Variable Independence and Aggregation Closure

J. Chomicki
Monmouth University
West Long Branch, NJ
chomicki@moncol.monmouth.edu

D. Q. Goldin *
Brown University
Providence, RI
dgk@cs.brown.edu

G. M. Kuper
ECRC
München, Germany
kuper@ecrc.de

Abstract

We discuss the issue of adding aggregation to constraint databases. Previous work has shown that, in general, adding aggregates to constraint databases results in languages that are not closed. We show that by imposing a natural restriction, called *variable independence* (which is a generalization of the assumptions underlying the classical relational model of data) on the schema, we can guarantee that a restricted version of the language with aggregation is closed. We illustrate our approach in the context of linear constraint databases.

1 Introduction

Constraint databases [KKR90] are a natural generalization of the relational model of data by allowing infinite relations that are finitely representable using constraints. Constraint databases find numerous applications in spatial [BJM93, BK95, BLLM95, PVdBVG94, VGVG95] and temporal databases [Cho94]. Generalizing aggregation operators to constraint databases has been identified as one of the most important open research issues in this area [KG94, Kan95]. Some aggregation operators like **count** are not applicable to infinite relations. On the other hand, new operators like **area** [AS91] (or its generalization: n-dimensional volume [GK94]) occur there quite naturally.

A query language resulting from adding aggregation to a constraint query calculus [KKR90] or algebra [KG96] should be well behaved. [Kup94] describes a general framework, modelled after [Klu82], for adding aggregate operators to relational algebra and calculus. The outstanding open problem there is that of *closure*: the result of a query that uses aggregation should be finitely representable using the constraint language of the database. Unrestricted aggregation may fail to produce a closed language, as shown in [Kup94]. That paper proposes several approaches to deal with this problem. On the one hand, one may try to restrict the query language, and on the other, to restrict the classes of aggregate operators allowed. The second approach was illustrated by the **max** aggregate operator that (trivially) produces a closed language. Also, a new nontrivial aggregation operator producing a closed language was introduced and studied in [CK95].

A new approach, restricting the way aggregate operators are used in the query language, is proposed here. We show that, under certain natural restrictions captured by the notion of *variable independence* and reflecting the way aggregation is often used in real databases, we can add aggregate operators to relational algebra, and still get a closed language. In particular, our approach is applicable to the **area** aggregate operator and its higher-dimensional versions.

As shown in [Kup94], relational algebra over linear constraint databases is not closed under aggregation using **area**. The typical example where a query is not closed, is where we have, say, a region whose boundaries vary with time, and we want to know how the area of the region varies with time.

While there are applications where one might conceivably need the full generality of such a language (and for which the problems would be unavoidable), this is not the case for many applications.

Example 1.1 Consider, for example, a geographical database with cadastral information, i.e., information on land ownership and land boundaries. Suppose we are interested in finding the area of the land owned by each person at all points of time. Land ownership does not vary continuously–somebody acquires a piece of land at a single, discrete point of time.

This means that the data can be represented by a set of generalized tuples of the form

$$n = N \wedge t_1 \leq t \leq t_2 \wedge C(x, y)$$

*Research supported in part by ONR contract N00014-94-1-1153 and NSF grant IRI-9509933.

where $C(x, y)$ are constraints that describe the region owned by N between times t_1 and t_2. If we want to find the area of the land owned by N as a function of t, we can represent the result as a set of generalized tuple of the form

$$t_1 < t < t_2 \wedge (n = N) \wedge (z = A)$$

which is clearly finitely representable.

The important property of such tuples, that we shall generalize in this paper, is that the constraints on x and y—those on which the area computation is performed—are separate from the constraints on n and t. Many typical uses of area computation in GIS systems, such as the current example, have this property, so that such a restriction is quite natural.

There are, of course, possible applications for which such an assumption would not hold, and will not be covered by our approach. One example would be to give the area covered by a storm as a function of time. If the data is represented by linear constraints, there is no way around the need for quadratic constraints to express the result. On the other hand, if we are interested in the area at a specific point of time, then we would first perform a selection (time = constant), and the results in this paper would still apply.

The paper is organized as follows. In section 2, we review the basic concepts of constraint databases and aggregation. In section 3, we define the notion of *variable independence* that is central to the notion of restricted aggregation that we propose. In section 4, we define the relational algebra with restricted aggregation and show that it is closed. In section 5, we show that for constraint databases with linear arithmetic constraints variable independence is effectively decidable. In section 6, we present some results about inferring variable independence in relational algebra expressions. In section 7, we draw conclusions and discuss related and further work.

