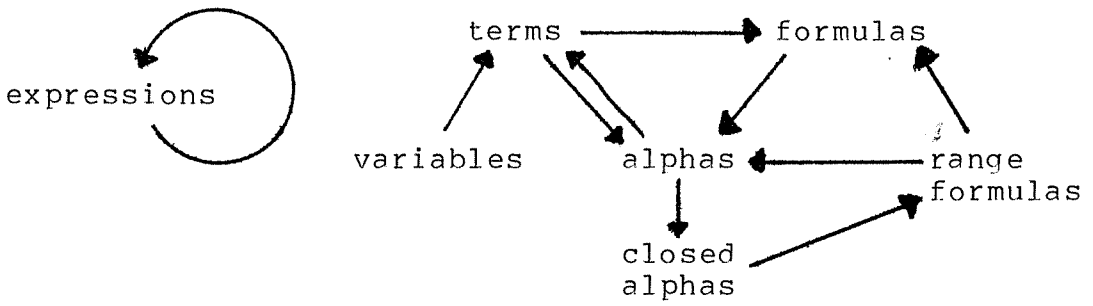

Figure 2. Recursive Structure.

student(name, yr, gpa),

we want to know, for each student year (1,2,3,4), the average gpa for students in that year. In the calculus we can write:

$$(v_1[2],\ ave_3((v_2[1],v_2[2],v_2[3]):student(v_2):v_2[2]=v_1[2])) : student(v_1) : -$$

In the algebra we could write:

$$student\langle 2,ave_3\rangle$$

Now suppose we wanted for each student year the average of students in that year or in a greater year. In the calculus one change to the previous query will suffice:

$$(v_1[2],\ ave_3((v_2[1],v_2[2],v_2[3]):student(v_2):v_2[2]>v_1[2])) : student(v_1) : -$$

However, in the algebra, aggregate functions are applied only to "equi-partitions"; there is no such thing as a "greater-than-partition"[5]. The algebra expression must first greater-than-join student with itself before applying the aggregate function:

$$(student\{2>2\}student)\langle 5,ave_3\rangle$$

[5] One, of course, could be defined, but the resulting set of operators would not be independent.

Next consider the example query in Section 1:

> For every department in the Letters & Science college, compute the total grad student support for each of the department's faculty members and print the average for the department.

In the calculus we can write:

$$(v1[1],\ ave_2(v_2[1],\ sum_2(v_3[1],\ v_3[3] : grad(v_3) : v_3[2]=v_2[1]) : faculty(v_2) : v_2[2]=v_1[1])) : dept(v_1) : v_1[3]='L\&S'$$

It would seem that to express this query in the algebra, we would only have to partition the grad relation by fname and sum grantamt, join this with the faculty relation, partition the result by the dept column and average the sum column. Finally, we would join with the dept relation, do the college selection and project out name and the average. The actual expression would be:

$$((((grad\langle 2,sum_3\rangle)\{1\equiv 1\}faculty)\langle 4,ave_2\rangle)\{1\equiv 1\}(dept\{3\equiv 'L\&S'\}))[1,2]$$

This is incorrect, however, because faculty with no students will not be counted in the average whereas they should appear with a zero sum. To properly express this query, and any similar query, we must generate sum tuples with zeros for those faculty having no students. To correct the above

query we should replace grad<2,sum_3> by the expression:

$$grad\langle 2,sum_3\rangle \cup (faculty[1] - grad[2]) \times \{\emptyset\}.$$

Range formulas in Codd's definition of the calculus were combinations of base relations. In the calculus with aggregate functions, we need range formulas built from combinations of alpha expressions. That this is necessary can be seen by considering the following algebra expression:

$$R\langle X,f\rangle \cup S\langle X,g\rangle$$

Without our general range clause, there is no way to write a calculus expression which can generate a set of tuples whose second column is sometimes an f-value and sometimes a g-value.

Literal relations are needed in the algebra because the calculus can put in the target list constants which actually occur nowhere in an instance.

The uniformness property for the set of aggregate functions can be expressed as follows: Define an aggregate formation operator of three variables by:

$$e\langle X,Z,f\rangle(I) = \{t[X]\hat{}y : t \in e(I) \;\&\; y = f(\{t'[Z] : t' \in e(I) \;\&\; t'[X]=t[X]\})\}.$$

This operator partitions, projects and then applies the function. The set F has the uniformness property if for

every e, X, Z and f there is an f' such that

$$e\langle X,Z,f\rangle(I) = e[X \cup Z]\langle X,f'\rangle(I), \text{ for all } I.$$

The three argument aggregate formation operator closely corresponds to applying an aggregate function to a calculus expression. The Z represents the domains in the target list of the alpha.

