Deciding Equivalences among Conjunctive Aggregate Queries

SARA COHEN

Technion—Israel Institute of Technology, Haifa, Israel

WERNER NUTT

Free University of Bozen-Bolzano, Bozen, Italy

AND

YEHOSHUA SAGIV

Hebrew University, Jerusalem, Israel

Abstract. Equivalence of aggregate queries is investigated for the class of conjunctive queries with comparisons and the aggregate operators count, count-distinct, min, max, and sum. Essentially, this class contains unnested SQL queries with the above aggregate operators, with a `where` clause consisting of a conjunction of comparisons, and without a `having` clause. The comparisons are either interpreted over a domain with a dense order (like the rationals) or with a discrete order (like the integers). Characterizations of equivalence differ for the two cases. For queries with either max or min, equivalence is characterized in terms of dominance mappings, which can be viewed as a generalization of containment mappings. For queries with the count-distinct operator, a sufficient condition for equivalence is given in terms of equivalence of conjunctive queries under set semantics. For some special cases, it is shown that this condition is also necessary. For conjunctive queries with comparisons but without aggregation, equivalence under bag-set semantics is characterized in terms of isomorphism. This characterization essentially remains the same also for queries with the count operator. Moreover, this characterization also applies to queries with the sum operator if the queries have either constants or comparisons, but not both. In the general case (i.e., both comparisons and

S. Cohen was partially supported by the Israel Science Foundation (Grant No. 1032/05).
W. Nutt was partially supported by EPSRC (Grant No. GR/SS44839/01 MAGIK-I).
Y. Sagiv was partially supported by the Israel Science Foundation (Grant No. 893/05).

Authors' Addresses: S. Cohen, Faculty of Industrial Engineering and Management, Technion—Israel Institute of Technology, Technion City, Haifa 32000, Israel, e-mail: sarac@ie.technion.ac.il; W. Nutt, Faculty of Computer Science, Free University of Bozen-Bolzano, Dominikanerplatz 3, I-39100 Bozen, Italy, e-mail: nutt@inf.unibz.it; Y. Sagiv, The Selim and Rachel Benin School of Engineering and Computer Science, The Hebrew University of Jerusalem, Edmond J. Safra Campus, Jerusalem 91904, Israel, e-mail: sagiv@cs.huji.ac.il.

constants), the characterization of the equivalence of queries with the sum operator is more elaborate. All the characterizations given in the paper are decidable in polynomial space.

Categories and Subject Descriptors: F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*Decision problems*; H.2.3 [**Database Management**]: Languages—*Query languages*; H.2.4 [**Database Management**]: Systems—*Query processing*; *relational databases*

General Terms: Languages, Theory

Additional Key Words and Phrases: Aggregation, query equivalence, Datalog, bag-set semantics

ACM Reference Format:

Cohen, S., Nutt, W., and Sagiv, Y. 2007. Deciding equivalences among conjunctive aggregate queries. J. ACM 54, 2, Article 5 (April 2007), 50 pages. DOI = 10.1145/1219092.1219093 http://doi.acm.org/10.1145/1219092.1219093

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 [Chaudhuri and Dayal 1997; Gupta and Mumick 1999].

Aggregate queries have been studied in a variety of settings. Recently there has been much interest in querying and analyzing stream data. Such analysis often requires aggregate queries, in order to store running statistics on the data. The problem of processing aggregate queries over streams was considered in Dobra et al. [2002]. For networks of sensors, which produce streams of measurements, aggregate queries were studied as a data-reduction tool. Data reduction is important in sensor networks, since the cost of communication is often high [Madden et al. 2002]. Aggregation has also been studied in constraint databases [Benedikt and Libkin 1999]. Relational algebra and calculus extended by aggregation functions were considered in Klug [1982] and Özsoyoğlu et al. [1987].

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. Containment of nonaggregate queries over relational databases has been studied extensively for conjunctive queries [Chandra and Merlin 1977; Aho et al. 1979; Johnson and Klug 1983; Sagiv and Saraiya 1992; Chekuri and Rajaraman 2000], for conjunctive queries with comparisons [van der Meyden 1992; Levy and Sagiv 1995], for queries with union and difference [Sagiv and Yannakakis 1981], and for conjunctive queries defined by Datalog programs [Levy and Sagiv 1995; Levy et al. 1993; 2001; Calvanese 2003].Containment of queries over semistructured data has also been studied [Popa and Tannen 1999; Calvanese 2000; 2001; Miklau and Suciu 2002].

