Equivalences Among Aggregate Queries with Negation*

Sara Cohen
Institute of Computer Science
The Hebrew University
Jerusalem 91904, Israel
sarina@cs.huji.ac.il

Werner Nutt
Department of Computing
and Electrical Engineering
Heriot-Watt University,
Edinburgh EH14 4AS
Scotland, UK
nutt@cee.hw.ac.uk

Yehoshua Sagiv
Institute of Computer Science
The Hebrew University
Jerusalem 91904, Israel
sagiv@cs.huji.ac.il

ABSTRACT

Query equivalence is investigated for disjunctive aggregate queries with negated subgoals, constants and comparisons. A full characterization of equivalence is given for the aggregation functions *count*, *max*, *sum*, *prod*, *top*2 and *parity*. A related problem is that of determining, for a given natural number N, whether two given queries are equivalent over all databases with at most N constants. We call this problem *bounded equivalence.* A complete characterization of decidability of bounded equivalence is given. In particular, it is shown that this problem is decidable for all the above aggregation functions as well as for *cntd* (count distinct), *stdev* (standard deviation), *median* and *avg*. For quasilinear queries (i.e., queries in which predicates that occur positively are not repeated) it is shown that equivalence can be decided in polynomial time for the aggregation functions *count*, *max*, *sum*, *prod*, *top*2, *parity*, and *avg*. A similar result holds for *cntd* provided that a few additional conditions hold. The results are couched in terms of abstract characteristics of aggregation functions, and new proof techniques are used. Finally, the results presented also imply that equivalence, under bag-set semantics, is decidable for nonaggregate queries with negation.

1. INTRODUCTION

The emergence of data warehouses and of decision-support systems has highlighted the importance of efficiently processing aggregate queries. In such systems the amount of data is generally large and aggregate queries are used as a standard means of reducing the volume of the data. Aggregate queries tend to be expensive as they "touch" many items while returning few. Thus, optimization techniques for aggregate queries are a necessity. Many optimization techniques, such as query rewriting, are based on checking query equivalence. For this purpose, a coherent understanding of the equivalence problem of aggregate queries is necessary.

One of our main results in this paper is that equivalence is decidable for disjunctive queries with comparisons and negated subgoals if they contain one of the aggregation functions *max*, *top*2, *count*, *sum*, *prod*, or *parity*.

A query that does not have negated subgoals is *positive*. Equivalence of positive non-aggregate queries has been studied extensively [2, 1, 13, 8, 15, 11]. Furthermore, in [10] it has been shown that equivalence is decidable for non-aggregate disjunctive queries with negation. Syntactic characterizations of equivalences among aggregate queries with the functions *max*, *sum*, and *count* have been given in [12, 4]. These results have been extended in [5] to queries with the functions *prod* and *avg*, for the special case of queries that contain neither constants nor comparisons. Thus, there exist results on the equivalence problem for both, *non*-aggregate queries *with* negation as well as *aggregate* queries *without* negation. However, to the best of our knowledge nothing is known about the equivalence of *aggregate* queries *with* negated subgoals.

Our decidability proofs rely on abstract properties of aggregation functions. We consider functions that are defined by means of operations on abelian monoids. Our proofs work out if the monoids are either idempotent or are groups. Functions of the first kind are *max* and *top*2, functions of the second kind are *count*, *sum*, *prod*, and *parity*.

For these functions we reduce equivalence with respect to all possible databases to equivalence over databases that have at most as many constants as there are constants and variables in the queries, a property which we call *local equivalence.* We do not study local equivalence immediately, but the more general problem of *bounded equivalence.* It consists in determining, given a nonnegative integer N and two queries, whether the queries return identical results over all databases with at most N constants. We give a complete characterization of decidability of bounded equivalence. In particular, we show that bounded equivalence is decidable for queries with the functions *count*, *cntd*, *max*, *sum*, *prod*, *avg*, *median*, *stdev*, *top*2 and *parity*.

*This work was supported in part by grant 9481-3-00 of the Israeli Ministry of Science.

Finally, we consider the special case of *quasilinear queries*, that is, queries where predicates that occur positively are not repeated. For quasilinear queries equivalence boils down to isomorphism, which can be decided in polynomial time.

2. AGGREGATION FUNCTIONS

An aggregate query is executed in two steps. First, data is collected from a database as specified by the non-aggregate part of the query. Then the results are grouped into multisets (or bags), aggregation functions are applied to the multisets, and the aggregates are returned as answers.

The queries that we consider in this paper contain the aggregation functions *count* and *cntd*, which for a bag return the number of elements or distinct elements, respectively; *parity*, which returns 0 or 1, depending on whether the number of elements in the bag is even or odd; *sum*, *prod* and *avg*, which return the sum, product, or average of the elements of a bag; *max*, which returns the maximum among the elements of a bag; *median*, which returns the median among the elements of a bag; *stdev*, which returns the standard deviation of the elements of a bag and *top*2, which returns a pair, consisting of the two greatest different elements of a bag.

The reader will notice in the course of the paper that our results for *max* and *top*2 immediately carry over to *min* and *bot*2, which select the minimum or the two least elements out of a multiset of numbers. Moreover, our results for *top*2 can easily be generalized to the function *top*K, which selects the K greatest different elements.

Our arguments to prove decidability of equivalence for certain classes of aggregate queries rely on the fact that the aggregation functions take values in special kinds of abelian monoids and are defined in terms of the operations of those monoids. To make this formal, we will introduce the class of *monoidal aggregation functions* and two of its subclasses. We will show that all of the above functions except *cntd*, *stdev*, *median* and *avg* belong to one of these two subclasses.

In general, an *aggregation function* maps multisets of tuples of numbers to values in some structure, which in most cases consists again of numbers. Here, we assume that the results of the aggregation are elements of some abelian monoid.

An *abelian monoid* is a set M with an associative and commutative binary operation, which we denote as "+", and a neutral element, which we denote as 0. An abelian monoid M is *idempotent* if $a + a = a$ holds for all $a \in M$, and M is a *group* if for every a there is a b such that $a + b = 0$. The element b is called the *inverse* of a and is usually denoted as $-a$. Instead of $a + (-b)$ we will usually write $a - b$.

EXAMPLES 2.1. Standard abelian monoids are the set of integers $\mathbf{Z}$ and the set of rational numbers $\mathbf{Q}$, with the binary operations of addition or multiplication, and the neutral elements 0 or 1, respectively. A further example is the two element group $\mathbf{Z}_2 = \{0, 1\}$, where addition satisfies $1 + 1 = 0$.

By $\mathbf{Q}_\perp$ we denote the rational numbers augmented by a new element $\perp$, which is less than any element in $\mathbf{Q}$. Then $\mathbf{Q}_\perp$ is an abelian monoid if the operation is selecting the maximum of two numbers. The neutral element is $\perp$.

A less common example is the monoid $\mathbf{T}_2$, which is defined as the set of pairs

$$\mathbf{T}_2 := \{(d, e) \in \mathbf{Q}_\perp \times \mathbf{Q}_\perp \mid d > e\},$$

with the binary operation "$\oplus$", where $(d_1, e_1) \oplus (d_2, e_2)$ is defined as the pair (d, e) that consists of the two greatest different elements among $\{d_1, e_1, d_2, e_2\}$. For instance, we have $(5, \perp) \oplus (2, 1) = (5, 2)$ and $(5, 2) \oplus (5, 1) = (5, 2)$. □

If $(M, +)$ is an abelian monoid, we can extend the binary operation to subsets of M and to multisets over M in a canonical way—because of the associativity and commutativity of "+", the order in which we apply the operation does not matter. If M' is such a set or multiset, we denote the result of applying "+" to M' as $\sum_{a \in M'} a$.

Many common aggregation functions are computed by first mapping the elements of a multiset into an abelian monoid and then combining the values obtained through the mapping by the monoid operation. Technically, we assume that there is a set S, a monoid M with operation "+", and a function $f\colon S \to M$. Then the *aggregation function* based on f and "+", which maps multisets B over S to elements of M, is denoted as α_f^+ and defined by

$$\alpha_f^+(B) := \sum_{a \in B} f(a),$$

for all bags B over S. We say that α is a *monoid aggregation function* if $\alpha = \alpha_f^+$ for some abelian monoid operation "+". In particular, we say that α_f^+ is *idempotent* or a *group aggregation function* if the underlying monoid is idempotent, or a group, respectively.