3. Translating Algebra to Calculus

In this section, we want show that for every algebra expression $e \in \mathbb{E}$ there is an alpha α with $e(I) = \alpha(I)$ for all $I \in \mathbb{I}$. This is done recursively as follows:

(Ø) A literal {c} corresponds to the alpha (c):-:- (empty range and predicate).

(1) For a schema relation R_i, the corresponding alpha is simply R_i.

For the next three steps, assume e has a corresponding alpha of the form $(t_1,\ldots,t_n):r_1(v_1),\ldots,r_m(v_m):\Psi$. The target list will be abbreviated t. In (2)-(7) we will also abbreviate other target lists.

(2) To translate the projection e[X], assume for notational convenience that the free variables in t[X] are $v_1,\ldots,v_h$. Then with e[X] we associate the alpha

$$t[X] : r_1(v_1),\ldots,r_h(v_h) : (\exists r_{h+1} v_{h+1})\ldots(\exists r_m v_m)\Psi.$$

(3) With $e\{X\Theta Y\}$ we associate $t : r_1,\ldots,r_m : \Psi \& t[X] \Theta t[Y]$.

(4) With e<X,f> we associate

$(v_1[X],\ f(v_2:\mathfrak{a}(v_2):v_2[X]=v_1[X]))\ :\ \mathfrak{a}(v_1)\ :\ -\ .$

For the next three steps assume e_1 and e_2 are associated with $\mathfrak{a}_1$ and $\mathfrak{a}_2$, respectively.

(5) With $e_1 \times e_2$ associate $v_1, v_2\ :\ \mathfrak{a}_1(v_1)\ :\ \mathfrak{a}_2(v_2)$.

(6) With $e_1 \cup e_2$ associate $v_1\ :\ \mathfrak{a}_1(v_1) \vee \mathfrak{a}_2(v_1)\ :\ -$.

(7) With $e_1 - e_2$ associate $v_1\ :\ \mathfrak{a}_1(v_1)\ :\ \sim\mathfrak{a}_2(v_1)$.

All steps excluding possibly (2) should be clear.

For step (2), suppose $a \in e[X](I)$. There is an $a' \in e(I)$ with $a'[X] = a$. By induction, $a' \in \mathfrak{a}(I)$, so for some valuation x defined on $v_1,\ldots,v_h$, $a' = (t_1(I,x),\ldots,t_n(I,x))$ and $\Psi(I,x) = 1$. From this we get $a = t[X](I,x)$ and $(\exists r_{h+1}v_{h+1})\ldots(\exists r_m v_m)\Psi(I,x) = 1$.

For the converse, suppose $a \in t[X]\ :\ r_1(v_1),\ldots,r_h(v_h):(\exists r_{h+1}v_{h+1})\ldots(\exists r_m v_m)\Psi(I)$. For some valuation x $a = t[X](I,x)$, and $(\exists r_{h+1}v_{h+1})\ldots(\exists r_m v_m)\Psi(I) = 1$. From this we can get a valuation x' such that $\Psi(I,x') = 1$. Let $a' = t(I,x')$. By induction, $a' \in e(I)$, and we see that $a'[X] = a$; hence $a \in e[X](I)$.

4. Translating Calculus to Algebra

The goal of this section is to prove that every closed alpha has an equivalent algebra expression. Formally, for every $\mathfrak{a} \in C\mathbb{A}$, there is an $e \in \mathbb{E}$ such that $\mathfrak{a}(I) = e(I)$ for all $I \in \mathbb{I}$.

Our approach is necessarily different from Codd's. The latter approach[6] sought to form a cross product from all range expressions and use restriction, projection and division to extract the answer from this cross product. With aggregate functions, new columns are created, and forming cross products will not work. We attack alpha expressions directly: For all terms, formulas and alphas, we seek to produce "equivalent" algebra expressions. These are then combined recursively. The algebraic representation for a term should evaluate to a set containing a single 1-tuple which is the interpretation of the term. The algebraic representation for a formula should evaluate to an empty set if the interpretation of the formula is false and to a nonempty set if it is true. The algebraic representation of an alpha should evaluate to a set of tuples which equals the interpretation of that alpha.

Clearly, the problem with this approach is the presence of free variables. How do we represent free variables

---

[6] Codd's proof seems to be in error: The first step in his reduction, putting the qualification into prenex normal form, is impossible with range-coupled quantifiers. For example, the formula:

$$p \vee (\exists r_i v_i) q,$$

where v_i is not free in p, would be transformed to the formula:

$$(\exists r_i v_i)(p \vee q).$$

These formulas are not equivalent, however, because the second can never be true in an instance I if $r_i(I)$ is empty whiles the first can be.

in an algebraic expression? The answer is that we represent calculus objects not by just an algebra expression, but by an algebra expression plus a restriction clause. The function of the restriction clause is to represent the free variables of the calculus object.