Considerable work has been done on efficiently computing aggregate queries (e.g., Chaudhuri et al. [1995], Gupta et al. [1995], and Srivastava et al. [1996]. 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 previously presented algorithms were based on sufficient conditions, and complete characterizations 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. In Nutt et al. [1998], we provided, for the first time, characterizations for deciding equivalence that apply to a large and significant class of aggregate queries. These characterizations were extended in Cohen et al. [1999] for aggregate queries with disjunctions and in Cohen et al. [2001; 2005] for aggregate queries with negation. In Cohen et al. [2003], we showed how to reduce containment of aggregate queries to equivalence for a wide class of aggregation functions. Characterizations of equivalences among queries with the aggregate operator average were presented in Grumbach et al. [1999].

Concepts seemingly similar to those considered in this article have been investigated in Hella et al. [1999], Benedikt and Keisler [1997], Etessami and Immerman [2000], which study the expressivity of logics that extend first-order logic by some form of aggregation. In particular, Hella et al. [1999] considers aggregation definable in terms of commutative monoids, and shows that formulas in this extended logic 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 Benedikt and Keisler [1997] and Etessami and Immerman [2000], first-order logic is extended by a counting operator. Both of these papers discuss expressivity bounds of the derived language. Benedikt and Keisler [1997] show that a count operator over elements does not express counting over terms and Etessami and Immerman [2000] show that tree isomorphism is not expressible in first-order logic with transitive closure and count.

This article is a significantly extended version of Nutt et al. [1998]. The results reported in Cohen et al. [1999; 2001; 2005; 2003] rely heavily on the characterizations presented in Nutt et al. [1998]. That paper, however, did not provide proofs. The proofs, which appear in this article, are based on novel techniques that highlight the structural differences between equivalences among aggregate queries as opposed to nonaggregate queries. Aggregate queries are evaluated in two phases: first a nonaggregate query retrieves data, which then are amalgamated by an aggregation function. This gives rise to two complications, which make the aggregate case considerably more difficult, and which characterizations for equivalence have to take into account: (1) different sets of data may result in the same aggregate value, and (2) the multiplicity by which data are retrieved has an influence on the aggregate result for functions like count and sum.

The class of queries that we investigate in this article consists of conjunctive queries with comparisons and aggregations. Essentially, this class contains unnested SQL queries with aggregations, with a `where` clause consisting of a conjunction of comparisons, and without a `having` clause. For the comparisons, we separately consider whether they are interpreted over a domain with a dense order (like the rationals) or with a discrete order (like the integers). Generally, our characterizations differ for the two cases and different techniques are needed to establish them. Our queries have the aggregation operators max, min, sum, count, and cntd (where "cntd" means "count-distinct").

This article is organized as follows: In Section 2, we introduce nonaggregate conjunctive queries, review their basic properties, and discuss set and bag-set semantics.

Traditionally, equivalence of conjunctive queries (without aggregation) has mostly been investigated under set semantics, which means that both the operands of a given query and the result are sets. Equivalence of this type is called *set equivalence*. However, for a general treatment of aggregate operators, it is necessary to consider bag-set semantics. Under that semantics, the operands of a query are sets, but the result is a bag.

The syntax and semantics of aggregate queries is presented in Section 3. We also show that for the purpose of deciding equivalence, a given query can be decomposed into several queries, such that each query has a single aggregation. Therefore, we deal separately with each aggregate operator, and we name types of queries according to that operator, for example, *max-query*.

In Section 4, we develop a normal form for sets of comparisons, called *reduced sets of comparisons,* and a similar normal form for queries with comparisons, called *reduced queries*. These normal forms are used in some of our characterizations and proofs. In Section 5, we briefly survey the equivalence characterizations and complexity results proven in this article. Thus, this section serves as a gentle introduction and a road-map to the equivalence results presented in this article.

In Section 6, we characterize equivalence of max-queries and min-queries in terms of dominance mappings, which can be viewed as a generalization of containment mappings. For queries with the count-distinct operator, we provide, in Section 7, a sufficient condition for equivalence in terms of equivalence of conjunctive queries under set semantics (i.e., set equivalence). For some special cases, we show that this condition is also necessary.

