Containment of Aggregate Queries*

Sara Cohen
Faculty of Industrial Engineering and Management
Technion—Israel Institute of Technology
Haifa 32000, Israel
sarac@ie.technion.ac.il

1 Introduction

It is now common for databases to contain many gigabytes, or even many terabytes, of data. Scientific experiments in areas such as high energy physics produce data sets of enormous size, while in the business sector the emergence of decision-support systems and data warehouses has led organizations to build up gigantic collections of data. Aggregate queries allow one to retrieve concise information from such a database, since they can cover many data items while returning a small result. OLAP queries, used extensively in data warehousing, are based almost entirely on aggregation [4, 16]. Aggregate queries have also been studied in a variety of settings beyond relational databases, such as mobile computing [1], global information systems [21], stream data analysis [12], sensor networks [22] and constraint databases [2].

The execution of aggregate queries tends to be time consuming. Computing one aggregate value often requires scanning many data items. Since aggregate queries are a popular means to query many types of database systems, it is essential to develop algorithms for two major problems. One is optimizing aggregate queries. The other is using materialized views in the evaluation of those queries. It is widely accepted that the ability to determine containment or equivalence between queries is a key to solving both problems. Thus, containment of nonaggregate queries over relational databases has been studied extensively, e.g., [3, 19, 27, 20].

Considerable work has been done on the problem of efficiently computing aggregate queries, e.g. [5, 15, 25]. However, without a coherent understanding of the underlying principles, it is not possible to present algorithms and techniques that are complete. Hence, most of the algorithms were based on sufficient conditions for equivalence, and complete algorithms were presented in these papers only for very restricted cases. A better understanding of these problems requires a complete characterization of equivalences among aggregate queries.

The ability to characterize equivalences among aggregate queries is also of primary importance when optimizing nonaggregate queries that are evaluated under bag-set semantics. These semantics are the default for evaluating SQL queries (e.g., SQL queries without the keyword DISTINCT). Determining equivalence of nonaggregate queries under bag-set semantics can be reduced to determining equivalence of queries with the aggregation function *count*. Hence, the study of aggregate-query equivalences and optimization are also of immediate benefit when attempting to optimize nonaggregate SQL queries.

There are quite a few papers that deal with the aggregate-query containment and equivalence problems. This survey contains in detail only a small sampling of previous results. The emphasis in this paper is on results that have a short proof sketch. In addition, we demonstrate with these results the different strategies that have been employed for solving the equivalence and containment problems. Some important results have been mentioned only briefly due to space limitations. For these results, the reader is referred to the appropriate papers.

This survey is organized as follows. In Section 2 we discuss how determining equivalence of aggregate queries differs from determining equivalence of nonaggregate queries. In Section 3 we present the formal syntax and semantics of aggregate queries. Section 4 contains some necessary definitions. We present several interesting results on aggregate-query equivalence and containment in Sections 5 and 6. Finally,

*Database Principles Column. Column editor: Leonid Libkin, Department of Computer Science, University of Toronto, Toronto, Ontario M5S 3H5, Canada. E-mail: libkin@cs.toronto.edu.

we conclude in Section 7 with a discussion of complexity results and related work.

2 Motivation

We discuss how determining equivalence among aggregate queries differs from determining equivalence among nonaggregate queries. Thus, this section motivates the study of aggregate-query equivalence by showing that previous results for nonaggregate queries do not carry over easily to this case. The discussion will be somewhat informal and the main ideas will be conveyed through a series of examples.

The examples in this section will be based on queries of the form

$$q(\bar{s}, \alpha(\bar{t})) \leftarrow A\,,$$

where A is a conjunction of non-negated relational atoms and comparisons, and $\alpha(\bar{t})$ is an *aggregate term*. Such queries are *positive* (i.e., contain no negated atoms). We sometimes write the body of q as $R \wedge C$ when we want to indicate that R is a conjunction of relational atoms and C is a conjunction of comparisons. A formal definition of a query (which may also contain negation) and its semantics will be presented in Section 3.

We give an informal account of the semantics of a positive aggregate query here, by showing how such a query is translated into SQL. The process of translating such queries into SQL is almost identical to that of translating a conjunctive nonaggregate Datalog query into SQL. In particular, *(1)* the relational atoms in A define the relations appearing in the FROM clause, *(2)* both comparisons and repeated occurrences of variables in A define the conditions appearing in the WHERE clause *and (3)* the head of the query $\bar{s}, \alpha(\bar{t})$ defines the SELECT clause. In addition, the variables in $\bar{s}$ also appear in the GROUP BY clause of the query. Thus, the variables in $\bar{s}$ are both output variables and grouping variables.

To demonstrate this process, consider the relations $P(A, B)$ and $R(C, D)$ and the query

$$q_1(x, sum(y)) \leftarrow p(x, y) \wedge r(z, y) \wedge x < z\,.$$

In SQL, q_1 is written in the following manner:

```
SELECT P.A, SUM(P.B)
FROM P,R
WHERE P.B=R.D and P.A<R.C
GROUP BY P.A;
```

We consider the problem of characterizing equivalence of aggregate queries, by comparing this problem to the corresponding one for nonaggregate queries.