2 Basic notions

For a full account of constraint databases, see [KKR90]. The basic definitions are as follows:

Definition 2.1 *Let Φ be a class of constraints.*

1. *A* generalized k-tuple *(over variables $x_1, \ldots, x_k$) is a finite conjunction $\varphi_1 \wedge \cdots \wedge \varphi_N$, where each $\varphi_i, 1 \leq i \leq N$, is a constraint in Φ. Furthermore, the variables in each φ_i are all free and among $x_1, \ldots, x_k$.*
2. *A* generalized relation of arity k *is a finite set $r = \{\psi_1, \ldots, \psi_M\}$, where each $\psi_i, 1 \leq i \leq M$ is a generalized k-tuple over the same variables $x_1, \ldots, x_k$.*
3. *The* formula corresponding to a generalized relation *r is the disjunction $\psi_1 \vee \cdots \vee \psi_M$. We use ϕ_r to denote the quantifier-free formula corresponding to the relation r.*
4. *A* generalized database *is a finite set of generalized relations.* □

We view a set of variables $\{x_1, \ldots, x_k\}$ as a *relation schema* and generalized relations over $\{x_1, \ldots, x_k\}$ as *instances* of this schema.

In database theory, a k-ary relation r is a finite set of k-tuples (or points in a k-dimensional space) and a database is a finite set of relations. However, the relational calculus and algebra can be developed without the finiteness assumption for relations. We will use the term *unrestricted relation* for finite or infinite sets of points in a k-dimensional space. In order to be able to do something useful with such unrestricted relations, we need a finite representation that we can manipulate. This is exactly what the generalized tuples provide.

Definition 2.2 *Let Φ be a class of constraints interpreted over domain D, r a generalized relation of arity k with constraints in Φ, let ϕ_r be the formula corresponding to r with free variables $x_1, \ldots, x_k$. The generalized relation r represents the unrestricted k-ary relation which consists of all $(a_1, \ldots, a_k)$ in D^k such that $\phi_r(a_1, \ldots, a_k)$ is true. Two generalized relations over the same set of variables are* equivalent *if they represent the same unrestricted relation.* □

With some abuse of notation, we will use the same symbol for a generalized tuple (relation) and the unrestricted relation it represents.

A query ϕ on a constraint database is a first-order formula, whose predicates are constraints or generalized relation symbols. The semantics of a query, are, intuitively, the mapping from unrestricted relations to unrestricted relations defined by this formula. Such a query is well-defined if the result of substituting a generalized relation for each occurrence of the corresponding relation symbol, is equivalent to some generalized relation over the same constraint domain.

A query algebra can also be defined, using natural extensions of the normal relational algebra (see [KG96] for an extensive treatment of constraint query algebras). The extension to aggregation is described later in this section.

A query language L is *closed* for a class of constraint databases C if the result of any query belonging to L evaluated over a database from C can be finitely represented as a generalized relation from C.

We assume that the constraint language is closed under negation. Moreover, it should admit effective quantifier elimination. These assumptions are necessary for the closure of relational algebra operations. For the rest of this paper, we consider only those constraint languages that satisfy the above conditions. In particular, we consider linear arithmetic constraints and dense order constraints; linear arithmetic constraints are of the form $a_1x_1 + \cdots + a_m x_m \; op \; a_0$ with $op \in \{\leq, <, =\}$ and with rational coefficients $a_0, \ldots, a_m$.

As shown in [Kup94], Klug's relational calculus and algebra with aggregation can be extended to constraint databases with minor modifications, at least as far as the underlying semantics on unrestricted relations is concerned. The definitions in [Kup94] are as follows:

Definition 2.3 *An* aggregate function f *maps (possibly infinite) relations with an appropriate schema to the domain D of the constraints. For every relation S over attributes X, if S' is a constant expansion of S over $X \cup Y$, i.e., if there is a projection such that $\pi_X(S') = S$ and $\pi_Y(S')$ contains exactly one tuple, then the function f' such that $f(S) = f'(S')$ is also an aggregate function.*

For our purposes here, it is sufficient to assume that the algebra is extended by the following operator for every aggregate function:

Definition 2.4 *Let r be an unrestricted relation over the set of attributes U, $X \subseteq U$ such that $|U - X| = n$, and f is an aggregate function of arity n which outputs a constraint over attributes Y. Then, the aggregate operator $\langle X, f \rangle$, when applied to r, produces a new relation r' with attributes $X \cup Y$:*

$$r\langle X, f\rangle = \{(t[X], y) | t \in r \wedge \\ \wedge y = f(\{t'[U - X] | t' \in r \wedge t'[X] = t[X]\})\}.$$

Intuitively, the above corresponds to grouping the tuples in r on X and applying the aggregation operator to the remaining attributes in each group. For more details, including the construction of an equivalent calculus, see [Kup94].

Generalized relational algebra with aggregation may fail to be closed even for dense order constraints [Kup94].