Equivalence of count-queries is essentially the same as equivalence of conjunctive queries with bag-set semantics. In Chaudhuri and Vardi [1993], equivalence of conjunctive queries (with neither comparisons nor constants) under bag-set semantics, called *bag-set equivalence*, was characterized in terms of isomorphisms. That result also applies to count-queries, since equivalence of count-queries is essentially the same as bag-set equivalence of nonaggregate queries. We provide a proof of this result that also allows for constants in the query. Moreover, we generalize the characterization to conjunctive queries with comparisons, using a different proof technique. These results are presented in Section 8.

Sum-queries are considered in Section 9. For sum-queries, bag-set equivalence is a sufficient condition for equivalence. However, a complete characterization of the equivalence of sum-queries is more elaborate than bag-set equivalence, and we break it into several subcases. For sum-queries without constants (but with comparisons involving only variables), equivalence can be characterized in terms of bag-set equivalence. Furthermore, the proof technique for this case also applies in the presence of integrity constraints. For the general case of sum-queries (i.e., both constants and comparisons), a more elaborate characterization is needed, which requires its own proof technique. As a special corollary, a simplified characterization, in terms of bag-set equivalence, is obtained for sum-queries with constants, but without comparisons.

2. *Preliminaries*

In this section, we introduce conjunctive queries and review their basic properties.

2.1. CONJUNCTIVE QUERIES. We assume that there is an infinite set of predicates (denoted as p, q, r). Each predicate has an arity. A *schema* Σ is a finite set of predicate symbols $\Sigma = \{p_1, \ldots, p_n\}$.

An *ordered domain* is a nonempty set with a linear order. In this article, we are only interested in the ordered domains of integers and rational numbers. We denote domains by the letter $\mathcal{I}$. We do not distinguish between the elements of an ordered domain and the symbols denoting them and call both *constants*. A *database* $\mathcal{D}$ for the schema Σ contains for each predicate p of arity k in Σ a finite k-ary relation $p^{\mathcal{D}}$ over constants.[1] Each relation contains a set of tuples. The set of constants that occur in a tuple of one of the relations $p_i^{\mathcal{D}}$ is called the *carrier* of $\mathcal{D}$ and is denoted as $|\mathcal{D}|$.

We assume that there is an infinite set of *variables* (denoted as w, x, y, z). A *term* is either a variable or a constant. We denote terms as s, t. A *relational atom* has the form $p(s_1, \ldots, s_k)$, where p is a predicate of arity k. The *ordering predicates* are $<$, $\leq$, $>$, and $\geq$. We will use the equality $s = t$ as an abbreviation for the conjunction $s \leq t \wedge t \leq s$. Ordering predicates are interpreted over the domain underlying the database. An *ordering atom* or *comparison* has the form $s_1 \ \rho \ s_2$, where ρ is an ordering predicate. The comparisons $s_1 < s_2$ and $s_1 > s_2$ are *strict,* while $s_1 \leq s_2$ and $s_1 \geq s_2$ are *nonstrict.*

For relational atoms, we also use the notation $p(\bar{s})$, where $\bar{s}$ stands for a tuple of terms. Similarly, $\bar{x}$ stands for a tuple of variables. We will often identify a tuple $\bar{s}$ with the set of terms $\{s_1, \ldots, s_k\}$ occurring in the tuple. It is often convenient to view a database as a set of finitely many ground relational atoms $p(\bar{d})$, that is, atoms with only constants.

An *atom* is a relational atom or a comparison. We denote atoms as a, b. We write $a(x_1, \ldots, x_m)$ or $a(\bar{x})$ to indicate that $x_1, \ldots, x_m$ or the variables in $\bar{x}$, respectively, are the variables occurring in a. A *condition* is a conjunction $a_1 \wedge \cdots \wedge a_n$ of atoms. A condition is *safe* if every variable that occurs in a comparison also occurs in a relational atom.

A *conjunctive query* is an expression of the form

$$q(s_1, \ldots, s_k) \leftarrow a_1(\bar{w}_1) \wedge \cdots \wedge a_n(\bar{w}_n), \tag{1}$$

where $s_1, \ldots, s_k$ are terms, each variable among the s_i occurs in one of the atoms $a_1, \ldots, a_n$, and the condition $a_1 \wedge \cdots \wedge a_n$ is safe. We emphasize that we generally use the term *conjunctive query* to refer to a nonaggregate query, as in Eq. (1), unless it is clearly otherwise from the context.