We form an algebraic expression D representing all possible valuations. The interpretation of a calculus object in instance I with valuation x will then correspond to the value of the algebraic expression $(D\|\sigma_x\|\|\rho\|e)[Z]$ on I, where σ_x is a selection clause corresponding to valuation x, and ρ and e are the join clause and algebraic expression representing the calculus object. The projection Z serves to remove D and other unneeded factors.

Let α have unique scopes for all of it variables $v_1,\ldots,v_n$, and assume the theorem is true for all alphas smaller than α. Thus, there are algebraic expressions $D_1,\ldots,D_n$ such that $r_i(I) = D_i(I)$ for all $I \in \mathbb{I}$. For each i, we let D_i^* be $(\{1\}-(\{1\}\times D_i)[1])\times\{1\}^{n-1} \cup D_i$, where n is the degree of D_i. With this formula, D_i^* is never empty, and $D_i^*(I) = D_i$ if $D_i(I) \neq \emptyset$. Let D be $D_1^*\times\ldots\times D_n^*$.

For notational convenience we define:

$$K = \deg(D),$$

$$K_i = \deg(D_1^*\times\ldots\times D_{i-1}^*)$$

$$d_i = K_i+1,\ldots,K_i+\deg(D_i)$$

For every valuation x there is a corresponding selection clause σ_x such that

$$D\{\sigma_x\}[d_i] = \{x_i\}, \text{ if } x \text{ is defined on } i.$$

We let D_x denote $D\{\sigma_x\}$. In the appendix we show that an expression of the form $(D_x\{\rho\}e)[Z_e]$, where Z_e refers to domains of E, is equivalent to $e\{\rho_x\}[Z_e]$, where ρ_x is a selection derived from D_x and ρ.

The translation to the algebra is done in three steps:

(a) For every term t we define an expression e, a join clause ρ and a projection list Z such that for all I and x, if t(I,x) is defined, then

$$(D_x\{\rho\}e)[Z](I) = \{t(I,x)\}.$$

(b) For every formula Ψ we define an expression e, a join clause ρ and a formula E such that for all I and x, if $\Psi(I,x)$ is defined, then

$$(D_x\{\rho\}E)(I) \neq \emptyset,$$
$$(D_x\{\rho\}e)(I) = (D_x\{\rho\}E)(I), \text{ if } \Psi(I,x)=1,$$
$$(D_x\{\rho\}e)(I) = \emptyset, \text{ if } \Psi(I,x)=\emptyset.$$

The expression E will be used for negations.

(c) For every alpha α' contained in α we define an expression e, a join clause ρ and a projection list Z such that for all I and x, if $\alpha'(I,x)$ is defined, then

$$(D_x\{\rho\}e)[Z](I) = \alpha'(I,x).$$

In every case it will be seen that the join clause ρ will join a part of e to D_i* iff v_i is free in the corresponding calculus object.

Note that once we prove property (c) we are almost done since for a closed alpha α we can (and will) easily show that $\alpha(I) = e[Z](I)$.

First we give the details of these constructions, and then we prove the stated properties.

To keep the notation manageable, we do not derive new selection, restriction and projection clauses when rearranging expressions. For example, to be precise, the rule that restrictions commute with projections as long as the domains are in the projection is precisely written as:

$$e\{X\Theta Y\}[Z] \equiv e[Z]\{X'\Theta Y'\},$$

where X' is the position in Z at which X occurred, and Y' is the position in Z at which Y occurred. However, we will simply write:

$$e\{X\Theta Y\}[Z] \equiv e[Z]\{X\Theta Y\},$$

and this should not cause any confusion.

As another notational convenience, we use projections with possibly empty projection lists. An empty projection is

defined as follows:

$$e[] \equiv (\{1\}\times e)[1].$$

The essential property of e[] is:

$$e[](I) = \emptyset \quad \text{iff} \quad e(I) = \emptyset.$$

(a) Terms: For a constant term c, let e be {c}; let ρ be the empty clause, and let Z be K+1.

For a term $v_i[A]$, let e be $D_i[A]$; let ρ be $K_i+A\equiv 1$, and let Z be K+1.

For an aggregate term $f(\alpha)$ where the translation of α has yielded e_α, ρ_α and Z_α, write ρ_α as $d_1,d_2,\ldots,d_k\equiv a_1,a_2,\ldots,a_k$, where the d's refer to domains of D. Then let e be $e_\alpha[(a_1,\ldots,a_k),Z_\alpha]<(1,\ldots,k),f>$. Let ρ be $d_1d_2\ldots d_k\equiv 1,2,\ldots,k$, and let Z be K+k+1.