Homomorphisms Are Not Sufficient. For positive nonaggregate queries, equivalence has been characterized in terms of homomorphisms [3, 19]. A *homomorphism* from $q(\bar{s}) \leftarrow R \wedge C$ to $q'(\bar{s}') \leftarrow R' \wedge C'$ is a substitution θ of the variables of q with the terms of q' such that *(1)* $\theta(\bar{s}) = \bar{s}'$, *(2)* $\theta(R) \subseteq R'$ *and (3)* $C' \models \theta(C)$.

If the nonaggregate queries q and q' do not contain comparisons, then q' is contained in q if and only if there is a homomorphism from q to q'. In addition, q is equivalent to q' if and only if such homomorphisms exist in both directions. (Determining containment and equivalence requires checking for the existence of several homomorphisms if the queries may contain comparisons.)

Intuitively, a characterization in terms of homomorphisms is possible since, for nonaggregate queries, a tuple is in the result if there is at least one satisfying assignment of the body that derives it. The number of satisfying assignments does not affect the result. Consider, for example, the following queries:

$$q_2(x) \leftarrow p(x, w)$$
$$q_3(x) \leftarrow p(x, w) \wedge p(x, z)\,.$$

It is not difficult to show that there is a homomorphism from q_2 to q_3 and a homomorphism from q_3 to q_2. Indeed, it is clear that these queries are equivalent, since both return x values such that there is at least one y for which $p(x, y)$.

On the other hand, consider the following *count*-queries, derived by adding the *count* function to each of q_2 and q_3:

$$q_4(x, count) \leftarrow p(x, w)$$
$$q_5(x, count) \leftarrow p(x, w) \wedge p(x, z)\,.$$

Now, each query returns both the satisfying x values, along with the *number of satisfying assignments* for each value of x. The queries q_4 and q_5 are not equivalent. This can be demonstrated by the database $\{p(10, 20), p(10, 30)\}$, for which q_4 will retrieve $(10, 2)$ and q_5 will retrieve $(10, 4)$.

From this simple example, it is apparent that any characterization will have to take into account the number of assignments and not only the existence of an assignment. Thus, the existence of a homomorphism will not usually be a sufficient condition for equivalence of aggregate queries.

Isomorphisms Are Not Necessary. Since we must account for the number of satisfying assignments, it is natural to try to characterize equivalence of aggregate queries in terms of isomorphisms, instead of homomorphisms. Formally, queries q and q' are *isomorphic* of there is a homomorphism θ from q to q' that is bijective and its inverse is also a homomorphism. Characterizing equivalence in terms of isomorphisms is appealing since the existence of an isomorphism is a obviously a sufficient condition for equivalence among aggregate queries. For positive *count*-queries that have no comparisons this is in fact a complete characterization [6, 23]. In other words, two positive *count*-queries that have no comparisons are equivalent if and only if they are isomorphic.

It turns out that the existence of an isomorphism is not always a necessary condition for aggregate-query equivalence. Consider, for example, the following *count*-queries:

$$q_6(count) \leftarrow p(x) \wedge p(y) \wedge p(z) \wedge x < y \wedge x < z$$
$$q_7(count) \leftarrow p(x) \wedge p(y) \wedge p(z) \wedge x < z \wedge y < z\,.$$

These queries are not isomorphic, yet it is not difficult to show that they are equivalent.

One can also find aggregate queries without comparisons for which isomorphism is not a necessary condition for equivalence. Too see this consider the following queries, which are a variation on q_4, q_5:

$$q_8(x, avg(w)) \leftarrow p(x, w)$$
$$q_9(x, avg(w)) \leftarrow p(x, w) \wedge p(x, z)\,.$$

Queries q_8 and q_9 are equivalent even though they are not isomorphic (and have a different number of satisfying assignments for each value of x).[1]

To conclude this part of the discussion, isomorphism is always a sufficient condition for equivalence, but is not always a necessary condition for equivalence.

Size of a Counter-Example. We now consider a tangential problem that arises when trying to show that some given characterization for equivalence or for containment is correct (i.e., complete). In order to show that a characterization for containment (resp. equivalence) is correct, it is often useful to demonstrate that if the characterization does not hold for queries q and q', then a counter-example can be built that shows that q is not contained in (resp. equivalent to) q'. For positive nonaggregate queries one can always create a "small" counter-example, i.e., a counter-example the size of the given queries. (In fact, such counter-examples are often built by taking the body of one of the queries as a database.) This is not surprising, since each value in the output is created by a single assignment.

One may ask whether "small" counter-examples always exist for aggregate queries. Recall that aggregate values are computed by aggregating together many different values. Hence, it would seem possible for a query q to always be contained in a query q' when "small" databases are considered, but for this relationship to no longer hold over larger ones. This is in fact the case. Consider the following queries:

$$q_{10}(x, count) \leftarrow p(x, w)$$
$$q_{11}(x, count) \leftarrow p(x, w) \wedge r(x, z)\,.$$

Over any database that is the size of q_{10} or q_{11} (i.e., that contains at most two atoms), q_{11} is contained in q_{10}. However, over databases with three atoms this no longer holds, e.g., the database $\{p(1,1), r(1,1), r(1,2)\}$. Thus, one of the difficulties when proving correctness of a given characterization is that larger databases must often be considered. (Rather surprisingly, the are many cases for which it is sufficient to consider databases that are at most the size of the queries.)