The atom $q(s_1, \ldots, s_k)$ is called the *head* of the query, and the condition $a_1 \wedge \cdots \wedge a_n$ is called the *body* of the query. The terms in the head are called *output terms* and variables in the head are called *output variables*. Variables that occur in the body, but don't occur in the head are called *existential* variables. The atoms in the body can be relational atoms or comparisons, and the arguments can be variables as well as constants. If the body contains no comparisons, then the query is *relational*.

[1] We consider databases over ordered domains because our queries may contain comparisons. To simplify the exposition, we restrict ourselves to integers and rationals, which are prototypical for domains with strict and dense order. All results can easily be generalized to databases where the attributes of relations have several types.

We abbreviate a query as

$$q(\bar{s}) \leftarrow B(\bar{w}),$$

where $B(\bar{w})$ stands for the body of the query and $\bar{w}$ for the variables occurring in the body. If the variables are not important, we simply write $q(\bar{s}) \leftarrow B$. By abuse of notation, we will often refer to a query by its head $q(\bar{s})$ or simply by the predicate of its head q.

2.2. SEMANTICS OF CONJUNCTIVE QUERIES. An *assignment* γ for the query $q(\bar{s}) \leftarrow B(\bar{w})$ over the ordered domain $\mathcal{I}$ is a mapping $\gamma : \bar{w} \rightarrow \mathcal{I}$ from the set of variables occurring in q to elements of the domain $\mathcal{I}$. For a constant d we define $\gamma(d) := d$. For $\bar{s} = (s_1, \dots, s_k)$ we let $\gamma(\bar{s})$ denote the tuple $(\gamma(s_1), \dots, \gamma(s_k))$. Assignments are extended to other syntactic objects in the obvious way.

An assignment γ *satisfies* the comparison $s_1 \ \rho \ s_2$ over $\mathcal{I}$ if $\gamma(s_1) \ \rho \ \gamma(s_2)$ holds over $\mathcal{I}$, and γ *satisfies* the relational atom a over the database $\mathcal{D}$ if $\gamma(a) \in \mathcal{D}$. Satisfaction of conjunctions of atoms is defined as one would expect. The domain $\mathcal{I}$ and the database $\mathcal{D}$ are not mentioned if they are clear from the context.

2.2.1. *Set Semantics.* Under *set semantics,* a conjunctive query $q(\bar{s}) \leftarrow B(\bar{w})$ defines a new relation $q^{\mathcal{D}}$ over the carrier of a database $\mathcal{D}$ as follows:

$$q^{\mathcal{D}} := \{\gamma(\bar{s}) \mid \gamma \text{ satisfies } B(\bar{w})\}.$$

This means, we obtain $q^{\mathcal{D}}$ by restricting the assignments satisfying the body of the query to the tuple of variables appearing in the head.

2.2.2. *Bag-Set-Semantics.* Bag-set semantics has been introduced by Chaudhuri and Vardi [1993] to semantically model query execution by SQL-based database systems. There, the database contains relations, that is, sets of tuples, while a query returns a bag of tuples, that is, a multiset, where a tuple can occur more than once. This differs from bag semantics, for example, Albert [1991] and Ioannidis and Ramakrishnan [1995], in which the database is a multiset of tuples.

Under *bag-set semantics,* a conjunctive query $q(\bar{s}) \leftarrow B(\bar{w})$ defines a multiset $\{\!\{ q \}\!\}^{\mathcal{D}}$ of tuples over the carrier of a database $\mathcal{D}$. The bag $\{\!\{ q \}\!\}^{\mathcal{D}}$ contains the same tuples as the relation $q^{\mathcal{D}}$, but each tuple $\gamma(\bar{s})$ occurs as many times as there are assignments γ' that satisfy $B(\bar{w})$ and agree with γ on $\bar{s}$. Letting $\{\!\{ \cdot \}\!\}$ denote a multiset, we formally define $\{\!\{ q \}\!\}^{\mathcal{D}}$ in analogy to $q^{\mathcal{D}}$ as:

$$\{\!\{ q \}\!\}^{\mathcal{D}} := \{\!\{ \gamma(\bar{s}) \mid \gamma \text{ satisfies } B(\bar{w}) \}\!\}.$$

2.2.3. *Equivalence and Containment.* A query q is *contained* in a query q' under set-semantics if over every database the set of results returned by q is a subset of the results returned by q'. Formally, q is contained in q' if $q^{\mathcal{D}} \subseteq q'^{\mathcal{D}}$ for all databases $\mathcal{D}$.