Example 2.1 Consider the instance of R consisting of a single generalized tuple:

$$0 < x < y < z < 1$$

The query $R\langle z, \mathbf{area}_{x,y}\rangle$ evaluates to a binary relation $S(z,t)$ where $S(z,t)$ holds iff $z^2 = 2t$. The latter constraint cannot be represented using order constraints (or linear arithmetic constraints).

In this paper, we will provide a restriction on applying the aggregate operator to a relation, and prove closure for the resulting class of relational expressions. In particular, we will define the notion of *variable independence*, and stipulate that X and $U - X$ must be independent in r (see Definition 2.4). So, the expression $R\langle z, \mathbf{area}_{x,y}\rangle$ from the above example would not be permitted by our restriction.

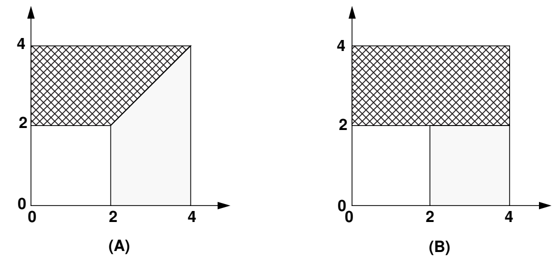

Figure 1: Instance (A) is equivalent to instance (B).

Example 2.2 Returning now to Example 1.1, suppose that the cadastral database contains a generalized relation L with tuples of the following form:

$$t_1 < t < t_2 \wedge (n = N) \wedge C(x, y)$$

where $C(x,y)$ is a conjunction of linear arithmetic constraints describing a piece of land owned by person N over the time interval (t_1, t_2). The relational algebra query $L\langle\{1,2\}, \mathbf{area}_{3,4}\rangle$ lists the area of land owned by each person at all times. This query satisfies our restriction on the use of aggregation,

On a concluding note, it should be noted that the relation r in Definition 2.4 can be either extensional or intensional. We will deal in the paper with maintaining the restriction in both of those cases.

3 Independence of Variables

In the following definitions, we assume that R is a generalized relation over attribute set U.

3.1 Definitions

Definition 3.1 *Let $X, Y \subseteq U$ and t be a generalized tuple in R. We say that X and Y are* independent in t *if:*

$$\pi_{XY}(t) = \pi_X(t) \bowtie \pi_Y(t);$$

they are related in t *otherwise.*

Clearly, variable independence in a tuple is decidable. Also, note that variable independence in a tuple t may be viewed as an *embedded join dependency* [Kan90] holding in the unrestricted relation corresponding to t.

Definition 3.2 *We say that X and Y are* independent in R *if there exists a relation R' equivalent to R where X and Y are independent in every (generalized) tuple of R'. Otherwise, we say that they are* related in R.

Example 3.1 The instance of $R(x, y)$ in Figure 1 (A) contains two tuples:

$$2 \leq y \leq 4 \wedge 0 \leq x \leq y;$$
$$2 \leq x \leq 4 \wedge 0 \leq y \leq x.$$

In each tuple, x and y are related. However, there is an an equivalent relation (Figure 1 (B)) where this is not the case:

$$2 \leq y \leq 4 \wedge 0 \leq x \leq 4;$$
$$2 \leq x \leq 4 \wedge 0 \leq y \leq 2.$$

Note that for classical relations, where each tuple always represents exactly one point, it is *trivially true* that all subsets of variables are independent in every relation. In general, however, *decidability of variable independence* is far from obvious. As the above example shows, it is not sufficient to check the individual tuples in order to verify variable independence. In section 5, we show that for linear constraints, there is an effective method to tell whether two variables are independent in a given generalized relation.

To conclude the subsection, we define variable independence for the *schema* of a generalized relation. Here, variable independence is viewed as a restriction, to be maintained by all relations that satisfy the schema. In this way, we *enrich* the relation schema beyond the standard attribute name and type information.

Definition 3.3 *We say X and Y are* independent in the relational schema $\mathcal{R}$ *if every relation satisfying $\mathcal{R}$ preserves the independence of X and Y; X and Y are* related in $\mathcal{R}$ *otherwise.*

3.2 Semantic vs. Syntactic Independence

The above definitions of variable independence are *semantic*: for each generalized tuple t, they consider the semantics of t. One can also define variable independence *syntactically*:

Definition 3.4 *Let $X, Y \subseteq U$ be disjoint, and t be a generalized tuple in R. We say that X and Y are* syntactically independent in t *if t is a conjunction of $C_1(X \cup Z)$ and $C_2(Y \cup Z)$, where $Z = U - X - Y$, and $C(X)$ denotes a conjunction of constraints over X.*

The above definition extends to relations, just as for semantic variable independence.