EXAMPLES 2.2. Obviously, *sum*, *prod*, and *max* are the aggregation functions based on the identity mapping and on addition, multiplication, or the binary operation "max", respectively. The functions *count* and *parity* arise from the additive groups $\mathbf{Z}$ and $\mathbf{Z}_2$ by choosing as f the mapping that maps every element to the constant 1. We obtain the aggregate *top*2 over the rationals by choosing the monoid $\mathbf{T}_2$ and the mapping $f\colon \mathbf{Q} \to \mathbf{T}_2$ defined by $f(a) := (a, \perp)$.

Note that *sum*, *prod*, *count* and *parity* are group aggregation functions, while *max* and *top*2 are idempotent. However, one can prove that *cntd*, *stdev*, *median* and *avg* are not monoid aggregation functions. □

3. DISJUNCTIVE AGGREGATE AND NON-AGGREGATE QUERIES

We now introduce conjunctive and disjunctive queries with negated subgoals and review their basic properties. We use standard Datalog syntax extended by aggregation functions. We thus give an abstract representation of unnested aggregate queries as they are definable in SQL without using the `HAVING` construct.

3.1 Syntax of Non-aggregate Queries

Predicate symbols are denoted as p, q, r. A *term,* denoted as s, t, is either a variable or a constant. A *relational atom* has the form $p(s_1, \ldots, s_k)$, where p is a predicate of arity k. We also use the notation $p(\bar{s})$, where $\bar{s}$ stands for a tuple of terms $(s_1, \ldots, s_k)$. Similarly, $\bar{x}$ stands for a tuple of variables. An *ordering atom* or *comparison* has the form $s_1 \ \rho \ s_2$, where ρ is one of the ordering predicates $<$, $\leq$, $>$, $\geq$, or $\neq$. A relational atom can be *negated.* A relational atom that is not negated is *positive.* A literal is a positive relational atom, a negated relational atom, or a comparison. A *condition,* denoted as L, is a conjunction of literals. We assume that conditions are safe [14].

A *query* is a non-recursive expression of the form

$$q(\bar{x}) \leftarrow L_1 \vee \ldots \vee L_n,$$

where each L_i is a condition containing all the variables appearing in the tuple $\bar{x}$. The variables that occur in the head, i.e. in $\bar{x}$, are the *distinguished* variables of the query. Those that occur only in the body are the *nondistinguished* variables.

A query is *conjunctive* if it contains only one disjunct. A query is *positive* if it does not contain any negated relational atoms. By abuse of notation, we will often refer to a query by its head $q(\bar{x})$ or simply by the predicate of its head q.

3.2 Semantics of Non-aggregate Queries

Databases are sets of ground relational atoms and are denoted by the letter $\mathcal{D}$. The carrier of $\mathcal{D}$, written $carr(\mathcal{D})$, is the set of constants occurring in $\mathcal{D}$. We define in which way a query q, evaluated over a database $\mathcal{D}$, gives rise to a set of tuples $q^{\mathcal{D}}$.

An *assignment* γ for a condition L is a mapping of the variables appearing in L to constants, and of the constants appearing in L to themselves. Assignments are naturally extended to tuples and atoms. *Satisfaction* of atoms and of conjunctions of atoms by an assignment with respect to a database are defined in the obvious way. For $\bar{s} = (s_1, \ldots, s_k)$ we let $\gamma\bar{s}$ denote the tuple $(\gamma(s_1), \ldots, \gamma(s_k))$.

For the interpretation of comparisons it makes a difference whether they range over a dense order, like the rational numbers, or a discrete order, like the integers. A conjunction of comparisons, like $0 < y < z < 2$, may be satisfiable over the rationals, but not over the integers. If, as in most cases, a result in this paper holds for comparisons over both, integers and rationals, we will not mention this explicitly. However, often two different proofs are needed to establish such a result.

A query $q(\bar{x}) \leftarrow L_1 \vee \ldots \vee L_n$ defines a new relation $q^{\mathcal{D}}$, for a given database $\mathcal{D}$, as

$$\bigcup_{i=1}^{n} \{\gamma\bar{x} \mid \gamma \text{ satisfies } L_i \text{ with respect to } \mathcal{D}\}. \qquad (1)$$

Chaudhuri and Vardi [3] have introduced *bag-set semantics,* which records the multiplicity with which a tuple occurs as an answer to the query. The definition in (1) can be turned into one for bag-set semantics by replacing set braces by multisets and set union by multiset union. *Bag semantics* [3] differs from bag-set semantics in that both, database relations and relations created by queries, are multisets of tuples.

3.3 Syntax of Aggregate Queries

In [12, 4] we have shown that equivalence of positive disjunctive queries with several aggregate terms can be reduced to equivalence of queries with a single aggregate term. This still holds if the queries can contain negated subgoals. For this reason, we consider in the present paper only queries having a single aggregate term in the head. We give a formal definition of the syntax of such queries.

An *aggregate term* is an expression built up using variables and an aggregation function. For example *count* and $sum(y)$ are aggregate terms. We use $\alpha(\bar{y})$ as an abstract notation for aggregate terms. Note that $\bar{y}$ can be the empty tuple as in the case of the function *count*.

An *aggregate query* is a query augmented by an aggregate term in its head. Thus it has the form

$$q(\bar{x}, \alpha(\bar{y})) \leftarrow L_1 \vee \ldots \vee L_n. \qquad (2)$$

In addition, we require that

- $\alpha(\bar{y})$ is an aggregate term;
- no variable $x \in \bar{x}$ occurs in $\bar{y}$;
- each condition L_i contains all the variables in $\bar{x}$ and in $\bar{y}$.

We call $\bar{x}$ the *grouping terms* of the query. If the aggregation term in the head of a query has the form $\alpha(\bar{y})$, we call the query an α*-query* (e.g., a *max*-query).

3.4 Semantics of Aggregate Queries

Consider an aggregate query q as in Equation (2). We define how, for a database $\mathcal{D}$, the query yields a new relation $q^{\mathcal{D}}$. We proceed in two steps.

We denote the set of assignments γ over $\mathcal{D}$ that satisfy one of the disjuncts L_i in the body of q as $\Gamma(q, \mathcal{D})$. We assume that such a γ is defined only for the variables that occur in L_i. Moreover, if an assignment γ satisfies two or more disjuncts, we want it to be included as many times in $\Gamma(q, \mathcal{D})$ as there are disjuncts it satisfies. To achieve this, we assume that there are as many copies of γ in $\Gamma(q, \mathcal{D})$ as there are disjuncts that γ satisfies, and that each copy carries a label indicating which disjunct it satisfies.[1]

Recall that $\bar{x}$ are the grouping terms of q and $\bar{y}$ are the aggregation variables. For a tuple $\bar{d}$, let $\Gamma_{\bar{d}}(q, \mathcal{D})$ be the subset of $\Gamma(q, \mathcal{D})$ consisting of labeled assignments γ with $\gamma\bar{x} = \bar{d}$. In the sets $\Gamma_{\bar{d}}(q, \mathcal{D})$, we group those satisfying

[1] We could make this more formal by defining $\Gamma(q, \mathcal{D})$ to consist of pairs (γ, i), where γ is an (ordinary) assignment and i the index of a condition that it satisfies. However, to avoid charging our notation with too much detail, we prefer to introduce the concept of "labeled assignments" informally.

assignments that agree on $\bar{x}$. Therefore, we call $\Gamma_{\bar{d}}(q, \mathcal{D})$ the *group* of $\bar{d}$.

Let A be a set of labeled assignments and $\bar{y}$ be a tuple of variables for which the elements of A are defined. Then we define the *restriction of* A to $\bar{y}$ as the multiset

$$A_{|\bar{y}} := \{\!\{\gamma\bar{y} \mid \gamma \in A\}\!\}.$$

We can apply the restriction operator to $\Gamma_{\bar{d}}(q, \mathcal{D})$. If $\alpha(\bar{y})$ is an aggregation function, we can apply α to the the multiset $A_{|\bar{y}}$, which results in the aggregate value $\alpha(A_{|\bar{y}})$. As an alternative notation, we define

$$\alpha(\bar{y}) \downarrow A := \alpha(A_{|\bar{y}}).$$

Now we define the result of evaluating $q(\bar{x}, \alpha(\bar{y}))$ over $\mathcal{D}$, denoted $q^{\mathcal{D}}$ as

$$\left\{(\bar{d}, \alpha(\bar{y}) \downarrow \Gamma_{\bar{d}}(q, \mathcal{D})) \;\middle|\; \bar{d} = \gamma\bar{x} \text{ for some } \gamma \in \Gamma(q, \mathcal{D})\right\}.$$

Similarly as for non-aggregate queries, $q^{\mathcal{D}}$ is a set of tuples.