Two queries q and q' are *equivalent* under set-semantics, or *set-equivalent*, if over every database they return the same sets of results. Obviously, two queries are set-equivalent if and only if they contain each other.

Similarly, q and q' are *equivalent* under bag-set-semantics, or *bag-set-equivalent*, if over every database they return the same results with the same multiplicities, that is, $\{\!\{ q \}\!\}^{\mathcal{D}} = \{\!\{ q' \}\!\}^{\mathcal{D}}$ for all databases $\mathcal{D}$.

2.3. CONTAINMENT AND QUERY HOMOMORPHISMS. Containment and equivalence of conjunctive queries can be characterized in terms of query homomorphisms (see Chandra and Merlin [1977]). Also, the characterizations for the equivalence of aggregate queries in this paper will be formulated in terms of homomorphisms and isomorphisms.

For a conjunctive query with comparisons, we distinguish between the *relational part* of the body and the *comparisons*. We write such a query as

$$q(\bar{s}) \leftarrow R(\bar{w}) \wedge C(\bar{z}),$$

where $R(\bar{w})$ is the conjunction of all relational atoms in the body and $C(\bar{z})$ is the conjunction of all comparisons. The notation $R(\bar{w})$ and $C(\bar{z})$ indicates that only variables in the tuples $\bar{w}$ or $\bar{z}$ occur in the relational part or the comparisons, respectively. We often omit the tuples.

If C and C' are conjunctions of comparisons and $\mathcal{I}$ is an ordered domain, we write $C \models_{\mathcal{I}} C'$ if C' is a *consequence* of C over $\mathcal{I}$. Note, that for different domains the consequences of C may be different. For instance, $x > 0 \models_{\mathbf{Z}} x \geq 1$ holds over the integers, but does not hold over the rationals. If the domain is clear from the context, we write "$\models$" instead of "$\models_{\mathcal{I}}$".

Let $q(\bar{s}) \leftarrow R \wedge C$ and $q'(\bar{s}') \leftarrow R' \wedge C'$ be conjunctive queries with comparisons, ranging over the domain $\mathcal{I}$. A *homomorphism* from q' to q is a substitution θ of the variables in q' by terms in q such that

(1) $\theta\bar{s}' = \bar{s}$;

(2) $\theta a'$ is in R for every relational atom a' of R';

(3) $C \models_{\mathcal{I}} \theta(s')\ \rho\ \theta(t')$ for every comparison $s'\ \rho\ t'$ in C'.

In a loose notation, we write the last two conditions as (2) $\theta R' \subseteq R$ and (3) $C \models_{\mathcal{I}} \theta C'$. A substitution θ is a *relational homomorphism* from q' to q if it is a *homomorphism* from $\tilde{q}'(\bar{s}) \leftarrow R'$ to $\tilde{q}(\bar{s}') \leftarrow R$, that is, if it satisfies Conditions (1) and (2) in the definition of homomorphism. The set of all homomorphisms from q' to q is denoted as $Hom(q', q)$.

The following classical theorem, which is due to Chandra and Merlin [1977], relates homomorphisms and containment.

THEOREM 2.1 (HOMOMORPHISMS AND CONTAINMENT). *Let q and q' be conjunctive queries.*

—*If there is homomorphism from q' to q, then q is contained in q';*

—*If q and q' are relational queries, and q is contained in q', then there exists a homomorphism from q' to q.*

Because of Theorem 2.1, homomorphisms between queries are also called *containment mappings*. Note that the theorem completely characterizes containment between relational queries, but only provides a sufficient criterion for the containment of queries with comparisons. Checking containment of conjunctive queries with comparisons is more complicated and described in Section 2.4. Set-equivalence of relational conjunctive queries q and q' can be decided by checking whether there are homomorphisms from q' to q and from q' to q. Note that even if such homomorphisms exist, the queries q and q' may not necessarily be isomorphic. Deciding whether there exists a homomorphism from one relational query to the other is NP-complete.

A homomorphism is *injective* if it maps different variables in q' to different variables in q.[2] It is *surjective* if every variable in q is the image of a variable in q'. It is *bijective* if it is injective and surjective. If it is bijective, then it has an inverse, which is a mapping from the variables in q to the variables in q'. Note that the inverse need not be a homomorphism. A homomorphism is an *isomorphism* if it is bijective and if its inverse is also a homomorphism. Thus, a bijective homomorphism θ from q' to q is an isomorphism if for every atom a in the body of q there is an atom in the body of q' such that $a = \theta a'$. The queries q' and q are *isomorphic* if there is an isomorphism from q' to q.[3] Thus isomorphic queries are identical up to a renaming of the existential variables and up to the multiplicity of atoms.