(b) Formulas: For an atomic formula $t_1\theta t_2$, suppose we already have e_1, ρ_1, Z_1 and e_2, ρ_2, Z_2 for t_1, t_2, respectively. Write ρ_1, ρ_2, respectively, as $d_1\equiv w_1$ and $d_2\equiv w_2$. Let e be $(e_1\{Z_1\theta Z_2\}e_2)[w_1,w_2]$. Let ρ be (ρ_1,ρ_2). Let E be $(e_1\times e_2)[w_1,w_2]$.

For an atomic formula $\alpha'(v_i)$, suppose the translation for α' has yielded e'. (α' is closed, so ρ is empty.) Let e be e'; let ρ be $d_i\equiv d'$, where d' is 1,...,deg(e'). Let E be D_i^*.

For a negation $\sim\psi$, suppose the algebraic parts for ψ are e_ψ, ρ_ψ and E_ψ. Let e be $E_\psi - e_\psi$; let ρ be ρ_ψ; let E be E_ψ.

For a disjunction $(\psi \vee \pi)$, suppose we have obtained e_ψ, ρ_ψ, E_ψ and e_π, ρ_π and E_π. Let e be $(E_\psi \times e_\pi) \cup (e_\psi \times E_\pi)$. Let ρ be (ρ_ψ, ρ_π). Let E be $E_\psi \times E_\pi$.

For a quantification $(\exists r_i v_i)\psi$, suppose e_ψ, ρ_ψ and E_ψ have been defined. Write ρ_ψ as $(d\equiv w, \rho^-)$, where $d\equiv w$ are the join terms referring to D_i*. Let e be $((D \{ d\equiv w \} e_\psi)[\sim w] \times D_i)[\sim w]$, where $\sim w$ refers to the domains of e not in w. (The D_i term makes sure that if the range D_i is empty, then $(D_x \{ \rho \} e)(I)$ is always empty.) Let ρ be ρ^-, and let E be $E_\psi[\sim w]$.

(c) Alphas: For an atomic alpha R_i, let e be R_i; let ρ be empty, and let Z be $K+1,\ldots,K+\deg(R_i)$.

Suppose α' is $(t_1,\ldots,t_n):r_1,\ldots,r_m:\psi$. (We assume the D's are renumbered for notational convenience.) Assume the translation has yielded e_i, ρ_i and Z_i for each t_i and e_ψ and ρ_ψ for ψ. Let e be

$$((D \times e_1 \times \ldots \times e_n \times e_\psi) \{ \rho_1, \ldots, \rho_n, \rho_\psi^+ \} \times D_1 \times \ldots \times D_m)[W],$$

where ρ_ψ^+ are the clauses of $\rho\psi$ referring to $D_1,\ldots,D_m$, and where W projects out $D_1 \times \ldots \times D_m$. (The $D_1 \times \ldots \times D_m$ term makes sure that no tuples go in the result when any of the free variable ranges is empty.) Let ρ be obtained from ρ_ψ by deleting all terms referring to $D_1,\ldots,D_m$.

Let Z be $Z_1,....Z_n$.

Now we proceed to prove these properties. In every case the hypothesis is that the interpretation of the object being considered is defined at instance I and valuation x. The numbers in parentheses at the end of lines refer to items in the appendix justifying the step which derives the next line.

(a) For a constant term c, we have

$(D_x\{\!|\rho|\!\}e)[Z](I)$

$= (D_x \times \{c\})[K](I)$ (1)

$= \{c\}(I)$

$= \{c\}$

$= \{c(I,x)\}$.

For a term $v_i[A]$, we have

$(D_x\{\!|\rho|\!\}e)[Z](I)$

$= (D_x\{\!|K_i+A\equiv 1|\!\}D_i[A])[K+1](I)$ (1)

$= D_i[A]\{\!|1\equiv x_i[A]|\!\}(I)$

$= \{x_i[A]\}$

$= \{v_i[A](I,x)\}$.

For a term $f(\alpha)$, we have

$(D_x\{\!|\rho|\!\}e)[Z](I)$

$= (D_x\{\!|d_1,...,d_k\equiv 1,...,k|\!\}(e_\alpha[(a_1,...,a_k),Z_\alpha]$

$<(1,...,k),f>))[K+k+1](I)$ (1)

$= e_\alpha[(a_1,...,a_k),Z_\alpha]<(1,...,k),f>\{\!|\rho_x|\!\}[k+1](I)$ (2)

$= \overline{f}(e_\alpha\{\!|\rho_x|\!\}[Z_\alpha](I))$ (1)

$= \overline{f}((D_x \{\!| \rho |\!\} e_\alpha)[Z_\alpha](I))$

$= \overline{f}(\alpha(I,x))$.