3.5 Equivalence

Two queries q and q', aggregate or non-aggregate, are *equivalent* if over every database they return the same sets of results, that is, if $q^{\mathcal{D}} = {q'}^{\mathcal{D}}$ for all databases $\mathcal{D}$. For positive non-aggregate queries, equivalence is decidable and has been characterized in terms of the existence of query homomorphisms [2, 13, 8]. Levy and Sagiv have shown that equivalence is still decidable for disjunctive queries with negated atoms [11].

In [12, 4], we have proved decidable characterizations for the equivalence of positive conjunctive and disjunctive aggregate queries with the operators *max*, *count*, and *sum*. Note that two non-aggregate queries $q(\bar{x})$, $q'(\bar{x})$ are equivalent under bag-set semantics if and only if the *count*-queries $q(\bar{x}, count)$, $q'(\bar{x}, count)$ are equivalent. Thus, characterizations of the equivalence of *count*-queries immediately yield criteria for non-aggregate queries to be equivalent under bag-set semantics.

4. BOUNDED EQUIVALENCE

Our goal is to to reduce the problem of deciding equivalence of two aggregate queries over all possible databases to the problem of deciding *local equivalence,* that is, equivalence over databases containing no more constants than the size of the queries. In this section, we present the conditions necessary for the more general *bounded equivalence problem* to be decidable. We also show decidability of this problem for a wide range of aggregate queries.

Let N be a nonnegative integer. We say that two queries q, q' are *N-equivalent,* denoted $q \equiv_N q'$, if for all databases $\mathcal{D}$ whose carrier has at most N elements, we have $q^{\mathcal{D}} = {q'}^{\mathcal{D}}$.

The *bounded equivalence problem* for a class of queries is to decide, given $N > 0$ and queries q, q' from that class, whether $q \equiv_N q'$.

Let L be a condition. Then the *variable size* of L is number of variables in L. Let q be a disjunctive query. The *variable size* of q is the maximum of the variable sizes of the conditions in q. If a query contains an equality $y = z$, it does not matter if the variables are counted once or twice for the proofs later on.

The *term size* of a query is the total number of constants occurring in that query plus the variable size. The *term size* of a pair of queries q, q' is the total number of constants occurring in at least one of q or q' plus the maximum of the variable sizes of q, q'. We denote the term size of q as $\tau(q)$ and the term size of q, q' as $\tau(q, q')$. We say that two queries q, q' are *locally equivalent,* if $q^{\mathcal{D}} \equiv_{\tau(q,q')} {q'}^{\mathcal{D}}$, that is, if q and q' return the same results over all databases with at most $\tau(q, q')$ constants.

Clearly, two queries are equivalent if and only if they are N-equivalent for all $N > 0$. However, the decidability of bounded equivalence for a class of queries does not necessarily imply that equivalence is decidable. Sections 5 and 6 establish criteria for this implication to hold. Moreover, decidability of N-equivalence, for a fixed N, does not imply decidability of local equivalence, since in the latter problem the size of the databases to be tested depends on the size of the queries.

PROPOSITION 4.1. *If the bounded equivalence problem is decidable for a class of queries, then local equivalence is decidable, too.*

In the rest of this section we study the decidability of the bounded equivalence problem for several aggregation functions. Note that N-equivalence is not necessarily a trivial property. Even if the size of databases is bounded, there are still infinitely many databases whose size is below the bound, and the aggregation results may well depend on the values of the constants in a database.

We introduce the notion of shiftable aggregation functions and of order-decidable aggregation functions. It is shown that shiftable aggregation functions are a special case of order-decidable aggregation functions. We then show that bounded equivalence is decidable exactly for queries with order-decidable aggregation functions.

4.1 Shiftable Aggregation Functions

We introduce the notion of *shiftable aggregation functions.* Intuitively, the value of such a function does not depend on the specific values in a multiset, but only on the ordering of the elements. Let D and D' be sets of values. A function $\varphi\colon D \to D'$ is a *shifting function* if for all d, $d' \in D$ we have

$$d < d' \;\Rightarrow\; \varphi(d) < \varphi(d').$$

A shifting function is extended to bags as one would expect. An aggregation function α is *shiftable* if for all shifting functions $\varphi\colon D \to D'$, where D' is an arbitrary ordered domain over which α is defined, and for all bags B, B' with elements in D, we have

$$\alpha(B) = \alpha(B') \iff \alpha(\varphi(B)) = \alpha(\varphi(B')).$$

PROPOSITION 4.2. *The aggregation functions median, parity, cntd, count, max and top2 are shiftable.*

Note, however, that the aggregation functions *sum* and *prod* are not shiftable. For example, consider the bags $B = \{\!\{2, 2\}\!\}$ and $B' = \{\!\{4\}\!\}$ and suppose φ is a shifting function with $\varphi(2) = 3$ and $\varphi(4) = 5$. Then $sum(B) = sum(B') = prod(B) = prod(B') = 4$, while neither *sum* nor *prod* agree on $\varphi(B) = \{\!\{3, 3\}\!\}$ and $\varphi(B') = \{\!\{5\}\!\}$.

4.2 Order-Decidable Aggregation Functions

Before defining order-decidable aggregation functions, we present some auxiliary definitions. A *domain,* denoted $\mathcal{I}$ is a set of ordered values. For example, the integers $\mathbf{Z}$ and the rationals $\mathbf{Q}$ are domains.

We say that a conjunction of ordering atoms L is a *complete ordering* of a set of terms T *with respect to a domain* $\mathcal{I}$ if for every two terms $t, t' \in T$, exactly one of the following holds:

- $L \models t < t'$;
- $L \models t > t'$;
- $L \models t = t'$

over $\mathcal{I}$. Note that by definition, complete orderings are satisfiable.

If B is a bag of terms, we denote the variables appearing in B by $v(B)$. Similarly, we denote the variables in B or in B' by $v(B, B')$.

Let α be an aggregation function and let $\mathcal{I}$ be a domain. We say that α is *order-decidable over* $\mathcal{I}$ if for all nonempty bags B and B' and all complete orderings L of the terms of B and B' with respect to $\mathcal{I}$ it is possible to decide if the formula

$$\Big(\bigvee_{u \in v(B,B')} u\Big)\big(L \rightarrow \alpha(B) = \alpha(B')\big) \tag{3}$$

is true over $\mathcal{I}$. Note that the truth value of the formula may be dependent on $\mathcal{I}$.

Intuitively, Formula 3 states that it is possible to decide if the value of α when applied to B and to B' is equal, for all *instantiations* δ, that satisfy L, of the variables in B and B'.

EXAMPLE 4.3. It is easy to see that the function *parity* is order-decidable over any domain. Consider, for example, the bags $\{\!\{1, 2, u\}\!\}$ and $\{\!\{v, v, 7\}\!\}$ and an arbitrary complete ordering L of $\{1, 2, u, v, 7\}$. It is possible to decide the equation

$$(\forall u)(\forall v)\big(L \rightarrow parity(\{\!\{1, 2, u\}\!\}) = parity(\{\!\{v, v, 7\}\!\})\big)$$

since the parity of both bags is odd regardless of the values assigned to u and v. □

It is not by chance that the function *parity* is order-decidable over all domains. It is actually a consequence of the fact that *parity* is a shiftable aggregation function.