2.4. CONTAINMENT OF QUERIES WITH COMPARISONS. For queries with comparisons, the existence of a homomorphism is only a sufficient, but not a necessary condition for containment. A classical example illustrating this fact consists of the two queries[4]

$$\begin{aligned} q \ &\leftarrow r(x, y) \wedge r(y, x) \\ q' \ &\leftarrow r(x, y) \wedge x \leq y. \end{aligned}$$

One readily checks that q is contained in q', but that there is no homomorphism from q' to q. This is because the body of q contains no information about the relationship between x and y. There are three mutually exclusive ways in which the terms occurring in q can be linearly ordered:

$$x < y, \qquad x = y, \qquad y < x.$$

If we add these three linear orderings to q, we obtain the three queries

$$\begin{aligned} q_1 \ &\leftarrow r(x, y) \wedge r(y, x) \wedge x < y \\ q_2 \ &\leftarrow r(x, y) \wedge r(y, x) \wedge x = y \\ q_3 \ &\leftarrow r(x, y) \wedge r(y, x) \wedge y < x. \end{aligned}$$

If now we define $\theta_1 := \theta_2 := \{x \mapsto x,\ y \mapsto y\}$ and $\theta_3 := \{x \mapsto y,\ y \mapsto x\}$, then each θ_i is a homomorphism from q' to q_i for $i \in 1, \ldots, 3$.

In the preceding example, we have rewritten q as a disjoint union of queries q_i, each of which is contained in q'. Hence, q is contained in q'. To turn this idea into a formal characterization of containment, we define linearizations of sets of terms, linearizations of queries, and linear expansions of queries. These definitions will also be used later on to characterize containment of *max*, *count*, and *sum*-queries.

Let $\mathcal{I}$ be an ordered domain, D be a set of constants from $\mathcal{I}$, W be a set of variables, and let $T := D \cup W$ denote their union. A *linearization* of T over $\mathcal{I}$ is a set of comparisons L over the terms in T such that for any $s, t \in T$ exactly one of

[2] Observe that we require an injective homomorphism to map variables to variables, and not just to terms. Actually, if we consider a homomorphism not only as a mapping on variables, but on terms, then it is evident that in order to be injective it has to map variables to variables, since it maps every constant to itself.
[3] Note that our definition of isomorphism of queries does not depend on the multiplicity with which an atom occurs in a query.
[4] A query with no terms in the head returns the empty tuple if there is an assignment that satisfies the body.

the following holds:

—$L \models_{\mathcal{I}} s < t$
—$L \models_{\mathcal{I}} s = t$
—$L \models_{\mathcal{I}} s > t$.

Thus, a linearization partitions the terms into classes such that the terms in each class are equal and the classes are arranged in a strict linear order. In each class of L, there is at most one constant. Otherwise, L would be unsatisfiable and entail any consequence. Note that the domain is crucial for determining whether L is a linearization or not. For instance,

$$L := \{1 < x,\ x < 2\}$$

is a linearization of $\{1, 2, x\}$ over rational numbers, but not over the integers because over the integers, L is unsatisfiable. A linearization L of T over $\mathcal{I}$ is *compatible* with a set of comparisons C if $L \cup C$ is satisfiable over $\mathcal{I}$.

For technical reasons, when checking containment of two queries, we have to consider linearizations that contains the constants of both queries. Therefore, we define linearizations with respect to a set of constants. Let

$$q(\bar{s}) \leftarrow R \wedge C$$

be a query with comparisons ranging over $\mathcal{I}$, W be the set of variables occurring in q, and D be a set of constants from $\mathcal{I}$ that comprise the constants of q. Then, we denote with $\mathcal{L}_D(q)$ the set of all linearizations of $D \cup W$ that are compatible with the comparisons C of q.

Now, let L be a linearization of $T = D \cup W$ that is compatible with C. As pointed out before, L defines an equivalence relation on T, where each equivalence class contains at most one constant. A substitution ϕ is *canonical* for L if it maps all elements in an equivalence class of L to one term of that class, and, if a class contains a constant, then it maps the class to that constant.