Differences Between Aggregation Functions. We consider a final problem of note. There are infinitely many different aggregation functions that can appear in an aggregate query. Even if the discussion is narrowed down to common aggregation functions, there are still many, e.g., *count*, *sum*, *max*, *avg*, *cntd* (count distinct), *prod*, to name only a few. Each aggregation function has its own quirks. For example,

- *count* counts values and is sensitive to the number of values;
- *max* ignores repeated values;
- *sum* ignores the value 0;
- *prod* ignores the value 1, and returns the value 0, when there computed over a bag containing the value 0.

The different oddities of aggregation functions make finding a "one-size-fits-all" solution for the

[1] Note that it is incorrect to conclude from this that equivalence of *avg*-queries can be characterized in the same way as equivalence of nonaggregate queries. It is easy to find a counter-example for such a characterization.

equivalence and containment problems very difficult. In particular, it is not difficult to find equivalent queries q and q' such that switching the aggregation function in the heads of q and q' to a different function yields queries that are no longer equivalent. For example the following queries are equivalent:

$$q_{12}(sum(y)) \leftarrow p(y) \wedge y > 0 \wedge p(z) \wedge z > 0$$
$$q_{13}(sum(y)) \leftarrow p(y) \wedge y \geq 0 \wedge p(z) \wedge z > 0\,.$$

However, an attempt to replace *sum* with *prod* yields queries that are not equivalent. (Interestingly, replacing *sum* with *max* does yield equivalent queries for this special case.)

Since every aggregation function has its own oddity, characterizations for equivalence of aggregate queries often are *custom-made*, i.e., defined separately for each aggregation function. In Section 5 we discuss customized characterizations for the aggregation functions *count* and *max*, and refer the reader to additional work on the topic.

It is sometimes possible to define classes of aggregation functions and then present general characterizations for equivalence of queries with any aggregation function within a class of functions. Such characterizations are often more complex than customized characterizations. We call this approach the *one-size-fits-all* approach (or the one-size approach, for short) and it is considered in Section 6.

3 Syntax and Semantics

We present the formal syntax and semantics for aggregate queries using an extended Datalog notation.

Predicate symbols are denoted as p, q or r. A *term,* denoted as s or 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 $=$. 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 A, is a conjunction of literals. A condition A is *safe* [26] if every variable appearing in A either appears in a positive relational atom or is equated with such a variable. Throughout this paper we will assume that all conditions are safe.

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 $\alpha(\bar{t})$ as an abstract notation for an aggregate term. Note that $\bar{t}$ can be the empty tuple as in the case of the functions *count* or *parity*.

To simplify the exposition, we will only consider aggregate queries which have a single aggregation term. In many cases, it is possible to reduce the query equivalence problem for queries with several aggregate terms to one of equivalence with a single aggregate term, e.g., [23]. We will also only consider queries with conjunctive bodies (i.e., without disjunctions). Many of the results surveyed here have been extended to queries with disjunctions.

An *aggregate query* is a non-recursive expression of the form

$$q(\bar{s}, \alpha(\bar{t})) \leftarrow A, \tag{1}$$

where A contains all the variables in $\bar{s}$ and in $\bar{t}$. We call $\bar{t}$ the *grouping terms* of the query, and we call $\bar{s}$ the *aggregation terms* of the query. If the aggregate term in the head of a query has the form $\alpha(\bar{t})$, we call the query an *α-query* (e.g., a *max*-query).

We distinguish several special types of aggregate queries. A query is *relational* if it contains no comparisons. A query is *positive* if it does not contain any negated relational atoms. A query is *linear* if it is positive and contains no relational predicate more than once (i.e., has no self-joins). Finally, a query is *quasilinear* if no predicate that occurs in a positive literal, occurs more than once.

It is convenient to consider queries in a particular normal form. Let q be a query with comparisons C. We say that q is *reduced* if *(1)* there are no two distinct variables x, y in C such that $C \models x = y$ *and (2)* there is no variable x in C such that $C \models x = d$, for some constant d. For every query, it is possible to compute in polynomial time an equivalent reduced query.

Example 3.1 Consider the relations `teach(prof, course)` and `study(course, student, grade)`, and the queries:

$$q_{14}(c, max(g)) \leftarrow \texttt{study}(c, s, g) \wedge \texttt{teach}(\texttt{Lau}, c)$$
$$q_{15}(c, avg(g)) \leftarrow \texttt{study}(c, s, g) \wedge \texttt{teach}(\texttt{Lau}, c) \wedge \neg\texttt{teach}(\texttt{Levy}, c) \wedge g > 55$$

The query q_{14} computes the maximum grade in each course taught by Prof. Lau. The query q_{15} computes the average passing grade (i.e., over 55) of students in each of Prof. Lau's courses that are not also taught by Prof. Levy.