THEOREM 4.4. *Let α be a shiftable aggregation function and let $\mathcal{I}$ be a domain. Then α is order-decidable over $\mathcal{I}$.*

PROOF. Suppose that α is a shiftable aggregation function and that $\mathcal{I}$ is a domain. Let B and B' be bags over $\mathcal{I}$ with terms T. Let L be a complete ordering of T with respect to $\mathcal{I}$. Let $\delta\colon T \rightarrow \mathcal{I}$ be a function. We say that δ *satisfies* L if the following conditions hold:

- δ maps all constants to themselves;
- for all $t, t' \in T$ and all ordering predicates ρ it holds that

$$t \;\rho\; t' \iff \delta(t) \;\rho\; \delta(t').$$

Consider Formula 3. Since L is a complete ordering, L is also satisfiable with respect to $\mathcal{I}$. We show that if $\alpha(\delta(B)) = \alpha(\delta(B'))$ for some δ that satisfies L, then $\alpha(\delta'(B)) = \alpha(\delta'(B'))$ for any δ' satisfying L. From this, it follows that to test Formula 3 it is enough to generate one example δ and to check the formula for it.

Now, let δ, $\delta'\colon T \rightarrow \mathcal{I}$ be two arbitrary assignments that satisfy L. We assume without loss of generality that there are no two different terms t_1, $t_2 \in T$ for which $L \models t_1 = t_2$. (If there were such terms we could remove one of them by renaming.) Hence, δ and δ' are injections. Thus, the function $\delta' \circ \delta^{-1}$ is well defined.

Since both δ and δ' preserve order, $\delta' \circ \delta^{-1}$ is a shifting function. Thus, $\alpha(\delta(B)) = \alpha(\delta(B'))$ implies $\alpha(\delta'(B)) = \alpha(\delta(B'))$, as required. □

The other direction of Theorem 4.4 does not hold. An aggregation function can be order-decidable over a given domain even if it is not shiftable.

PROPOSITION 4.5. *The aggregation functions sum, prod, avg and stdev are order-decidable over* $\mathbf{Z}$ *and over* $\mathbf{Q}$.

PROOF. This follows directly from the fact that the universal theories of the structures

- $\langle \mathbf{Z}, (+, \cdot, 0, 1, <) \rangle$
- $\langle \mathbf{Q}, (+, \cdot, 0, 1, <) \rangle$

are decidable (cf. [9, pages 24–27]). □

4.3 Decidability of Bounded Equivalence

It is possible to show that if an aggregation function α is order-decidable over a domain $\mathcal{I}$ then bounded equivalence can be decided for α-queries containing comparisons that range over $\mathcal{I}$. Actually, bounded equivalence for α-queries ranging over $\mathcal{I}$ is decidable *if and only if* α is order-decidable

over $\mathcal{I}$. This gives a complete characterization of decidability of bounded-equivalence of aggregate queries with negation, disjunction, constants and comparisons. In addition, we derive as a direct result that bounded equivalence is decidable for queries with a wide range of common aggregation functions.

THEOREM 4.6. *Let α be an aggregation function and let $\mathcal{I}$ be a domain. Then the bounded equivalence problem is decidable for disjunctive α-queries with comparisons ranging over $\mathcal{I}$ if and only if α is order-decidable over $\mathcal{I}$.*

Proof (Sketch). "$\Leftarrow$" Consider α-queries q and q'. Suppose that α is order-decidable over $\mathcal{I}$. We show it is possible to check if $q \equiv_N q'$, for a given N. Let C be the set of constant values appearing in q or q' and let U be a set of N variables. Let P be the set of predicates appearing either in q or in q'. The set P contains predicates that appear either positively or negatively in the queries. We use $ary(p)$ to denote the arity of a predicate $p \in P$.

We denote by BASE the set of all atoms that can be created using the terms in $C \cup U$ and the predicates in P. Formally,

$$\mathsf{BASE} := \{p(t_1, \ldots, t_{ary(p)}) \mid p \in P \text{ and } t_1, \ldots, t_{ary(p)} \in T\}.$$

An *instantiation* of a set $S \subseteq$ BASE is a function δ from the variables in S to $\mathcal{I}$. Instantiations define databases. It is possible to show that it is sufficient to consider databases that can be derived by instantiating subsets of BASE.

There are infinitely many instantiations of a given subset S of BASE. Thus, we do not consider each instantiation directly. Instead, for each complete ordering L of $C \cup U$ with respect to $\mathcal{I}$, we consider instantiations of subsets of BASE that satisfy L.

Consider a subset S of BASE and a complete ordering L of $C \cup U$. We may attempt to evaluate q and q' over S where the ordering of its terms is defined by L. We can verify that q and q' return the same tuples of terms for their grouping variables. However, it is not possible to compute the values of the aggregation function in the queries since S contains variables. We can compute the bag of terms assigned to the aggregation variables for a given assignment of values to the grouping variables. For an assignment of tuple $\bar{t}$ to the grouping variables, we denote the bag of values assigned to the aggregation variables of q (q') as $B_{\bar{t}}$ ($B'_{\bar{t}}$).

One can show there is an instantiation δ of S that satisfies L and defines a database $\delta(S)$ over which q and q' do not return the same aggregation value for $\delta\bar{t}$ if and only if the formula

$$(\bigforall_{u \in U} u)\,(L \rightarrow \alpha(B_{\bar{t}}) = \alpha(B'_{\bar{t}}))$$

is false. Since α is order-decidable over $\mathcal{I}$, the truth value of the formula can be determined.

"$\Rightarrow$" We show that if bounded equivalence of α-queries ranging over $\mathcal{I}$ is decidable, then α is order-decidable over $\mathcal{I}$. Let B and B' be bags and let L be a complete ordering of the terms in B and B'.

Let $T = \{t_1, \ldots, t_N\}$ be the set of terms in B and B'. We define the condition A as

$$A := p(t_1)\ \&\ p(t_2)\ \&\ \ldots\ \&\ p(t_N).$$

Suppose that $B = \{\!\{s_1, \ldots, s_n\}\!\}$ and $B' = \{\!\{s'_1, \ldots, s'_m\}\!\}$. We assume that the variable y does not appear in B or B'. We define conditions

$$L_i := A\ \&\ L\ \&\ y = s_i \qquad i = 1 \ldots n$$
$$L'_j := A\ \&\ L\ \&\ y = s'_j \qquad j = 1 \ldots m$$

Let q and q' be the queries

$$q(\alpha(y)) \leftarrow \bigvee_{i=1}^{n} L_i$$
$$q'(\alpha(y)) \leftarrow \bigvee_{j=1}^{m} L'_j$$

It is possible to show that $q \equiv_N q'$ if and only if Formula 3 is true for the bags B, B' and the complete ordering L. □

COROLLARY 4.7. *Let α be a shiftable aggregation function and let $\mathcal{I}$ be a domain. Then the bounded equivalence problem is decidable for disjunctive α-queries, provided that the comparisons range over $\mathcal{I}$.*

COROLLARY 4.8. *For the classes of disjunctive max, sum, prod, avg, cntd, count, parity, median, stdev, and top2 queries, bounded equivalence is decidable provided that the comparisons range over* **Q** *or* **Z**.

In Theorem 4.6 we reduced order-decidability to bounded equivalence. A point of interest is that in our reduction we only used positive queries. Therefore, negation in queries q and q' does not affect decidability of bounded equivalence of q and q'.

COROLLARY 4.9. *The bounded equivalence problem is decidable for* positive *disjunctive α-queries with comparisons ranging over $\mathcal{I}$ if and only if the bounded equivalence problem is decidable for disjunctive α-queries* with negation *and comparisons ranging over $\mathcal{I}$.*

5. DECOMPOSITION PRINCIPLES

Levy and Sagiv [10] have shown that two disjunctive non-aggregate queries are equivalent if they are equivalent over all databases whose carrier is not greater than the size of the queries. For non-aggregate queries this is not surprising since an answer by a query q depends only on a single assignment satisfying q. Hence, if over some database $\mathcal{D}$, by means of the assignment γ, the query q returns the tuple $\bar{d}$, but q' does not return $\bar{d}$, then we can construct a database $\mathcal{D}_0 \subseteq \mathcal{D}$ that contains only constants occurring in q, q' and in γ such that $\bar{d} \in q^{\mathcal{D}_0}$ and $\bar{d} \notin q'^{\mathcal{D}_0}$.

For aggregate queries this argument cannot be applied since the results of a query are amalgamated of many single results that may involve arbitrarily many constants in the database. Nevertheless, for queries with an idempotent monoid or a group aggregation function we can reduce equivalence over arbitrary databases to equivalence over small databases.

As a first step, we formulate *decomposition principles* for these two classes of functions. Such a principle provides a method to compute the value of an aggregation over a union of sets of assignments from aggregations over the sets themselves and possibly some of their subsets.