In Section 4, we will impose a variable independence condition on the schema of those relations to which aggregation may be applied, and show that for all reasonable constraint classes, relational algebra with *restricted aggregation* is closed. The proof will rely on the following lemma:

Lemma 3.1 *X and $U - X$ aresemantically independent in R iff they are syntactically independent in R.*

Proof: The "if" direction is straightforward. We prove the "only if" direction. By Definition 3.2, there exists R' equivalent to R where for every tuple t in R', $t = \pi_X(t) \bowtie \pi_{U-X}(t)$. Since the constraint class admits quantifier elimination, both projections are representable as finite disjunctions of conjunctions of constraints; the first one is over X, the second over $U - X$. By pairing up the disjuncts from the two projections, we obtain a finite set of tuples whose disjunction is equivalent to t, where X and $U - X$ are syntactically independent in all tuples. By repeating this construction for every t in R', we obtain a relation equivalent to R where X and $U - X$ are syntactically independent in each tuple. ∎

Note that the above lemma does not generalize to arbitrary sets of attributes; in general, semantic independence does *not* imply syntactic and vice versa. For example, $\{x\}$ and $\{y\}$ are semantically (but not syntactically) independent in the singleton relation consisting of the tuple

$$0 < x < 1 \wedge 0 < y < 1 \wedge z = x + y.$$

On the other hand, $\{x\}$ and $\{y\}$ are syntactically (but not semantically) independent in the relation consisting of the tuple

$$x < y \wedge y < z.$$

In the rest of this paper, "independence" will mean semantic independence, unless explicity stated otherwise.

4 Restricted Aggregation

In this section, we impose a variable independence condition on the schema of those relations to which aggregation is applied. We show that for all reasonable constraint classes, relational algebra with *restricted aggregation* is *closed*; i.e., the result of any relational algebra expression can be represented as a generalized relation.

Definition 4.1 *The* relational algebra with restricted aggregation *consists of the standard relational algebra, together with expressions of the form $e\langle X, f\rangle$, provided that X and $U - X$ are independent in the schema of e (where U is the set of attributes of e).*

Theorem 4.1 *Relational algebra with restricted aggregation is closed for constraint databases whenever the constraint class admits quantifier elimination and is closed under negation.*

Proof: (sketch) Let e be a relational expression on the given schema, with attributes U, and let X be a subset of U such that the expression $e\langle X, f\rangle$ is permitted in the language, and let e evaluate to the relation R_0. We must show that $R_0\langle X, f\rangle$ is representable as a generalized relation in our constraint language.

By our restrictions on the use of aggregation, combined with Lemma 3.1, there must be R equivalent to R_0 of the form

$$R \equiv \bigvee_{1 \le i \le l} C^i$$

where each C^i is of the form

$$C^i = c_1^i \wedge \cdots \wedge c_{j_i}^i \wedge d_1^i \wedge \cdots \wedge d_{k_i}^i$$

where each c_j^i contains only variables from X, and each d_j^i contains only variables from $U - X$.

Let

$$\mathcal{C} = \{c_k^i, \neg c_k^i \mid 1 \le i \le l, 1 \le k \le j_i\}$$

Because the constraint language is closed under negation, $\mathcal{C}$ is a valid constraint set. For any $\mathcal{S} \subseteq \mathcal{C}$, define $\phi_{\mathcal{S}}$ as

$$\phi_{\mathcal{S}} = \bigwedge_{c \in \mathcal{S}} c.$$

Let

$$\Phi = \{\phi_{\mathcal{S}} \mid \mathcal{S} \subseteq \mathcal{C}, \phi_{\mathcal{S}} \text{ satisfiable }\}$$

and let Ψ be the set of *minimal* formulas in Φ, i.e., those $\phi \in \Phi$ with the property that, whenever $\phi' \in \Phi$ such that $\phi' \rightarrow \phi$ is valid, then ϕ is equivalent to ϕ'.

Define R' as

$$R' \equiv \bigvee_{\phi \in \Psi} C^{\phi},$$

where

$$C^{\phi} = \phi \wedge \bigvee \{d_1^i \wedge \cdots \wedge d_{k_i}^i \mid 1 \le i \le l, \phi \rightarrow c_1^i \wedge \cdots \wedge c_{j_i}^i\}$$

We can then show:

Lemma 4.1 $R' \equiv R$

Now, let t be a tuple (i.e., an n-dimensional point) in (the semantics) of R. By the minimality of Ψ, we can easily see that $t[X]$ satisfies exactly one element of Ψ, and hence that t satisfies exactly one C^{ϕ}. Then

$$\begin{aligned}&\{t'[U - X] | t' \in R \wedge t'[X] = t[X]\} = \\ &= \bigcup \{t'[U - X] \mid t' \models d_1^i \wedge \cdots \wedge d_{k_i}^i\}\end{aligned}$$

where the union is over those d's in the definition of C^{ϕ}. Call this set A^{ϕ}, since it clearly depends only on ϕ and not on the choice of t.