The query q_{14} is positive and linear. Note that q_{15} is not quasilinear since the predicate `teach` occurs in a positive literal and occurs more than once in q_{15}. The queries q_{14} and q_{15} are both reduced. □

Databases are sets of ground relational atoms, denoted $\mathcal{D}$. Consider a query q as in Equation 1. We define how, for a database $\mathcal{D}$, the query yields a new relation $q^{\mathcal{D}}$. We proceed in two steps.

Let $\Gamma(q, \mathcal{D})$ denote the set of assignments γ over $\mathcal{D}$ that satisfy A. Recall that $\bar{s}$ are the grouping terms of q and $\bar{t}$ are the aggregation terms. For a tuple of constants $\bar{d}$, let $\Gamma_{\bar{d}}(q, \mathcal{D})$ be the subset of $\Gamma(q, \mathcal{D})$ consisting of assignments γ with $\gamma(\bar{s}) = \bar{d}$. In the sets $\Gamma_{\bar{d}}(q, \mathcal{D})$, we group those satisfying assignments that agree on $\bar{s}$. We use $\Gamma^{\bar{t}}_{\bar{d}}(q, \mathcal{D})$ to denote the *bag* of values that $\Gamma_{\bar{d}}(q, \mathcal{D})$ associates with the tuple $\bar{t}$, i.e., $\Gamma^{\bar{t}}_{\bar{d}}(q, \mathcal{D}) := \{\!\{\gamma(\bar{t}) | \gamma \in \Gamma_{\bar{d}}(q, \mathcal{D})\}\!\}$.

Now we define the result of evaluating $q(\bar{s}, \alpha(\bar{t}))$ over $\mathcal{D}$, denoted $q^{\mathcal{D}}$, by

$$\left\{ \left(\bar{d}, \alpha(\Gamma^{\bar{t}}_{\bar{d}}(q, \mathcal{D}))\right) \;\middle|\; \bar{d} = \gamma(\bar{s}) \text{ for some } \gamma \in \Gamma(q, \mathcal{D}) \right\}.$$

We say that queries q and q' are *equivalent*, written $q \equiv q'$, if, over every database, they return identical sets of results, that is, if $q^{\mathcal{D}} = {q'}^{\mathcal{D}}$ for all databases $\mathcal{D}$. Similarly, q is contained in q', denoted $q \subseteq q'$, if $q^{\mathcal{D}} \subseteq {q'}^{\mathcal{D}}$ for all databases $\mathcal{D}$.

4 Linear Expansions

In this section we present some definitions needed for characterizing equivalence of queries. Generally, the comparisons in the body of a query induce a partial order among the variables of the query. In order to deal with containment and equivalence of arbitrary queries, which may have comparisons, this partial order should be extended to a linear order.

Let $q(\bar{s}, \alpha(\bar{t})) \leftarrow R \wedge C$ be a query. Let W be the set of variables appearing in q and let D be a set of constants that contains the constants of q. A query $q'(\bar{s}, \alpha(\bar{t})) \leftarrow R \wedge C'$ is a *linearization* of q with respect to D if *(1)* $C' \models C$ *and (2)* for every two terms $s, t \in W \cup D$, exactly one of $s < t$, $s = t$, or $s > t$ is implied by C'.

A *linear expansion* of q with respect to the constants D is a set of linearizations Q of q with respect to D such that *(1)* no two queries in Q have equivalent comparisons (i.e., comparisons that imply the same linear ordering over the variables of q) *and (2)* for every linearization q' of q there is a query $q'' \in Q$ such that the comparisons of q' and q'' are equivalent. A *reduced linear expansion* of q with respect to D is derived by first computing a linear expansion Q of q and then replacing each query q' in Q with a reduced version of q'.

Example 4.1 Consider query q_6, repeated here:

$$q_6(count) \leftarrow a(x) \wedge a(y) \wedge a(z) \wedge x < y \wedge x < z\,.$$

The set of queries $\{q_6^a, q_6^b, q_6^c\}$, defined below, is a linear expansion of q_6:

$$q_6^a(count) \leftarrow a(x) \wedge a(y) \wedge a(z) \wedge x < y \wedge y < z\,,$$
$$q_6^b(count) \leftarrow a(x) \wedge a(y) \wedge a(z) \wedge x < z \wedge z < y\,,$$
$$q_6^c(count) \leftarrow a(x) \wedge a(y) \wedge a(z) \wedge x < y \wedge y = z\,.$$

Note that q_6^a and q_6^b are isomorphic, even though their sets of comparisons are not equivalent. Note also that $\{q_6^a, q_6^b, q_6^c\}$ is not a reduced linear expansion of q_6, since q_6^c is not reduced. □

Let Q and Q' be sets of queries. We say that Q and Q' are *isomorphic* if there is a bijection $\mu\colon Q \to Q'$ that maps queries in Q to isomorphic queries in Q'. For a given query q and constants D, there is no unique reduced linear expansion. However, it is easy to see that any two such reduced linear expansions are isomorphic. Hence, we use $\mathcal{E}_D(q)$ to denote an arbitrary reduced linear expansion of q w.r.t. D.