PROPOSITION 5.1. *Let α be an idempotent monoid aggregation function and $(A_i)_{i=1}^k$ a family of sets of assignments, all defined for $\bar{y}$. Then*

$$\alpha(\bar{y}) \downarrow \bigcup_{i=1}^k A_i = \sum_{i=1}^k (\alpha(\bar{y}) \downarrow A_i).$$

Note that the "$\sum$" in the equation above is the extension of the idempotent monoid operation. In the case of *max*, for instance, the right hand side of the equation becomes $\max_{i=1}^k (max(y) \downarrow A_i)$.

Before we treat the case of groups, we remind the reader of the well-known Principle of Inclusion and Exclusion for computing the cardinality of a union of sets. It says that for any finite family of sets $(A_i)_{i=1}^k$ we have

$$\left|\bigcup_{i=1}^k A_i\right| = \sum_{i=1}^k |A_i| - \sum_{i<j} |A_i \cap A_j| + \cdots + (-1)^{k-1} |\bigcap_{i=1}^k A_i|. \quad (4)$$

For group aggregates, we can generalize Equation (4) to the following decomposition principle.

PROPOSITION 5.2. *Suppose that α is a group aggregation function and $(A_i)_{i=1}^k$ is a finite family of sets of assignments, all defined for $\bar{y}$. Then*

$$\alpha(\bar{y}) \downarrow \bigcup_{i=1}^k A_i = \sum_{i=1}^k (\alpha(\bar{y}) \downarrow A_i) - \sum_{i<j} (\alpha(\bar{y}) \downarrow A_i \cap A_j) + \cdots + (-1)^{k-1} (\alpha(\bar{y}) \downarrow \bigcap_{i=1}^k A_i). \quad (5)$$

Here the "$-$"-sign denotes inverses with respect to the group operation. Note that for $\alpha = count$, Equation (5) simplifies to Equation (4), since $(count \downarrow A) = |A|$ for every set of assignments A.

Because of the two above propositions, we say that idempotent monoid and group aggregation functions are *decomposable*.

6. REDUCING EQUIVALENCE TO LOCAL EQUIVALENCE

We now show that for queries with decomposable aggregation functions local equivalence implies equivalence. To this end we show first that, given two queries and a database, we can identify small subsets of the database such that the satisfying assignments over the database are the union of the satisfying assignments over the subsets. Then we apply the decomposition principles to conclude from the fact that the queries return the same results over the small databases that they return the same result over the original database.

Let q_1, q_2 be disjunctive queries, $\mathcal{D}$ be a database, and $\bar{d}$ be a tuple of constants. Let $(\mathcal{D}_i)_{i=1}^k$ be a family of databases with $\mathcal{D}_i \subseteq \mathcal{D}$ for all $i \in 1..k$. Then $(\mathcal{D}_i)_{i=1}^k$ is a *decomposition* of $\mathcal{D}$ with respect to q_1, q_2 and $\bar{d}$ if the following holds:

1. $|carr(\mathcal{D}_i)| \leq \tau(q_1, q_2)$ for all $i \in 1..k$;
2. $\Gamma_{\bar{d}}(q_j, \mathcal{D}) = \bigcup_{i=1}^k \Gamma_{\bar{d}}(q_j, \mathcal{D}_i)$ for $j = 1, 2$;
3. $\bigcap_h \Gamma_{\bar{d}}(q_j, \mathcal{D}_{i_h}) = \Gamma_{\bar{d}}(q_j, \bigcap_h \mathcal{D}_{i_h})$ for $j = 1, 2$ and for all subfamilies $(\mathcal{D}_{i_h})_h$ of $(\mathcal{D}_i)_i$.

The first condition means that, intuitively, the databases $\mathcal{D}_i$ are small. The second condition says that for each q_j, $j = 1, 2$, we obtain exactly the satisfying assignments over $\mathcal{D}$ that return $\bar{d}$ if we evaluate q_j over each $\mathcal{D}_i$ separately and select the assignments that return $\bar{d}$ over $\mathcal{D}_i$. The third condition says that for each q_j, in order to obtain the intersection of the assignment sets $\Gamma_{\bar{d}}(q_j, \mathcal{D}_{i_h})$, it suffices to evaluate q_j over the intersection of the databases $\mathcal{D}_{i_h}$.

THEOREM 6.1. *Let q, q' be a pair of disjunctive queries, let $\mathcal{D}$ be a database, and $\bar{d}$ be tuple of constants from $\mathcal{D}$. Then there exists a decomposition of $\mathcal{D}$ with respect to q, q' and $\bar{d}$.*

PROOF. The proof is in Appendix A. □

THEOREM 6.2. *Let α be a decomposable aggregation function, and let q, q' be disjunctive α-queries. Then q and q' are equivalent if and only if they are locally equivalent.*

PROOF. We only have to show that local equivalence implies equivalence. Suppose therefore that q and q' agree on all databases whose carrier has at most $\tau(q, q')$ elements. Let $\mathcal{D}$ be any database and $\bar{d}$ be a tuple of constants. It suffices to show that

$$\alpha(\bar{y}) \downarrow \Gamma_{\bar{d}}(q, \mathcal{D}) = \alpha(\bar{y}) \downarrow \Gamma_{\bar{d}}(q', \mathcal{D}).$$

Let $(\mathcal{D}_i)_{i=1}^k$ be a *decomposition* of $\mathcal{D}$ with respect to q, q' and to $\bar{d}$.

$$\alpha(\bar{y}) \downarrow \Gamma_{\bar{d}}(q, \mathcal{D}) = \alpha(\bar{y}) \downarrow \bigcup_{i=1}^{k} \Gamma_{\bar{d}}(q, \mathcal{D}_i) \quad \text{(a)}$$

$$= \sum_{i=1}^{k} (\alpha(\bar{y}) \downarrow \Gamma_{\bar{d}}(q, \mathcal{D}_i)) - \cdots + (-1)^{k-1} (\alpha(\bar{y}) \downarrow \bigcap_{i=1}^{k} \Gamma_{\bar{d}}(q, \mathcal{D}_i)) \quad \text{(b)}$$

$$= \sum_{i=1}^{k} (\alpha(\bar{y}) \downarrow \Gamma_{\bar{d}}(q, \mathcal{D}_i)) - \cdots + (-1)^{k-1} (\alpha(\bar{y}) \downarrow \Gamma_{\bar{d}}(q, \bigcap_{i=1}^{k} \mathcal{D}_i)) \quad \text{(c)}$$

$$= \sum_{i=1}^{k} (\alpha(\bar{y}) \downarrow \Gamma_{\bar{d}}(q', \mathcal{D}_i)) - \cdots + (-1)^{k-1} (\alpha(\bar{y}) \downarrow \Gamma_{\bar{d}}(q', \bigcap_{i=1}^{k} \mathcal{D}_i)) \quad \text{(d)}$$

$$= \sum_{i=1}^{k} (\alpha(\bar{y}) \downarrow \Gamma_{\bar{d}}(q', \mathcal{D}_i)) - \cdots + (-1)^{k-1} (\alpha(\bar{y}) \downarrow \bigcap_{i=1}^{k} \Gamma_{\bar{d}}(q', \mathcal{D}_i)) \quad \text{(e)}$$

$$= \alpha(\bar{y}) \downarrow \bigcup_{i=1}^{k} \Gamma_{\bar{d}}(q', \mathcal{D}_i) \quad \text{(f)}$$

$$= \alpha(\bar{y}) \downarrow \Gamma_{\bar{d}}(q', \mathcal{D}) \quad \text{(g)}$$

Figure 1: Equations for proof of Theorem 6.2.

If α is an idempotent monoid function, we apply Proposition 5.1, which yields

$$\alpha(\bar{y}) \downarrow \Gamma_{\bar{d}}(q, \mathcal{D}) = \alpha(\bar{y}) \downarrow \bigcup_{i=1}^{k} \Gamma_{\bar{d}}(q, \mathcal{D}_i) \quad \text{(6a)}$$

$$= \sum_{i=1}^{k} (\alpha(\bar{y}) \downarrow \Gamma_{\bar{d}}(q, \mathcal{D}_i)) \quad \text{(6b)}$$

$$= \sum_{i=1}^{k} (\alpha(\bar{y}) \downarrow \Gamma_{\bar{d}}(q', \mathcal{D}_i)) \quad \text{(6c)}$$

$$= \alpha(\bar{y}) \downarrow \bigcup_{i=1}^{k} \Gamma_{\bar{d}}(q', \mathcal{D}_i) \quad \text{(6d)}$$

$$= \alpha(\bar{y}) \downarrow \Gamma_{\bar{d}}(q', \mathcal{D}), \quad \text{(6e)}$$

where Equations (6a) and (6e) hold because of Property 2 of decompositions, Equations (6b) and (6d) hold because of Proposition 5.1, and Equation (6c) holds because q and q' are locally equivalent and the databases $\mathcal{D}_i$ contain at most $\tau(q, q')$ constants.