(b) In this section we make use of the fact that $(D_x \{\!| \rho |\!\} e_1)(I)$ $= (D_x \{\!| \rho |\!\} e_2)(I)$ iff $(D_x \{\!| \rho |\!\} e_1)[Z_e](I) =$ $(D_x \{\!| \rho |\!\} e_2)[Z_e](I)$,

where Z_e consists of the domains of e_1 (= domains of e_2).

For an atomic formula $t_1 \theta t_2$ we have:

$(D_x \{\!| \rho |\!\} e)[Z_e](I)$

$= (D_x \{\!| \rho_1, \rho_2 |\!\} ((e_1 \{\!| Z_1 \theta Z_2 |\!\} e_2)[w_1, w_2]))[Z_e](I)$ (3)

$= (D_x \{\!| \rho_1, \rho_2 |\!\} (e_1 \{\!| Z_1 \theta Z_2 |\!\} e_2))[w_1, w_2](I)$ (1)

$= (e_1 \{\!| Z_1 \theta Z_2 |\!\} e_2) \{\!| \rho_{1x}, \rho_{2x} |\!\} [w_1, w_2](I)$ (4)

$= ((e_1 \{\!| \rho_{1x} |\!\}) \{\!| Z_1 \theta Z_2 |\!\} (e_2 \{\!| \rho_{2x} |\!\}))[w_1, w_2](I)$ (1)

$= ((D_x \{\!| \rho_1 |\!\} e_1) \{\!| Z_1 \theta Z_2 |\!\} (D_x \{\!| \rho_2 |\!\} e_2))[w_1, w_2](I)$ (5)

$= ((D_x \{\!| \rho_1 |\!\} e_1) \times (D_x \{\!| \rho_2 |\!\} e_2))[w_1, w_2](I)$

if $t_1(I,x)$ is θ to $t_2(I,x)$;

$= \emptyset$ if $t_1(I,x)$ is not θ to $t_2(I,x)$.

Also,

$(D_x \{\!| \rho |\!\} E)[Z_e](I)$

$= (D_x \{\!| \rho_1, \rho_2 |\!\} (e_1 \times e_2)[w_1, w_2])[Z_e](I)$ (as above)

$= ((D_x \{\!| \rho_1 |\!\} e_1) \times (D_x \{\!| \rho_2 |\!\} e_2))[w_1, w_2](I)$ (6)

$\neq \emptyset$.

For an atomic formula $\alpha'(v_i)$, we have

$(D_x \{\!| \rho |\!\} e)[Z_e](I)$

$= (D_x \{\!| d_i \equiv d' |\!\} e')[Z_e](I)$ (1)

$= e' \{\!| d' \equiv x_i |\!\} (I)$

$= \{x_i\} = D_i^*\{d' \equiv x_i\}(I)$ if $x_i \in e'(I)$

$= \emptyset$ if $x_i \notin e'(I)$.

Also,

$(D_x\{\rho\}E)[Z_e](I)$

$= (D_x\{d_i \equiv d'\}D_i^*)[Z_e](I)$

$= D_i^*\{d' \equiv x_i\}(I)$

$= \{x_i\}$

$\neq \emptyset$.

For a negation $\sim\psi$ we have:

$(D_x\{\rho\}e)(I)$

$= (D_x\{\rho_\psi\}(E_\psi - e_\psi))(I)$ (7)

$= (D_x\{\rho_\psi\}E_\psi)(I) - (D_x\{\rho_\psi\}e_\psi)(I)$

$= (D_x\{\rho_\psi\}E_\psi)(I) - (D_x\{\rho_\psi\}E_\psi)(I) = \emptyset$

if $\sim\psi(I,x)=\emptyset$ $(\psi(I,x)=1)$,

$= (D_x\{\rho_\psi\}E_\psi)(I) - \emptyset = (D_x\{\rho_\psi\}E_\psi)(I)$

if $\sim\psi(I,x)=1$ $(\psi(I,x)=\emptyset)$.

Also,

$(D_x\{\rho\}E)(I)$

$\neq \emptyset$.

For a disjunction $(\psi \vee \pi)$ we have:

$(D_x\{\rho\}e)[Z_e](I)$

$= (D_x\{\rho_\psi, \rho_\pi\}(E_\psi \times e_\pi \cup e_\psi \times E_\pi))[Z_e](I)$ (8)

$= (D_x\{\rho_\psi, \rho_\pi\}(E_\psi \times e_\pi))[Z_e](I) \cup (D_x\{\rho_\psi, \rho_\pi\}(e_\psi \times E_\pi))[Z_e](I)$ (9)

$= (D_x\{\rho_\psi, \rho_\pi\}(E_\psi \times E_\pi))[Z_e](I)$,

if $\psi(I,x)=1$ or $\pi(I,x)=1$

$= \emptyset$ if $\psi(I,x)=\emptyset$ and $\pi(I,x)=\emptyset$.