Then the result of $e\langle X, f\rangle$ can be represented by the set of tuples of the form (m is the column for the result of the aggregation)

$$\phi \wedge m = f(A^{\phi})$$

which is clearly finite. ∎

To make restricted aggregation practical, we must be able to determine effectively, for a generalized database $\{R_1, \ldots, R_n\}$ and a relational expression $e\langle X, f\rangle$, whether X and $U - X$ are independent in e. There are two aspects to this problem:

1. testing whether some R_i satisfies the independence restrictions on its schema
2. testing whether an arbitrary relational algebra expression e satisfies its schema restrictions.

Note that neither of the above tests needs to be performed at run-time, i.e., during query evaluation. The first test, discussed in Section 5, can be done when creating or updating R_i. The second, discussed in Section 6, can be done at compile-time, provided that all schema restrictions for $\{R_1, \ldots, R_n\}$ are satisfied.

5 Testing variable independence

As mentioned at the end of Section 4, given a relation R whose schema restricts X and Y to be independent, we must be able to test whether R satisfies the restriction. Clearly, if X and Y are independent in each tuple of R, the answer is positive. Therefore, one way of enforcing independence restrictions on R's schema is by stipulating that each tuple satisfies these restrictions.

To allow the user maximum flexibility, it is desirable to have an algorithm for testing, given an *arbitrary* R with attibute set U, and arbitrary $X, Y \subseteq U$, whether X and Y are independent in R. In this section, we provide such an *independence test* for linear constraint databases. As a side effect, this test generates on success a relation $\overline{R}$ equivalent to R where X and Y are independent for all tuples.

Unlike the work of [Las90], where all tests are performed on constraint sets (i.e., tuples), this test is for disjunctions of constraint sets (i.e., relations). The independence test consists of three steps:

Step 1. The first step of the test is to create the *boundary representation* $B(R)$ for the polyhedral object consisting of the *feasible points* of R, i.e., the points whose coordinates have values satisfying R's formula ϕ_R. This is done using standard techniques from CAD, summarized below.

Each generalized tuple t corresponds to a convex set P_t of n-dimensional points, bounded by n-dimensional half-planes. The boundary representation of P_t is computed by a convex hull algorithm, such as the "gift wrapping" method in [CK70].

$B(R)$ is obtained by *unioning* together the boundary representations of the individual tuples, a standard Solid Modeling operation. See [FvDFH] for an introduction to polyhedral Solid Modeling, or [MM] for details of the algorithms. These algorithms extend to an arbitrary number of dimensions, as in [PS86].

Step 2. The next step of the algorithm is to create a *vertex grid partitioning* $\overline{R}$ of R.

We project the set of $B(R)$'s vertices onto each of the n axes, and sort the k_j obtained values, for $1 \le j \le n$. This partitions each axis into $(k_j + 1)$ intervals:

$$\{(-\infty, v_1), (v_1, v_2), \ldots, (v_{k_j}, +\infty)\}.$$

In turn, this induces a partitioning of the n-space into $((k_1+1)\times\cdots\times(k_n+1))$ n-dimensional rectangles, where each rectangle is the cross-product of the corresponding intervals. We denote this set of rectangles by V_R, the *vertex grid* of R.

The intersection of each rectangle in V_R with $B(R)$ consists of 0 or more disjoint point sets. These point sets are convex, since no rectangle can contain a vertex of $B(R)$ as its interior point; therefore, each one corresponds to some tuple t_j. The set of these tuples, denoted by $\overline{R}$, is the *vertex grid partitioning* of R. It is equivalent to R.

Step 3. The last step of the algorithm is to check, for each t in $\overline{R}$, whether X and Y are independent in t.

The following theorem provides the motivation for the *independence test*:

Theorem 5.1 *X and Y are independent in R if and only if, for all tuples t in $\overline{R}$, X and Y are independent in t.*

Proof: (Sketch) The "if" direction of the theorem follows from definitions. For the other direction, we assume that X and Y are independent in R. Let R' be equivalent to R where X and Y are independent for each tuple $t \in R'$. Let $\overline{R}'$ be the intersection of the tuples of R' with the vertex grid of R':

$$\overline{R}' = \{t_j \cap t_k : t_j \in R', t_k \in V_{R'}\}.$$

It can be shown that X and Y are independent for each tuple in $\overline{R}'$. It can also be shown that for each tuple t in the vertex grid partitioning of R, there exists a subset of $\overline{R}'$ whose union is equivalent to t. This implies that X and Y are independent in t, completing the proof. ∎

We conclude the chapter with a short discussion of *complexity* issues. It can be shown that the data complexity of the independence test is polynomial. However, the combined complexity is exponential in n. This is due to the fact that the size of $B(R)$ can be exponential in n:

Example 5.1 Consider a relation R over n variables consisting of one tuple with the following constraints:

$$0 \le x_1 \le 1, \ldots, 0 \le x_n \le 1.$$

Its feasible points form a hypercube with 2^n vertices and $O(2^n)$ edges.