5 Customized Characterization

A customized equivalence characterization is one that is defined for a specific aggregation function. Several papers have presented customized characterizations for various aggregation functions. In [23], characterizations for equivalence of positive *count*-queries, *sum*-queries and *max*-queries were presented. Characterizations of equivalence for restricted *cntd*-queries were also presented. The results of [23] were extended in [10] to queries with disjunctive bodies. Equivalence of positive *avg*-queries and of positive *percent*-queries were characterized in [13].

This section contains a sampling of customized characterizations for equivalence. Subsections 5.1 and 5.2 consider equivalence of positive *count*-queries and *max*-queries, respectively. These results appeared in [23].

5.1 Equivalence of *count*-Queries

In this section we present a complete characterization of equivalence of positive *count*-queries. This characterization can easily be extended to deal with queries that have disjunctions [10]. The characterization is of perhaps of particular interest since its proof of correctness is of a similar style to correctness proofs for characterizations of equivalence for nonaggregate queries (since the proof involves creating a counter-example out of the body of a query).

Equivalence of *count*-queries can be characterized in terms of isomorphism of their linear expansions.

Theorem 5.1 *Let $q_1(\bar{s}, count)$ and $q_2(\bar{t}, count)$ be positive count-queries. Let D be the set of constants appearing in q_1 or in q_1. Then $q_1 \equiv q_2$ if and only if $\mathcal{E}_D(q_1)$ and $\mathcal{E}_D(q_2)$ are isomorphic.*

Proof. It is not difficult to see that isomorphism of $\mathcal{E}_D(q_1)$ and $\mathcal{E}_D(q_2)$ is a sufficient condition for equivalence of q_1 and q_2. We give a sketch of the proof that this is a necessary condition for equivalence. Suppose that $\mathcal{E}_D(q_1)$ and $\mathcal{E}_D(q_2)$ are not isomorphic. We will show that we can create a database out of the body of one of the queries in $\mathcal{E}_D(q_1) \cup \mathcal{E}_D(q_2)$ that is a counter-example for equivalence of q_1 and q_2.

Let q be a query. We use $|\mathcal{E}_D(q_1)|_q$ and $|\mathcal{E}_D(q_2)|_q$ to denote the number of queries in $\mathcal{E}_D(q_1)$ and $\mathcal{E}_D(q_2)$, respectively, that are isomorphic to q. We use $|q|_r$ and $|q|_v$ to denote the number of relational atoms and variables, respectively, in q.

Since $\mathcal{E}_D(q_1)$ is not isomorphic to $\mathcal{E}_D(q_2)$ there is at least one query $q \in \mathcal{E}_D(q_1) \cup \mathcal{E}_D(q_2)$ such that $|\mathcal{E}_D(q_1)|_q \neq |\mathcal{E}_D(q_2)|_q$. Let q_* be a query in $\mathcal{E}_D(q_1) \cup \mathcal{E}_D(q_2)$ such that *(1)* $|\mathcal{E}_D(q_1)|_{q_*} \neq |\mathcal{E}_D(q_2)|_{q_*}$, *(2)* $|q_*|_r$ is minimal among all queries with Property 1 *and (3)* $|q_*|_v$ is minimal among all queries with Properties 1 and 2. It is possible to show that by creating a database out of the body of q_*, we derive a counter-example to the equivalence of q_1 and q_2. □

5.2 Equivalence of *max*-Queries

Some aggregation functions, such as *max*, are not sensitive to multiplicities. For such functions it may be possible to reduce equivalence of aggregate queries to equivalence of nonaggregate queries. This holds for positive relational *max*-queries.

Let $q(\bar{s}, max(t)) \leftarrow A$ be a *max*-query. The *core* of q, denoted $\breve{q}$, is the query derived by stripping off the function *max* from the head of q, i.e., the query $\breve{q}(\bar{s}, t) \leftarrow A$.

Proposition 5.2 *Let q and q' be positive relational max-queries. Then, q is equivalent to q' if and only if $\breve{q}$ is equivalent to $\breve{q}'$.*

This characterization no longer holds if the queries may contain comparisons.

Example 5.3 Consider the queries

$$q_{16}(max(y)) \leftarrow p(y) \wedge p(z_1) \wedge p(z_2) \wedge z_1 < z_2$$
$$q_{17}(max(y)) \leftarrow p(y) \wedge p(z) \wedge z < y.$$

Both queries return answers if there are at least two elements in p. If this is the case, then q_{16} returns the greatest element among all elements of p, while q_{17} returns the greatest elements among all elements of p, other than the least. Thus, the two queries are equivalent. However, $\breve{q}_{16}$ is not equivalent to $\breve{q}_{17}$ since $\breve{q}_{16}$ contains $\breve{q}_{17}$, but $\breve{q}_{17}$ does not contain $\breve{q}_{16}$. □

Let $q(\bar{s}, max(t)) \leftarrow R \wedge C$ and $q'(\bar{s}', max(t')) \leftarrow R' \wedge C'$ be two queries. We say that q *is dominated* by q' if, for every database, whenever q returns a tuple $(\bar{d}, d)$, then q' returns a tuple $(\bar{d}, d')$ with $d' \geq d$. The following proposition states that dominance can be used to determine equivalence.