Also,

$(D_x \{\!| \rho |\!\} E)(I)$

$\neq \emptyset$.

For a quantification $(\exists r_i v_i)\Psi$ we have

If $r_i(I)$ $(= D_i(I))$ $= \emptyset$, then $(\exists r_i v_i)\Psi(I,x) = \emptyset$, and by construction (because of the D_i cross product term in e), $(D_x \{\!| \rho |\!\} e)[Z_e](I) = \emptyset$. Otherwise, we have

$(D_x \{\!| \rho |\!\} e)[Z_e](I)$ (10)

$= (D_x \{\!| \rho^- |\!\} ((D \{\!| d \equiv w |\!\} e_\Psi)[\tilde{}w]))[Z_e](I)$ (1)

$= (D \{\!| d \equiv w |\!\} e_\Psi)[\tilde{}w] \{\!| \rho_x^- |\!\}(I)$ (11)

$= U_{a \in r_i(I)} \ (D_{x[i/a]} \{\!| \rho |\!\} e_\Psi)[\tilde{}w](I)$

$= U_{a \in r_i(I)} \ (D_{x[i/a]} \{\!| \rho |\!\} E_\Psi)[\tilde{}w](I)$ (12)

$\quad \& \ \Psi(I,x[i/a])=1$

$= U_{a \in r_i(I)} \ E_\Psi \{\!| \rho_{x[i/a]} |\!\}[\tilde{}w](I)$

$\quad \& \ \Psi(I,x[i/a])=1$

$= \emptyset$ (empty union) if there is no $a \in r_i(I)$

with $\Psi(I,x[i/a])=1$,

i.e., if $(\exists r_i v_i)\Psi(I,x)=\emptyset$,

otherwise if there is some $a \in r_i(I)$

with $\Psi(I,x[i/a])=1$ (13)

$= E_\Psi[\tilde{}w](I)$

$= E_\Psi \{\!| \rho_x |\!\}[\tilde{}w](I)$

$= (D_x \{\!| \rho |\!\}(E_\Psi[\tilde{}w]))[Z_e](I)$

$= (D_x \{\!| \rho |\!\} E)[Z_e](I)$.

Also,

$(D_x \{\!| \rho |\!\} E)(I)$

$= \ldots = (D_x \{\rho_\psi\} E_\psi)[\tilde{w}](I)$

$\neq \emptyset.$

If $\tilde{w}$ happens to be empty, the equalities will all still hold. (In this case, ρ^- is empty.)

(c) For an atomic alpha R_i, we have

$(D_x \{\rho\} e)[Z](I)$

$= (D_x \times R_i)[d'](I)$ (1)

$= R_i(I)$

$= R_i(I,x).$

Now consider nonatomic alpha α'. If one of the ranges $r_1(I),\ldots,r(I)_m$ is empty, then $\alpha'(I,x)=\emptyset$, and we also have by construction $(D_x \{\rho\} e)[Z](I) = \emptyset$. Otherwise we have:

$(D_x \{\rho\} e)[Z](I)$ (like 10)

$= (D_x \{\rho^-_\psi\} (D \{\rho_1,\ldots,\rho_n,\rho^+_\psi\}(e_1 \times \ldots \times e_n \times e_\psi)))[Z_1,\ldots,Z_n](I)$ (14)

$= \bigcup_{j=1,\ldots,m} (D_{x[j/a_j]}[\rho_1,\ldots,\rho_n,\rho_\psi](e_1 \times \ldots \times e_n \times e_\psi))$

$a_j \in r_j(I)$ $[Z_1,\ldots,Z_n](I)$ (15)

$= \bigcup_{j=1,\ldots,m} (D_{x[j/a_j]} \{\rho_1\} e_1)[Z_1](I)$

$a_j \in r_j(I)$

$\psi(I,x[j/a_j])=1$ $\times$

$\ldots$

$\times$

$(D_{x[j/a_j]} \{\rho_n\} e_n)[Z_n](I)$

$= \bigcup_{j=1,\ldots,m} \{t_1(I,x[j/a_j])\} \times \ldots \times \{t_n(I,x[j/a_j])\}$

$a_j \in r_j(I)$

$\psi(I,x[j/a_j])=1$

$= \alpha'(I,x)$.