If α is a group aggregation function, we apply Proposition 5.2, which yields the equations in Figure 1, where Equations (a) and (g) hold because of Property 2 of decompositions, Equations (b) and (f) hold because of Proposition 5.2, Equations (c) and (e) hold because of Property 3 of decompositions, and Equation (d) holds because q and q' are locally equivalent and the databases $\mathcal{D}_i$ contain at most $\tau(q, q')$ constants. □

COROLLARY 6.3. *Suppose α is a decomposable aggregation function. If local equivalence is decidable for disjunctive α-queries, then also equivalence is decidable.*

COROLLARY 6.4. *Equivalence of disjunctive aggregate queries is decidable for the aggregation functions max, top2, count, parity, sum, and prod.*

7. EQUIVALENCE OF QUASILINEAR QUERIES

A positive conjunctive query is *linear* if no predicate occurs more than once. We generalize this by defining that a conjunctive query is *quasilinear* if no predicate that occurs in a positive literal, occurs more than once. Thus, in a quasilinear query no predicate occurs in both a positive and a negated literal and no predicate occurs more than once in a positive literal. In this section we show that for a wide range of quasilinear queries, equivalence is isomorphism.

We introduce an auxiliary definition. A conjunction of comparisons C is *reduced* if

- there are no variables x, y occurring in C such that $C \models x = y$;
- there is no variable x occurring in C such that $C \models x = d$ for a constant d.

We say that a conjunctive query is *reduced* if its comparisons are reduced.

We have shown in [12] that for any positive conjunctive query one can compute in polynomial time an equivalent reduced conjunctive query. This still holds when the query can contain negation. In [12] we have also shown that reduced linear *max*, *count* and *sum* queries are equivalent if and only if they are isomorphic. For queries with negaed literals we can generalize this result to quasilinear queries.

THEOREM 7.1. *Let α be an aggregation function. Suppose that reduced positive linear conjunctive α-queries are equivalent if and only if they are isomorphic. Then consistent reduced quasilinear conjunctive α-queries are equivalent if and if they are isomorphic.*

PROOF. Consider the reduced queries

$$q(\bar{x}, \alpha(\bar{y})) \leftarrow P \,\&\, N \,\&\, C$$
$$q'(\bar{x}, \alpha(\bar{y})) \leftarrow P' \,\&\, N' \,\&\, C'$$

where

- P and P' are conjunctions of positive relational atoms;

- N and N' are conjunctions of negated relational atoms;
- C and C' are conjunctions of comparisons.

Suppose that for positive conjunctive α-queries equivalence is isomorphism. Suppose that q is not isomorphic to q'. We show that q is not equivalent to q'.

We define the queries q_P and q'_P

$$q_P(\bar{x}, \alpha(\bar{y})) \leftarrow P \ \& \ C$$
$$q'_P(\bar{x}, \alpha(\bar{y})) \leftarrow P' \ \& \ C'$$

We consider two cases.

Case 1: Suppose that q_P is not isomorphic to q'_P. Then $q_P \not\equiv q'_P$. Let $\mathcal{D}$ be a database for which q_P and q'_P return different values. We may assume, without loss of generality, that $\mathcal{D}$ only contains atoms with predicates appearing in P or in P'. Hence, there is no atom in $\mathcal{D}$ containing a predicate appearing in N or N'. Thus, $q_P^{\mathcal{D}} = q^{\mathcal{D}}$ and ${q'}_P^{\mathcal{D}} = {q'}^{\mathcal{D}}$. We derive that $\mathcal{D}$ is a counter-example for equivalence of q and q'.

Case 2: Suppose that q_P is isomorphic to q'_P. Since q_P and q'_P are linear, there is only one isomorphism between them. Let θ be the isomorphism from q_P to q'_P. Note that θ is defined on all the variables in q, since q is a safe query. By our assumption, q is not isomorphic to q'. Thus, $\theta(N) \neq N'$. Suppose, w.l.o.g., that $\neg a$ appears in N and $\theta(\neg a)$ does not appear in N'. Let δ be a mapping of the variables in q to constants such that δ is consistent with the comparisons in q. We define a database $\mathcal{D} := \{\delta(b) \mid b \in P\} \cup \{\delta(a)\}$. Clearly, $\delta\bar{x} \notin q^{\mathcal{D}}$ whereas $\delta\bar{x} \in {q'}^{\mathcal{D}}$. Thus, $q \not\equiv q'$. □

Now, it follows from our results in [12] that for quasilinear queries with the aggregate functions *max*, *sum* and *count* equivalence boils down to isomorphism. In a similar fashion to the proofs there, we can extend our results to additional aggregate functions.

THEOREM 7.2. *Let q and q' be satisfiable conjunctive α-queries, possibly with constants, comparisons, and negated atoms. Suppose that q, q' are quasilinear and reduced and that α is one of* max, top2, count, parity, sum, prod, *or* avg. *Then q and q' are equivalent if and only if they are isomorphic.*

For *cntd* a similar result can be shown for common cases.

THEOREM 7.3. *Let q, q' be satisfiable conjunctive quasilinear cntd-queries. Moreover, suppose that*

- *the comparisons in q and q' use only $\leq, \geq$;*
- *q and q' either range over the rationals* or *do not have constants.*

Then q and q' are equivalent if and only if they are isomorphic.

COROLLARY 7.4. *The equivalence problem for the class of quasilinear α-queries is decidable in polynomial time if α is one of the aggregation functions* max, top2, count, parity, sum, prod, *or* avg *and for common cntd-queries.*

8. CONCLUSION

Necessary and complete conditions for the decidability of bounded equivalence of disjunctive aggregate queries with negation have been presented. This problem has been shown to be decidable for a wide class of aggregation functions. Equivalence of aggregate queries with negation has been reduced to a special case of bounded equivalence, called local equivalence, for decomposable aggregation functions. We have also shown that equivalence can be decided in polynomial time for the common case of quasilinear queries.

Bag-set semantics has been introduced in [3] to give a formal account of the way in which SQL queries are executed, which do not return a set of tuples but a multiset. It is easy to see that two non-aggregate queries are equivalent under bag-set semantics if and only if the aggregate queries obtained by adding the function *count* are equivalent. Thus, our results on *count*-queries directly carry over to non-aggregate queries that are evaluated under bag-set semantics. This is a significant contribution to the understanding of SQL queries. Moreover, these results can easily be extended to non-aggregate queries evaluated under *bag semantics* [3, 7], thereby, solving an additional open problem.

Novel proof techniques have been presented. One example is the application of the Principle of Inclusion and Exclusion to the case of group aggregation functions. Our results are couched in terms of abstract characterizations of aggregation functions. Thus, the results presented are easily extendible to additional aggregation functions.

Concepts seemingly similar to the ones introduced in the present paper have been investigated in [6]. In particular, the authors considered aggregation functions defined in terms of commutative monoids. However, the purpose of that research was to study the expressivity of logics that extend first-order logic by aggregation. In [6] it is shown that formulas in those extended logics are Hanf-local and Gaifman-local. Intuitively, this means that whether or not a formula is true for a tuple $\bar{d}$ in a structure, depends only on that part of the structure that is "close" to $\bar{d}$. A class of formulas that is Hanf- or Gaifman-local need not be decidable. In addition, the authors only considered monoids over the rationals, which excludes functions such as *topK* and *parity*.

We leave for future research the problem of deciding equivalence among *avg*, *stdev*, *median* and *cntd* queries as well as equivalence of aggregate queries with a `HAVING` clause. The adaption of our results to the view usability problem is another important open problem.

9. REFERENCES

[1] A. Aho, Y. Sagiv, and J. Ullman. Efficient optimization of a class of relational expressions. *ACM Transactions on Database Systems*, 4(4):435–454, 1979.