Proposition 5.4 *Queries q and q' are equivalent if and only if q dominates q' and q' dominates q.*

Dominance mappings, a variation on homomorphisms, are used to determine whether one query dominates another. A *dominance mapping* from $q(\bar{s}, max(t))$ to $q'(\bar{s}, max(t'))$ is a substitution θ of the variables of q with terms of q', such that *(1)* $\theta\bar{s} = \bar{s}'$, *(2)* $\theta(R) \subseteq R'$, *(3)* $C' \models \theta(C)$ *and (4)* $C' \models t' \leq \theta t$. Note that a dominance mapping differs from a homomorphism only in Property 4.

Theorem 5.5 *Let q_1 and q_2 be positive max-queries. Let D be the set of constants appearing in q_1 or in q_2. Then q_1 is dominated by q_2 if and only if for every linearization $q \in \mathcal{E}_D(q_1)$, there exists a dominance mapping from q_2 to q.*

Proposition 5.4 and Theorem 5.5 immediately yield a characterization for equivalence of *max*-queries.

6 One-Size Characterizations

Characterizations that determine containment or equivalence of aggregate queries for a class of aggregation functions (as opposed to considering a particular aggregation function) are called one-size characterizations. Such characterizations are very useful

since, given an aggregation function α not previously considered, it is generally easy to determine whether they are applicable to α-queries. However, such characterizations tend to be rather complex since they are based on abstract properties of aggregation functions.

The one-size approach was taken in [8] which considered equivalence of aggregate queries with *decomposable* aggregation functions. A small portion of this work is surveyed in Subsection 6.1. Containment of aggregate queries was studied in [9] for queries with *expandable* aggregation functions. Some of these results appear in Subsection 6.2. Note that throughout this section, we consider queries that may have negated relational atoms.

6.1 Equivalence

Recall that a query q is *quasilinear* if no predicate that occurs in a positive literal of q, 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. For every aggregation function α, we denote by $\mathcal{L}(\alpha)$ and $\mathcal{QL}(\alpha)$ the class of linear α-queries and quasilinear α-queries, respectively. We show that for a wide range of quasilinear queries, equivalence is isomorphism. This result appeared in [8].

We say that a class of queries $\mathcal{Q}$ is *proper* if for any two satisfiable reduced queries q, $q' \in \mathcal{Q}$ it is the case that q and q' are only equivalent if they are isomorphic. Theorem 6.1 relates $\mathcal{L}(\alpha)$ and $\mathcal{QL}(\alpha)$.

Theorem 6.1 *Let α be an aggregation function. Then $\mathcal{L}(\alpha)$ is proper if and only if $\mathcal{QL}(\alpha)$ is proper.*

A *singleton* bag is a bag that contains only one value. We say that an aggregation function α is a *singleton-determining* aggregation function, if for all singleton bags B and B' we have that $\alpha(B) = \alpha(B')$ if and only if $B = B'$. Clearly *max*, *sum*, *prod* and *avg* are singleton-determining aggregation functions. Note that *count* and *parity* are nullary aggregate functions. Thus, they are defined over a domain that contains only a single value, the empty tuple. Hence, *count* and *parity* are also singleton-determining aggregation functions.

Theorem 6.2 *Let α be an aggregation function. Then, α is singleton-determining if and only if $\mathcal{QL}(\alpha)$ is proper.*

Proof. By Theorem 6.1 it is sufficient to show that α is singleton-determining if and only if $\mathcal{L}(\alpha)$ is proper.

"$\Rightarrow$" Suppose that α is a singleton-determining aggregation function. We show that $\mathcal{L}(\alpha)$ is proper. To this end, let $q(\bar{s}, \alpha(\bar{t})) \leftarrow A$ and $q'(\bar{s}', \alpha(\bar{t}')) \leftarrow A'$ be satisfiable reduced linear α-queries. Suppose that $q \equiv q'$. We will show that q and q' are isomorphic.

In [3] it has been shown that linear nonaggregate queries without comparisons are set-equivalent if and only if they are isomorphic. This still holds even if the queries have comparisons. We associate with q a nonaggregate query $\hat{q}$, called the *nonaggregate projection* of q, which is derived from q by simply removing the aggregate term from the head of q. Thus, $\hat{q}$ has the form $\hat{q}(\bar{s}) \leftarrow A$.

Since $q \equiv q'$, they return values for the same grouping tuples. Thus, $\hat{q}$ is set-equivalent to $\hat{q}'$. Hence, $\hat{q}$ is isomorphic to $\hat{q}'$. Let θ be the isomorphism from $\hat{q}'$ to $\hat{q}$. If α is a nullary aggregation function, then θ is an isomorphism from q' to q. Suppose that α is not a nullary aggregation function.

Let γ be an instantiation of the terms in q that satisfies the comparisons in q and maps each term to a different value. We construct a database $\mathcal{D}$ out of q by applying γ to the relational part of q.

Clearly, the only satisfying assignment of q to the constants in $\mathcal{D}$ is exactly γ. Thus, q retrieves $(\gamma(\bar{s}), \alpha(\gamma(\bar{t})))$. The only satisfying assignment of q' is $\gamma \circ \theta$. Therefore, q' returns $(\gamma \circ \theta(\bar{s}'), \alpha(\gamma \circ \theta(\bar{t}')))$. Note that since θ is an isomorphism from q' to q, it holds that $\gamma \circ \theta(\bar{s}') = \gamma(\bar{s})$.