It is our opinion that the exponential combined complexity is unavoidable, since any independence test will have to somehow consider the complete geometry of the feasible points of R.

6 Inference of variable independence

In this section we show how to infer variable independence in relational algebra expressions.

In contrast to the previous section, the results in this section are applicable to any constraint language closed under negation and admitting quantifier elimination, not just linear arithmetic constraints.

Notation:

- U denotes the schema of the generalized relation R,
- $R : \mathrm{Indep}(X, Y)$ denotes the fact that X and Y are independent in R,
- $\beta(A_1, \ldots, A_k)$ denotes a constraint with free variables $A_1, \ldots, A_k$.

Theorem 6.1 *The following inference rules are valid:*

1. *if $X \subseteq U$, then $\sigma_{A=a}(R) : \mathrm{Indep}(\{A\}, X)$ and $\sigma_{A=a}(R) : \mathrm{Indep}(X, \{A\})$;*
2. *if $R : \mathrm{Indep}(X, Y)$ and $(\{A_1, \ldots, A_k\} \subseteq X$ or $\{A_1, \ldots, A_k\} \subseteq Y)$, then $\sigma_{\beta(A_1,\ldots,A_k)}(R) : \mathrm{Indep}(X, Y)$;*
3. *if $R : \mathrm{Indep}(X, Y)$ and $X, Y \subseteq Z$, then $\pi_Z(R) : \mathrm{Indep}(X, Y)$;*
4. *if $R : \mathrm{Indep}(X, Y)$ and $S : \mathrm{Indep}(X, Y)$, then $R \cup S : \mathrm{Indep}(X, Y)$;*
5. *if $R : \mathrm{Indep}(X, Y)$, then $R \times S : \mathrm{Indep}(X, Y)$ (similarly for S);*
6. *if S has schema U' disjoint from the schema of R and $X \subseteq U$ and $Y \subseteq U'$, then $R \times S : \mathrm{Indep}(X, Y)$ and $R \times S : \mathrm{Indep}(Y, X)$;*
7. *if $R : \mathrm{Indep}(X, Y)$ and $X, Y \subseteq Z$ then $R\langle Z, f\rangle : \mathrm{Indep}(X, Y)$;*
8. *if $Y \subseteq X$ and m is the new column corresponding to the result of the aggregation, then $R\langle X, f\rangle : \mathrm{Indep}(Y, \{m\})$ and $R\langle X, f\rangle : \mathrm{Indep}(\{m\}, Y)$.*

Proof: We prove the validity the second inference rule. The validity of the remaining rules can be established in a similar way.

The basic idea is as follows: given a representation for R in which X and Y are independent, we construct from it a representation for $\sigma_{\beta(A_1,\ldots,A_k)}(R)$ in which X and Y are independent. In this case, we conjoin every (generalized) tuple t of R with $\sigma_{\beta(A_1,\ldots,A_k)}(R)$ to obtain another generalized tuple t'. We have to show that

$$\pi_{XY}(t') = \pi_X(t') \bowtie \pi_Y(t').$$

The fact that the left-hand side is contained in the right-hand side follows directly from the definition of projection and join. To obtain the containment in the other direction, let p be a (ground) tuple in $\pi_X(t') \bowtie \pi_Y(t')$. Thus, there are tuples $p' \in t'$ and $p'' \in t'$ such that:

- $p'[X] = p[X]$ and $\beta(p'[A_1], \ldots, p'[A_k])$ holds,
- $p''[Y] = p[Y]$ and $\beta(p''[A_1], \ldots, p''[A_k])$ holds,
- $p'[X \cap Y] = p''[X \cap Y]$.

Now clearly $p \in \pi_X(t) \bowtie \pi_Y(t)$. Because

$$\pi_{XY}(t) = \pi_X(t) \bowtie \pi_Y(t)$$

$p \in \pi_{XY}(t)$. But also $\beta(p[A_1], \ldots, p[A_k])$ holds (notice the importance of $A_1, \ldots, A_k$ being entirely in X or Y) and thus $p \in \pi_{XY}(t')$. ∎

In general, we cannot infer variable independence in a *difference* of two relations.

Example 6.1 Consider the instance I_0 with schema $\{x, y, z\}$:

$$\begin{array}{c} y > z \\ x < y. \end{array}$$

Its complement consists of the tuple

$$z \leq y \wedge y \leq x.$$

Thus while x and z are independent in I_0, they are not independent in the complement of I_0.