[2] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. 9th Annual ACM Symposium on Theory of Computing*, 1977.

[3] S. Chaudhuri and M. Vardi. Optimization of real conjunctive queries. In *Proc. 12th Symposium on Principles of Database Systems*, Washington (D.C., USA), May 1993. ACM Press.

[4] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In C. Papadimitriou, editor, *Proc. 18th Symposium on Principles of Database Systems*, Philadelphia (Pennsylvania, USA), May 1999. ACM Press.

[5] S. Grumbach, M. Rafanelli, and L. Tininini. Querying aggregate data. In C. Papadimitriou, editor, *Proc. 18th Symposium on Principles of Database Systems*, pages 174–183, Philadelphia (Pennsylvania, USA), May 1999. ACM Press.

[6] L. Hella, L. Libkin, J. Nurmonen, and L. Wong. Logics with aggregate operators. In *Proc. 14th IEEE Symposium on Logic in Computer Science*, pages 35–44, Trento (Italy), July 1999. IEEE Computer Society Press.

[7] Y. Ioannidis and R. Ramakrishnan. Beyond relations as sets. *ACM Transactions on Database Systems*, 20(3):288–324, 1995.

[8] D. Johnson and A. Klug. Optimizing conjunctive queries that contain untyped variables. *SIAM Journal on Computing*, 12(4):616–640, 1983.

[9] G. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases*. Springer-Verlag, 2000.

[10] A. Levy and Y. Sagiv. Queries independent of updates. In *Proc. 19th International Conference on Very Large Data Bases*, pages 171–181, Dublin (Ireland), Aug. 1993. Morgan Kaufmann Publishers.

[11] A. Levy and Y. Sagiv. Semantic query optimization in datalog programs. In *Proc. 14th Symposium on Principles of Database Systems*, pages 163–173, San Jose (California, USA), Proc. 14th Symposium on Principles of Database Systems 1995. ACM Press.

[12] W. Nutt, Y. Sagiv, and S. Shurin. Deciding equivalences among aggregate queries. In J. Paredaens, editor, *Proc. 17th Symposium on Principles of Database Systems*, pages 214–223, Seattle (Washington, USA), June 1998. ACM Press. Long version as Report of Esprit LTR DWQ.

[13] Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, 1981.

[14] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, 1988.

[15] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *Proc. 11th Symposium on Principles of Database Systems*, pages 331–345, San Diego (California, USA), May 1992. ACM Press.

APPENDIX

A. PROOF FOR SECTION 6

In this section we prove Theorem 6.2. We consider the queries q and q' defined as

$$q(\bar{x}, \alpha(\bar{y})) \leftarrow \bigvee_{i \in I} P_i \ \& \ N_i \ \& \ C_i$$

$$q'(\bar{x}, \alpha(\bar{y}')) \leftarrow \bigvee_{j \in J} P'_j \ \& \ N'_j \ \& \ C'_j$$

where P_i, P'_j are conjunctions of positive relational atoms, N_i, N'_j are conjunctions of negated relational atoms and C_i, C'_j are conjunctions of comparisons. We use L_i as a shorthand for $P_i \ \& \ N_i \ \& \ C_i$ and we use L'_j as a shorthand for $P'_j \ \& \ N'_j \ \& \ C'_j$.

Let $\mathcal{D}$ be a database and let $\bar{d}$ be a tuple. We must show that there exists a decomposition of $\mathcal{D}$ with respect to q, q' and $\bar{d}$.

We create a decomposition of $\mathcal{D}$ with respect to q, q' and $\bar{d}$ in a two step process. We first create databases out of the satisfying assignments of q and of q' into $\mathcal{D}$ that retrieve $\bar{d}$. Next, we extend these databases using the procedure *Extend_Database* to prevent them from satisfying negated atoms that were not satisfied in $\mathcal{D}$.

Recall that $\Gamma_{\bar{d}}(q, \mathcal{D})$ is the set of satisfying assignments from q into $\mathcal{D}$ that retrieve $\bar{d}$. We denote the disjunct of q satisfied by $\gamma \in \Gamma_{\bar{d}}(q, \mathcal{D})$ as L_γ. For each $\gamma \in \Gamma_{\bar{d}}(q, \mathcal{D})$ we define a database

$$\mathcal{D}_\gamma := \{\gamma(a) \mid a \in P_\gamma\}.$$

We use this notation since we consider a database to be a set of ground positive relational atoms. Note that $\mathcal{D}_\gamma$ satisfies the positive atoms in L_γ with respect to the assignment γ. However, we must extend the databases $\mathcal{D}_\gamma$ to ensure that $\mathcal{D}_\gamma$ does not satisfy negated atoms that were not satisfied in $\mathcal{D}$. We now create a database $\mathcal{D}^*_\gamma$ out of $\mathcal{D}_\gamma$ using the procedure *Extend_Database* presented in Figure 2.[2] Formally, we define $\mathcal{D}^*_\gamma := \mathit{Extend\_Database}(\mathcal{D}_\gamma, q, q', \mathcal{D})$.

In a similar fashion, we create databases $\mathcal{D}_{\gamma'}$ out of the satisfying assignments $\gamma' \in \Gamma_{\bar{d}}(q', \mathcal{D})$ of q' into $\mathcal{D}$ that retrieve $\bar{d}$. As above, these databases are extended to derive databases $\mathcal{D}^*_{\gamma'}$ using the procedure *Extend_Database*.

We now define

$$\Delta := \{\mathcal{D}^*_\gamma \mid \gamma \in \Gamma_{\bar{d}}(q, \mathcal{D})\} \cup \{\mathcal{D}^*_{\gamma'} \mid \gamma' \in \Gamma_{\bar{d}}(q', \mathcal{D})\}.$$

We present a series of lemmas that will enable us to prove that Δ is a decomposition of $\mathcal{D}$ w.r.t. q, q' and $\bar{d}$. We first note that clearly for all $\mathcal{D}^* \in \Delta$, it holds that $\mathcal{D}^* \subseteq \mathcal{D}$.

The first lemma states that the databases in Δ have the correct number of constants, i.e., that Property 1 of decompositions holds for Δ.

LEMMA A.1. *For all databases $\mathcal{D}^* \in \Delta$, it holds that $|carr(\mathcal{D}^*)| \leq \tau(q, q')$.*

[2]This process does not necessarily uniquely determine the database D^*_γ. However, this is not important for our proof.

Procedure: *Extend_Database*($\mathcal{D}_\gamma$,q,q',$\mathcal{D}$)
Input: Database $\mathcal{D}_\gamma$ to be extended
Queries q,q'
Original database $\mathcal{D}$
Output: Extension of $\mathcal{D}_\gamma$ with respect to q, q' and $\mathcal{D}$

1. $l := 0$
2. $\mathcal{D}_0 = \mathcal{D}_\gamma$
3. Repeat
4. $\quad l := l + 1$
5. $\quad \mathcal{D}_l := \mathcal{D}_{l-1}$
6. $\quad$ If there is an assignment δ of q into $\mathcal{D}_{l-1}$ that satisfies some L_δ in q such that
$\qquad\qquad (\exists a)[\neg a \in N_\delta \ \wedge \ \delta(a) \in \mathcal{D}]$
7. $\qquad$ then $\mathcal{D}_l := \mathcal{D}_l \cup \{\delta(a) \mid \neg a \in N_\delta \wedge \delta(a) \in \mathcal{D}\}$
8. $\quad$ If there is an assignment δ' of q' into $\mathcal{D}_{l-1}$ that satisfies some $L'_{\delta'}$ in q' such that
$\qquad\qquad (\exists a)[\neg a \in N'_{\delta'} \ \wedge \ \delta'(a) \in \mathcal{D}]$
9. $\qquad$ then $\mathcal{D}_l := \mathcal{D}_l \cup \{\delta'(a) \mid \neg a \in N'_{\delta'} \wedge \delta(a) \in \mathcal{D}\}$
10. Until $\mathcal{D}_l := \mathcal{D}_{l-1}$
11. Return $\mathcal{D}_l$

Figure 2: Procedure used to extend a database.