Given any $\alpha \in \mathbb{A}$ we now have the identity:

$$(D_x \| \rho \| e)[Z](I) = \alpha(I,x).$$

If α is closed, then ρ is empty and we have:

$$(D_x \| \rho \| e)[Z](I) =$$
$$(D_x \times e)[Z](I) =$$
$$e[Z](I).$$

Thus, for all $I \in \mathbb{I}$, $\alpha(I) = e[Z'](I)$, and α is equivalent to an algebra expression.

5. Summary and Future Work

Report writers may be thought of as query languages having aggregate functions. Within the framework of the relational model we have formally defined aggregate functions so that the imprecise notion of "duplicates are not removed" is not needed.

Relational algebra was extended to include an aggregate formation operator. It partitions its operand by specified columns and applies the aggregate function to each member of the partition.

Relational calculus was extended so that a term, which may appear in a target list or in a qualification, can be formed

by applying an aggregate function to a calculus expression. Linking of outer terms to terms within an aggregate term is accomplished by using free variables. It was necessary to allow variables to range over alpha expressions rather than simply over unions and projections of base relations.

We showed that the set of queries expressible in the algebra is the same as the set of queries expressible in the calculus.

5.1. Future Work

The results reported here can form a foundation for more work in the following directions:

- Develop algorithms for logical optimization of expressions involving aggregate functions.
- Develop algorithms for deciding the equivalence of expressions involving aggregate functions.
- Extend the languages to include additional types of operations such as arithmetic operators on terms.
- Investigate the complexity of the calculus-to-algebra translation. Our examples have indicated that it might have a high complexity.

6. Appendix -- Details of Calculus-to-Algebra Proof

(1) $(D_x\{d \equiv w\}e)[Z](I) = e\{w \equiv x_d\}[Z](I)$; Z consists of domains of e, x_d is the subsequence of x formed from d, and x

is defined at all places referenced by d.

If $t \in (D_x\{d \equiv w\}e)[Z](I)$. then there are tuples $t_x \in D_x(I)$ and $t_e \in e(I)$ with $t_x[d] = t_e[w]$ and $t_e[Z] = t$. But by definition $t_x[d] = x_d$, so $t_e \in e\{w \equiv x_d\}(I)$, and $t \in e\{w \equiv x_d\}[Z](I)$. The converse is similar.

(2) $e[Y,Z]\langle \overline{k}, f\rangle\{\overline{k} \equiv c\}[k+1](I) = \overline{f}(e\{Y \equiv c\}[Z])(I)$, where $k = len(Y)$ and $\overline{k} = (1,\ldots,k)$.

The hypothesis on the family F of aggregate functions says that $e[Y,Z]\langle \overline{k}, f\rangle(I) = e\langle Y,Z,f\rangle(I) = \{t[Y]\hat{}y : t \in e(I) \ \& \ y = \overline{f}(\{t'[Z] : t' \in e(I) \ \& \ t'[Y] = t[Y]\})\}$. Hence,

$e[Y,Z]\langle \overline{k}, f\rangle\{\overline{k} \equiv c\}[k+1](I)$

$= \overline{f}(\{t'[Z] : t' \in e(I) \ \& \ t'[Y] = c\})$

$= \overline{f}(e\{Y \equiv c\}[Z](I))$.

(3) Projections distribute through cross products and unions, and they commute with restrictions (if all restricted domains are retained) and combine with themselves. See [Ullm].

(4) Restrictions commute with other restrictions and with cross products [Ullm].

(5) If $t_1(I,x)$ is θ to $t_2(I,x)$, then if $s_1\hat{}s_2 \in ((D_x\{\rho_1\}e_1) \times (D_x\{\rho_2\}e_2))(I)$, then $s_1[Z_1] = t_1(I,x)$ and $s_2[Z_2] = t_2(I,x)$. Hence $s_1[Z_1]$ is θ to $s_2[Z_2]$, and $s_1\hat{}s_2 \in ((D_x\{\rho_1\}e_1)\{Z_1 \theta Z_2\}(D_x\{\rho_2\}e_2))(I)$.

(6) Since $t_1(I,x)$ and $t_2(I,x)$ are defined,

$(D_x\{\!|\rho_1|\!\}e_1)[Z_1](I)$ and $(D_x\{\!|\rho_2|\!\}e_2)[Z_2](I)$ are nonempty.

(7) Restrictions distribute through set differences.

(8) Restrictions distribute through unions.

(9) This follows by using rule (1) and the above distributive and commutative rules.

(10) With $D_i(I) \neq \emptyset$, we use rule (1) to eliminate the D_i term.