Recall that α is a singleton-determining aggregation function. Therefore, we have $\alpha(\gamma \circ \theta(\bar{t}')) = \alpha(\gamma(\bar{t}))$ if and only if $\gamma \circ \theta(\bar{t}') = \gamma(\bar{t})$. The instantiation γ is an injection, thus $\gamma \circ \theta(\bar{t}') = \gamma(\bar{t})$ if and only if $\theta(\bar{t}') = \bar{t}$. This must hold since $q \equiv q'$. Therefore, θ is an isomorphism from q' to q.

"$\Leftarrow$" Suppose that α is not a singleton-determining aggregation function. We show that $\mathcal{L}(\alpha)$ is not proper. To this end, we create linear α-queries q and q' such that $q \equiv q'$, but q and q' are not isomorphic.

Since α is not a singleton-determining aggregation function, there are singleton bags $B = \{\!\{d\}\!\}$, and $B' = \{\!\{d'\}\!\}$ such that $d \neq d'$ and $\alpha(B) = \alpha(B')$. The following queries are equivalent, but are not isomorphic:

$$q(\alpha(d)) \leftarrow p(d) \wedge p(d')$$
$$q'(\alpha(d')) \leftarrow p(d) \wedge p(d')\,.$$

□

Corollary 6.3 (Equivalence and Isomorphism) *The classes of quasilinear max, top2, count, sum, prod, parity and avg queries are proper.*

Proof. This result follows from the fact that all the aggregation functions above are singleton-determining and from Theorem 6.2. □

6.2 Containment

Characterizations for containment of nonaggregate queries have been presented [3]. Equivalence of nonaggregate queries is determined by checking for containment in both directions. Interestingly, when dealing with aggregate queries it seems that the containment problem is more elusive. In fact, most known containment results are derived by reducing containment to equivalence. Hence, in this section, such a reduction is presented. This result appeared in [9].

We present the class of *expandable aggregation functions*. Intuitively, for such functions changing the number of occurrences of values in bags B and B' does not affect the correctness of the formula $\alpha(B) = \alpha(B')$, as long as the proportion of each value in each bag remains the same.

Let B be a bag of constants and N be a positive integer. We use $B \otimes N$ to denote the bag derived from B by increasing the multiplicity of each term in B by a factor of N. Aggregation functions can be characterized by their behavior on expanded bags.

An aggregation function α is *expandable* if for all bags B and B' and for all positive integers N, $\alpha(B \otimes N) = \alpha(B' \otimes N)$ if and only if $\alpha(B) = \alpha(B')$. Many common aggregation functions, such as *max*, *cntd*, *count*, *sum* and *avg*, are expandable.

Given $q(\bar{s}, \alpha(\bar{t})) \leftarrow A$ and $q'(\bar{s}', \alpha(\bar{t}')) \leftarrow A'$, we say that a query p is a *join* of q and q' if p is defined as $p(\bar{s}, \alpha(\bar{t})) \leftarrow A \wedge \theta A' \wedge \bar{s} = \theta\bar{s}'$, where θ is a substitution that maps the variables of A'_j to distinct unused variables and $\bar{s} = \theta\bar{s}'$ equates the terms in $\bar{s}$ with those in $\theta\bar{s}'$. We use $q \otimes q'$ to refer to an arbitrary join of q and q'.

Theorem 6.4, reduces containment to equivalence for queries with expandable aggregation functions.

Theorem 6.4 *Let q and q' be α-queries. Suppose that α is an expandable function. Then $q \subseteq q'$ if and only if $(q \otimes q) \equiv (q' \otimes q)$.*

Proof (Sketch). Consider a database $\mathcal{D}$ and a tuple $\bar{d}$. Suppose that q computes the bag B of values for $\bar{d}$ and q' computes the bag B' of values for $\bar{d}$. It is not difficult to show that in this case, $q \otimes q$ will compute the bag $B \otimes |B|$ for $\bar{d}$ and $q' \otimes q$ will compute the bag $B' \otimes |B|$ for $\bar{d}$. Since α is an expandable aggregation function, $\alpha(B) = \alpha(B')$ if and only if $\alpha(B \otimes |B|) = \alpha(B' \otimes |B|)$, for $|B| > 0$.

"$\Leftarrow$" Suppose that $q \otimes q \equiv q' \otimes q$. If q returns an aggregate value for $\bar{d}$, then $|B| > 0$. Therefore, $\alpha(B \otimes |B|) = \alpha(B' \otimes |B|)$ implies that $\alpha(B) = \alpha(B')$, i.e., q and q' return the same aggregate value for $\bar{d}$.