For the special case of $Y = U - X$, we have:

Theorem 6.2 *The following inference rule is valid:*

- *if* $R : \mathrm{Indep}(X, U - X)$ *and* $S : \mathrm{Indep}(X, U - X)$, *then* $R - S : \mathrm{Indep}(X, U - X)$.

Proof: (sketch) Assume I_1 is an instance of R satisfying $\mathrm{Indep}(X, U - X)$ and I_2 is an instance of S satisfying $\mathrm{Indep}(X, U - X)$. Then by definition there exist instances I_1' equivalent to I_1 and I_2' equivalent to I_2 consisting only of tuples in which X and $U - X$ are independent. Let $I_1' = \{t_1, \ldots, t_n\}$ and $I_2' = \{s_1, \ldots, s_m\}$. The difference of I_1' and I_2' can thus be represented as the set of tuples

$$w_i = (t_i \wedge \neg s_1 \wedge \neg s_m)$$

for $i = 1 = 1, \ldots, n$. Let $X = \{A_1, \ldots, A_k\}$ and $U - X = \{B_1, \ldots, B_m\}$. Now every s_i, $i = 1, \ldots, m$, can in turn be represented as a tuple of the form

$$\beta(A_1, \ldots, A_k) \wedge \gamma(B_1, \ldots, B_m)$$

by Lemma 3.1. Thus its negation is of the form

$$\neg\beta(A_1, \ldots, A_k) \vee \neg\gamma(B_1, \ldots, B_m).$$

Because of this and the fact that the constraint language is closed under negation, each tuple w_i can be represented using a finite number of tuples of the form

$$\beta(A_1, \ldots, A_k) \wedge \gamma(B_1, \ldots, B_m).$$

In every such a tuple X and $U - X$ are independent. ∎

The above special case is important because the application $e\,\langle X, f\rangle$ of a restricted aggregation operator in the version of relational algebra discussed in section 4 is allowed only if $e : \mathrm{Indep}(X, U - X)$ where U is the schema of e.

Remark: A tuple in a finite instance of relational schema $R(A_1, \ldots, A_n)$ may be viewed as a generalized tuple

$$\bigwedge_{i=1,\ldots,n} A_i = a_i.$$

Therefore, in every such instance every two disjoint subsets of the schema are independent. The result of any relational expression applied to such an instance can also be represented in this form and variable independence is preserved. Our rules are too weak to make this kind of inference, c.f.. the rule for $\sigma_{A_k = A_m}(R)$. This is because of their generality. Thus, there is clearly a need for developing specialized inference rules that will exploit the properties of restricted constraint languages.

7 Related and further work

In the context of constraint databases, we have defined a restricted version of aggregation whose addition to relational algebra results in a closed language. The restriction requires that the set of variables grouped upon is independent of the remaining variables. We formalized this restriction using the notion of *variable independence.* We have shown that for constraint databases with linear arithmetic constraints variable independence is decidable with acceptable complexity. We have also provided a set of rules for inferring variable independence in relational expressions.

Previous work on aggregation, with the exception of [Klu82, OOM87, Kup94, CK95], has concentrated on a fixed set of aggregation operators, e.g., `sum` or `count`, motivated by traditional database applications. Such operators are applicable to finite relations and do not generalize to infinite ones. In [CK95], a special aggregation operator for generalized relations that preserved closure of relational algebra operations was proposed. The issue of closure of general aggregation operators was not, however, addressed.

The idea of variable independence can be potentially applied beyond the context of aggregation. [Rev95] defined a restricted form of Datalog with negation and integer gap-order constraints. The restriction required essentially that all attributes be independent in *every generalized tuple* in order to guarantee the termination of bottom-up query evaluation. As Figure 1 shows, a generalized relation containing tuples in which not all attributes are independent can still be equivalent to one in which in all the tuples all attributes are independent. If the technique we proposed in section 5 can be generalized to the integer case, it will provide a tool to enlarge the class of terminating Datalog programs.

Other possible extensions of this work include: larger classes of constraints (e.g., polynomial constraints), other query languages with aggregation (e.g., relational calculus, Datalog [MPR90, RS92, SSRB93, VG92]), properties of variable independence considered as a dependency class (axiomatization, implication).

References

[AS91] W.G. Aref and H. Samet. Extending a DBMS with Spatial Operations. In *International Symposium on Large Spatial Databases*, pages 299–318, 1991.

[BJM93] A. Brodsky, J. Jaffar, and M.J. Maher. Towards Practical Constraint Databases. In *International Conference on Very Large Data Bases*, Dublin, Ireland, 1993.