(11) We want to show that

$$(D\{\!|d\equiv w|\!\}e_\psi)[\tilde{}w]\{\!|\rho_x^-|\!\}(I)$$
$$= U_{a\in r_i(I)}\ e_\psi\{\!|w\equiv a_d|\!\}[\tilde{}w]\{\!|\rho_x^-|\!\}(I).$$

Since d in $d\equiv w$ refers only to D_i, we can write $(D\{\!|d\equiv w|\!\}e_\psi)[\tilde{}w]$ as $(D_i\{\!|d\equiv w|\!\}e)[\tilde{}w]$. Now $D_i(I) = U_{a\in D_i(I)}\{a\}(I)$. Since (set theoretic) union commutes with cross product and restriction (and hence join), we have

$$(D_i\{\!|d\equiv w|\!\}e_\psi)[\tilde{}w]\{\!|\rho_x^-|\!\}(I)$$
$$= U_{a\in D_i(I)}\ e_\psi\{\!|w\equiv a_d|\!\}[\tilde{}w]\{\!|\rho_x^-|\!\}(I)$$
$$= U_{a\in D_i(I)}\ e_\psi\{\!|w\equiv a_d,\rho_x^-|\!\}[\tilde{}w](I),$$

where a_d are the components of a referred to by d. But $(w\equiv a_d,\rho_x^-)$ is just $\rho_{x[i/a]}$, and we have

$$(D_i\{\!|d\equiv w|\!\}e_\psi)[\tilde{}w]\{\!|\rho_x^-|\!\}(I)$$
$$= U_{a\in D_i(I)}\ e_\psi\{\!|\rho_{x[i/a]}|\!\}[\tilde{}w](I)$$
$$= U_{a\in D_i(I)}\ (D_x\{\!|\rho_\psi|\!\}e_\psi)[\tilde{}w](I)$$

(12) We use induction and restrict the union to the cases

where $\Psi(I,x[i/a])=1$.

(13) For any E associated with a formula, any restrictions ρ_1 and ρ_2 and any projection Z which does not include any domains of ρ_1, $E\{\rho_1\}\{\rho_2\}[Z](I) = E\{\rho_2\}[Z](I)$. This follows because E is equivalent to an expression having only cross products and projections. That E is so equivalent should be clear from the construction noting that in the case of atomic formulas of the form $t_1 \theta t_2$, the associated E has the form:

$(e_1\langle X_1,f_1\rangle \times e_2\langle X_2,f_2\rangle)[X_1,X_2]$,

where e_1 and e_2 have only cross products and projections. It is easy to see that this expression is equivalent to

$e_1[X_1] \times e_2[X_2]$.

(14) The proof is as in (11). We must only generalize to more that one quantified variable. The notation $x[j/a_j]$ is short for

$$x[1/a_1][2/a_2]\ldots[m/a_m].$$

(15) This follows from rule (1) and the induction hypothesis for Ψ.

7. References

[ABCE] Astrahan M.M., Blasgen M.W., Chamberlin D.D., Eswaran K.P., Gray J.N., Griffiths P.P., King W.F., Lorie R.A., McJones P.R., Mehl J.W., Putzolu G.R., Traiger I.L., Wade B.W. and Watson V. "System R: Relational Approach to Database Management" ACM-TODS 1, 2, pp.97-137 (1976)

[BeSl] Bell J.L. and Slomson A.B. "Models and Ultraproducts: An Introduction" North Holland Pub. Co., 1971

[CAEG] Chamberlin D.D., Astrahan M.M., Eswaran K.P., Griffiths P.P., Lorie R.A., Mehl J.W., Reisner P., and Wade B.W. "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control" IBM Journal of Research and Development, 1, pp.560-575, and in IBM Research Report RJ 1798

[ChBo] Chamberlin D.D., Boyce R.F. "SEQUEL: A Structured English Query Language" ACM SIGMOD Workshop on Data Description, Access and Control, 1974

[Codd70] Codd E.F. "A Relational Model of Data for Large Shared Data Banks" CACM, 13, pp.377-387, 1970

[Codd72a] Codd E.F. "Relational Completeness of Data Base Sublanguages" Data Base Systems, R. Rustin (ed.), Prentice Hall, 1972

[Codd72b] Codd E.F. "Further Normalization of the Data Base Relational Model" Data Base Systems, R. Rustin (ed.), Prentice Hall, 1972

[HeSW] Held G.D., Stonebraker M.R. and Wong E. "INGRES -- A Relational Data Base System", NCC 1975

[RoLe] Robinson L. and Levitt K.N. "Proof Techniques for Hierarchically Structured Programs" CACM 20, pp. 271-283 (1977)

[Shoe] Shoenfield J.R. "Mathematical Logic", Addison-Wesley, Reading-London, 1967

[SWKH] Stonebraker M., Wong E., Kreps P. and Held G. "The Design and Implementation of INGRES", ACM-TODS 1, #3, 1976, pp.189-222

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