PROOF. Consider a database $\mathcal{D}^*_\gamma \in \Delta$. Clearly, $\mathcal{D}_\gamma$ contains at most $\tau(q)$ constants. Note that when an atom is added during the procedure, the constants appearing in the atom must have already appeared in the database, or must appear in q or q'. This follows since the queries are safe [14] and all variables in negated atoms must also appear in positive atoms. Thus, $\mathcal{D}^*_\gamma$ contains at most $\tau(q, q')$ constants.

Similarly, one can show that for $\mathcal{D}^*_{\gamma'} \in \Delta$, it holds that $|carr(\mathcal{D}^*_{\gamma'})| \leq \tau(q, q')$. Thus, it easily follows that for all $\mathcal{D}^* \in \Delta$, $|carr(\mathcal{D}^*)| \leq \tau(q, q')$. □

We show Property 2 of decompositions for Δ.

LEMMA A.2. *The following relationships hold between the assignments of q and q' into $\mathcal{D}$ and into databases in Δ:*

1. $\Gamma_{\bar{d}}(q, \mathcal{D}) = \bigcup_{\mathcal{D}^* \in \Delta} \Gamma_{\bar{d}}(q, \mathcal{D}^*)$;
2. $\Gamma_{\bar{d}}(q', \mathcal{D}) = \bigcup_{\mathcal{D}^* \in \Delta} \Gamma_{\bar{d}}(q', \mathcal{D}^*)$.

PROOF. We show Equation 1. Equation 2 is proved analogously. We show that $\Gamma_{\bar{d}}(q, \mathcal{D}) \subseteq \bigcup_{\mathcal{D}^* \in \Delta} \Gamma_{\bar{d}}(q, \mathcal{D}^*)$. To prove this it is enough to show that for all $\gamma \in \Gamma_{\bar{d}}(q, \mathcal{D})$, the assignment γ satisfies q in $\mathcal{D}^*_\gamma$, i.e.,

$$\gamma \in \Gamma_{\bar{d}}(q, \mathcal{D}^*_\gamma).$$

Suppose that a is a positive relational atom in the conjunct L_γ, then $\gamma(a) \in \mathcal{D}_\gamma$ by definition. Clearly, $\mathcal{D}_\gamma \subseteq \mathcal{D}^*_\gamma$, and thus, $\gamma(a) \in \mathcal{D}^*_\gamma$. If $\neg b$ is a negated relational atom in L_γ then $\gamma(b) \notin \mathcal{D}$. Otherwise, γ would not be a satisfying assignment of q in $\mathcal{D}$. According to the definition of $\mathcal{D}^*_\gamma$, it holds that $\mathcal{D}^*_\gamma \subseteq \mathcal{D}$, and therefore, $\gamma(b) \notin \mathcal{D}^*_\gamma$. Satisfaction of C_γ (the comparisons in L_γ) are dependent only on γ and not on a database. Thus, γ is a satisfying assignment of q into $\mathcal{D}^*_\gamma$.

We now show the other direction of containment, i.e., that $\Gamma_{\bar{d}}(q, \mathcal{D}) \supseteq \bigcup_{\mathcal{D}^* \in \Delta} \Gamma_{\bar{d}}(q, \mathcal{D}^*)$. It suffices to show that for all $\mathcal{D}^* \in \Delta$ it holds that $\Gamma_{\bar{d}}(q, \mathcal{D}) \supseteq \Gamma_{\bar{d}}(q, \mathcal{D}^*)$.

Suppose that $\gamma \in \Gamma_{\bar{d}}(q, \mathcal{D}^*)$. Suppose that γ satisfies conjunct L_γ of q in $\mathcal{D}^*$. Consider an literal l in L_γ. If l is a positive relational atom then $\gamma(l) \in \mathcal{D}^*$. We know that $\mathcal{D}^* \subseteq \mathcal{D}$, thus, $\gamma(l) \in \mathcal{D}$. Suppose that l is a negated relational atom of the form $\neg b$, and suppose, by way of contradiction, that $\gamma(b) \in \mathcal{D}$. Then γ satisfies the condition in line 6 of the procedure *Extend_Database* presented above. Thus, $\gamma(b)$ would have been added to $\mathcal{D}^*$ in contradiction to the fact that γ is a satisfying assignment of q into $\mathcal{D}^*$. Finally, note that satisfaction of comparisons is only dependent on the assignment. Thus, γ is a satisfying assignment of q into $\mathcal{D}$. □

In Lemma A.3 Property 3 of decompositions is proved for Δ.

LEMMA A.3. *The following relationships hold between intersections of sets of assignments and intersections of sets of databases:*

1. $\bigcap_h \Gamma_{\bar{d}}(q, \mathcal{D}^*_h) = \Gamma_{\bar{d}}(q, \bigcap_h \mathcal{D}^*_h)$;
2. $\bigcap_h \Gamma_{\bar{d}}(q', \mathcal{D}^*_h) = \Gamma_{\bar{d}}(q', \bigcap_h \mathcal{D}^*_h)$

for all subfamilies $\mathcal{D}^*_h$ of Δ.

PROOF. We show Equation 1. Equation 2 is proved analogously. We first show that $\Gamma_{\bar{d}}(q, \bigcap \mathcal{D}^*_h) \subseteq \bigcap \Gamma_{\bar{d}}(q, \mathcal{D}^*_h)$. Let

γ be an assignment in $\Gamma_{\bar{d}}(q, \bigcap \mathcal{D}_h^*)$. Suppose that γ satisfies L_γ of q in $\bigcap \mathcal{D}_h^*$. Satisfaction of C_γ is dependent only on γ. Let a be a positive atom in L_γ. The atom $\gamma(a)$ appears in $\bigcap \mathcal{D}_h^*$ and thus, $\gamma(a)$ appears in $\mathcal{D}_h^*$ for all h. Thus, γ satisfies the positive atoms of L_γ in each of $\mathcal{D}_h^*$. Now, let l be a negated atom in L_γ of the form $\neg b$. Clearly, $\gamma(b) \notin \bigcap \mathcal{D}_h^*$. Suppose, by way of contradiction, that $\gamma(b) \in \mathcal{D}_h^*$ for some h. Then $\gamma(b) \in \mathcal{D}$, since $\mathcal{D}_h^* \subseteq \mathcal{D}$. However, it follows that we would have added $\gamma(b)$ to $\mathcal{D}_h^*$ for all h, since γ satisfies the condition in line 6 of *Extend_Database*. Thus, $\gamma(b) \in \bigcap \mathcal{D}_h^*$ in contradiction to the assumption. Hence, $\Gamma_{\bar{d}}(q, \bigcap \mathcal{D}_h^*) \subseteq \bigcap \Gamma_{\bar{d}}(q, \mathcal{D}_h^*)$ as required.

We now show that $\Gamma_{\bar{d}}(q, \bigcap \mathcal{D}_h^*) \supseteq \bigcap \Gamma_{\bar{d}}(q, \mathcal{D}_h^*)$. Suppose that $\gamma \in \bigcap \Gamma_{\bar{d}}(q, \mathcal{D}_h^*)$. Then $\gamma \in \Gamma_{\bar{d}}(q, \mathcal{D}_h^*)$ for all h. Let L_γ be a conjunct such that γ satisfies L_γ in $\mathcal{D}_h^*$ for all h. Once again, satisfaction of C_γ is dependent only on γ. Consider a positive relational atom a in L_γ. Then $\gamma(a) \in \mathcal{D}_h^*$ for all h. Thus, $\gamma(a) \in \bigcap \mathcal{D}_h^*$. Similarly, consider a negated atom l of the form $\neg b$ in L_γ. Then $\gamma(b) \notin \mathcal{D}_h^*$ for all h, and thus, $\gamma(b) \notin \bigcap \mathcal{D}_h^*$. Clearly it follows that $\gamma \in \Gamma_{\bar{d}}(q, \bigcap \mathcal{D}_h^*)$. Hence, the equality required holds. □

We can now prove our theorem about the existance of decompositions.

THEOREM A.4. *Let q, q' be a pair of disjunctive queries, let $\mathcal{D}$ be a database, and $\bar{d}$ be tuple of constants from $\mathcal{D}$. Then there exists a decomposition of $\mathcal{D}$ with respect to q, q' and $\bar{d}$.*

PROOF. From Lemmas A.1, A.2 and A.3 it follows that Δ is a decomposition of $\mathcal{D}$ w.r.t. q, q' and $\bar{d}$ as required. □