[BK95] A. Brodsky and Y. Kornatzky. The LyriC Language: Constraining Objects. In *ACM SIGMOD International Conference on Management of Data*, San Jose, California, 1995.

[BLLM95] A. Brodsky, C. Lassez, J-L. Lassez, and M.J. Maher. Separability of Polyhedra for Optimal Filtering of Spatial and Constraint Data. In *ACM Symposium on Principles of Database Systems*, San Jose, California, 1995.

[Cho94] J. Chomicki. Temporal Query Languages: A Survey. In D.M. Gabbay and H.J. Ohlbach, editors, *Temporal Logic, First International Conference*, pages 506–534. Springer-Verlag, LNAI 827, 1994.

[CK70] D.R.Chand, S.S. Kapur. An Algorithm for Convex Polytopes. *JACM*, 17(1): 78–86, 1970.

[CK95] J. Chomicki and G. Kuper. Measuring Infinite Relations. In *ACM Symposium on Principles of Database Systems*, 1995.

[FvDFH] J.D.Foley, A. van Dam, S.K.Feiner, J.F. Hughes *Computer Graphics, Principles and Practice.* Addison-Wesley, 1990.

[GK94] P. Gritzmann and V. Klee. On the Complexity of Some Basic Problems in Computational Convexity: II. Volume and Mixed Volumes. Technical Report TR-94-31, DIMACS, Rutgers University, New Brunswick, NJ, 1994.

[Kan90] P.C. Kanellakis. Elements of Relational Database Theory. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 17, pages 1073–1158. Elsevier/MIT Press, 1990.

[Kan95] P.C. Kanellakis. Constraint Programming and Database Languages: A Tutorial. In *ACM Symposium on Principles of Database Systems*, 1995.

[KG94] P. C. Kanellakis and D. Q. Goldin. Constraint programming and database query languages. In *Proc. 2nd Conference on Theoretical Aspects of Computer Software (TACS)*, April 1994.

[KG96] P.C. Kanellakis and D.Q. Goldin Constraint Query Algebras *Constraints Journal*, 1st issue, 1996 (to be published).

[KKR90] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint Query Languages. *JCSS*, 51(1): 26–52, 1995.

[Klu82] A. Klug. Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. *Journal of the ACM*, 29(3):699–717, 1982.

[Kup94] G. Kuper. Aggregation in Constraint Databases. In *PPCP'93, First International Workshop on Principles and Practice of Constraint Programming*, pages 161–172. MIT Press, 1994.

[Las90] J.L. Lassez. Querying Constraints. In *9th ACM Symposium on Principles of Database Systems*, pages 288–298, 1990.

[MM] Martti Mantyla. *Solid Modeling.* Computer Science Press, 1988.

[MPR90] I.S. Mumick, H. Pirahesh, and R. Ramakrishnan. Duplicates and Aggregates in Deductive Databases. In *International Conference on Very Large Data Bases*, August 1990.

[OOM87] G. Ozsoyoglu, Z.M. Ozsoyoglu, and V. Matos. Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Functions. *ACM Transactions on Database Systems*, 12:566–592, 1987.

[PBCF93] A.Paoluzzi, F.Bernandini, C.Cattani, V.Ferrucci. Dimension-Independent Modeling with Simplicial Complexes *ACM Trans. Graphics*, 12(1): 56–102, 1993.

[PVdBVG94] J. Paredaens, J. Van den Bussche, and D. Van Gucht. Towards a Theory of Spatial Database Queries. In *ACM Symposium on Principles of Database Systems*, pages 279–288, Minneapolis, Minnesota, 1994.

[PS86] L.K. Putnam, P.A. Subrahmanyam. Boolean Operations on n-dimensional objects. *IEEE Comput. Graph. Appl.*, 6(6): 43–51, 1986.

[Rev95] P. Z. Revesz. Safe Stratified Datalog with Integer Order Programs. In *International Conference on Constraint Programming*, Marseilles, France, September 1995. Springer-Verlag, LNCS 1000.

[RS92] K. A. Ross and Y. Sagiv. Monotonic Aggregation in Deductive Databases. In *ACM Symposium on Principles of Database Systems*, pages 114–126, 1992.

[SSRB93] S. Sudarshan, D. Srivastava, R. Ramakrishnan, and C. Beeri. Extending the Well-Founded and Valid Model Semantics for Aggregation. In *International Logic Programming Symposium*, 1993.

[VG92] A. Van Gelder. The Well-Founded Semantics of Aggregation. In *ACM Symposium on Principles of Database Systems*, pages 127–138, San Diego, California, June 1992.

[VGVG95] L. Vandeurzen, M. Gyssens, and D. Van Gucht. On the Desirability and Limitations of Linear Spatial Database Models. In *International Symposium on Large Spatial Databases*, pages 14–28, 1995.