"$\Rightarrow$" Suppose that $q \subseteq q'$. If q does not return an aggregate value for $\bar{d}$, then both $q \otimes q$ and $q' \otimes q$ will not return an aggregate value for $\bar{d}$. Otherwise, q returns an aggregate value for $\bar{d}$, and q' returns the same aggregate value. Therefore, from $\alpha(B) = \alpha(B')$, we conclude that $\alpha(B \otimes |B|) = \alpha(B' \otimes |B|)$, i.e., $q \otimes q$ and $q' \otimes q$ return the same value for $\bar{d}$. □

(a) Positive relational queries

Agg. Function	Complexity
max, *percent*, *avg*	NP-complete
count, *sum*	GI-complete[2]

(b) Positive queries

Agg. Function	Complexity
max	Π_2^P-complete
count, *sum*, *percent*, *avg*[3]	in PSPACE

Table 1: Complexity of equivalence

7 Related Work

We briefly state known complexity results for the equivalence problem. Table 1(a) summarizes complexity results for positive relational aggregate queries. Table 1(b) summarizes complexity results for positive aggregate queries (that may have comparisons). The results from both tables appear in [23, 13, 10]. For proper quasilinear queries, equivalence can be determined in polynomial time [8]. For arbitrary α-queries, the complexity of equivalence depends on the complexity of determining validity of ordered α-identities [8].

Containment and equivalence for positive relational nonaggregate queries evaluated under bag and bag-set was studied in [6, 18]. In [17], the expressivity of logics that extend first-order logic by aggregation was studied. The problem of determining satisfiability of a conjunction of aggregation con-

[2]GI denotes the class of problems that are many-one reducible to the graph isomorphism problem.

[3]This complexity result is only known for *avg*-queries that do not have constants.

straints was considered in [24]. An interesting open issue is combining results on aggregate-query equivalence with that of [24] in the investigation of aggregate queries with a HAVING clause. Other related work includes [14, 10, 11] which study the view usability problem for aggregate queries. The results on view usability are based on characterizations for equivalence of aggregate queries over a set of views.

Acknowledgments

Many of the results in this paper were derived during the author's doctoral research [7]. The author wishes to thank Werner Nutt and Yehoshua Sagiv, who were both her doctoral advisors and collaborators, for their invaluable guidance during her research.

References

[1] D. Barbara and T. Imielinski. Sleepers and workaholics: Caching strategies in mobile environments. In *Proc. of SIGMOD*, 1994.

[2] M. Benedikt and L. Libkin. Exact and approximate aggregation in constraint query languages. In *Proc. of PODS*, 1999.

[3] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. of STOC*, 1977.

[4] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1), 1997.

[5] S. Chaudhuri, S. Krishnamurthy, S. Potarnianos, and K. Shim. Optimizing queries with materialized views. In *Proc. of ICDE*, 1995.

[6] S. Chaudhuri and M. Vardi. Optimization of real conjunctive queries. In *Proc. of PODS*, 1993.

[7] S. Cohen. *Equivalence, Containment and Rewriting of Aggregate Queries*. PhD thesis, Hebrew University of Jerusalem, Israel, 2004.

[8] S. Cohen, W. Nutt, and Y. Sagiv. Equivalences among aggregate queries with negation. *ACM Transactions on Computational Logic*. To appear.

[9] S. Cohen, W. Nutt, and Y. Sagiv. Containment of aggregate queries. In *Proc. of ICDT*, 2003.

[10] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *Proc. of PODS*, 1999.

[11] S. Cohen, W. Nutt, and A. Serebrenik. Algorithms for rewriting aggregate queries using views. In *Proc. of ADBIS-DASFAA*, 2000.

[12] A. Dobra, M. N. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In *Proc. of SIGMOD*, 2002.

[13] S. Grumbach, M. Rafanelli, and L. Tininini. On the equivalence and rewriting of aggregate queries. *Acta Informatica*, 4(8), 2004.

[14] S. Grumbach and L. Tininini. On the content of materialized aggregate views. *Journal of Computer and System Sciences*, 66(1), 2003.

[15] A. Gupta, V. Harinarayan, and D. Quass. Aggregate query processing in data warehouses. In *Proc. of VLDB*, 1995.

[16] A. Gupta and I. S. Mumick, editors. *Materialized Views—Techniques, Implementations and Applications*. MIT Press, 1999.

[17] L. Hella, L. Libkin, J. Nurmonen, and L. Wong. Logics with aggregate operators. *Journal of the ACM*, 48(4), 2001.

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

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

[20] A. Levy and Y. Sagiv. Semantic query optimization in datalog programs. In *Proc. of PODS*, 1995.

[21] A. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 5(2), 1995.

[22] S. Madden, R. Szewczyk, M. J. Franklin, and D. Culler. Supporting aggregate queries over ad-hoc wireless sensor networks. In *Proc. 4th IEEE Workshop on Mobile Computing Systems and Applications*, 2002.

[23] W. Nutt, Y. Sagiv, and S. Shurin. Deciding equivalences among aggregate queries. In *Proc. of PODS*, 1998.

[24] K. Ross, D. Srivastava, P. Stuckey, and S. Sudarshan. Foundations of aggregation constraints. In *Proc. 2nd Int. Workshop on Principles and Practice of Constraint Programming*, 1994.

[25] D. Srivastava, S. Dar, H. Jagadish, and A. Levy. Answering queries with aggregation using views. In *Proc. of VLDB*, 1996.

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

[27] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *Proc. of PODS*, 1992.