By means of a canonical substitution ϕ for L, we transform the query q into a more special query q_L. The query q_L has the form

$$q_L(\phi(\bar{s})) \leftarrow \phi R \wedge \phi L,$$

that is, it is obtained from q by first replacing C with L and then eliminating all equalities by applying a canonical substitution ϕ. We call q_L a *linearization* of q with respect to L. There may be more than one linearization of q with respect to L, but all linearizations are isomorphic. Note that ϕ is a homomorphism from q to q_L.

A *linear expansion* of q over D is a family of queries $(q_L)_{L \in \mathcal{L}_D(q)}$, where each q_L is a linearization of q with respect to L. If q and D are clear from the context, or do not matter, we write simply $(q_L)_L$. Let $(q_L)_L$ and $(q'_M)_M$ be linear expansions of q and q' over D, respectively. We say that $(q_L)_L$ and $(q'_M)_M$ are *isomorphic* if there is a bijection $\mu: \mathcal{L}_D(q) \to \mathcal{L}_D(q')$ such that q_L and $q'_{\mu(L)}$ are isomorphic for all $L \in \mathcal{L}_D(q)$.

We demonstrate the notions introduced in this section with the following example.

Example 2.2. Consider the queries q and q'

$$\begin{aligned} q &\leftarrow r(x, y) \wedge r(y, x) \\ q' &\leftarrow r(x, y) \wedge x \leq y. \end{aligned}$$

A linear expansion of q is the family of queries

$$\begin{aligned} q_1 &\leftarrow r(x,y) \wedge r(y,x) \wedge x < y \\ q_2 &\leftarrow r(x,x) \\ q_3 &\leftarrow r(x,y) \wedge r(y,x) \wedge y < x. \end{aligned}$$

Note that q_2 is derived by considering the linearization $x = y$ of the terms of q, and choosing the canonical substitution ϕ in which y is substituted with x. A linear expansion of q' is the family of queries

$$\begin{aligned} q'_1 &\leftarrow r(x,y) \wedge r(y,x) \wedge x < y \\ q'_2 &\leftarrow r(x,x). \end{aligned}$$

Note that the linear expansion of q' contains only two queries since there are only two linearizations of the terms in q' that are compatible with the comparisons of q'.

For every query in the linear expansion of q, there is an isomorphic query in the linear expansion of q', and vice versa. However, these linear expansions are not isomorphic since they contain a different number of queries, and hence, the required bijection μ cannot exist.

For a given query q, there is no unique linear expansion over a set of constants D, because the canonical substitutions that produce the linearizations q_L are in general not uniquely determined. However, it is easy to see that any two such linear expansions are isomorphic.

The following is a reformulation and extension of a theorem by Klug [Klug 1988].

THEOREM 2.3 (CONTAINMENT WITH COMPARISONS). *Let q, q' be two conjunctive queries with comparisons, D be the set of constants occurring in q and q', and $(q_L)_L$ be the linear expansion of q over D. Then the following are equivalent:*

—*q is contained in q';*
—*for every q_L there is a homomorphism from q' to q_L.*

Clearly, in the above theorem, linear expansions and homomorphisms have to be taken with respect to the ordered domain over which the queries range.

Note that for queries with comparisons the characterization of containment is more complex than for relational queries, since there have to exist as many homomorphisms as there are queries in a linear expansion of q. In fact, it has been shown by van der Meyden [1992] that containment of conjunctive queries with comparisons is Π_2^{P}-complete.

3. *Aggregate Queries*

We give an abstract account of aggregate queries as they are definable in SQL without nesting and without using the `having` construct.

3.1. SYNTAX OF AGGREGATE QUERIES.

We consider the *aggregation functions* *min*, *max*, *count*, *cntd*, *sum*. The function *cntd* is read as "count distinct." Aggregation functions are abstractly denoted as α. An *aggregate term* has one of the forms

$$\mathit{min}(y),\ \mathit{max}(y),\ \mathit{cntd}(y),\ \mathit{count},\ \mathit{sum}(y).$$

Observe that *count* does not take an argument. Aggregate terms are abstractly denoted as $\alpha(y)$.

An *aggregate query* has the form

$$q(x_1, \ldots, x_k, \alpha_1(y_1), \ldots, \alpha_l(y_l)) \leftarrow B(\bar{w}), \qquad (2)$$

where

—$x_1, \ldots, x_k$ are distinct variables;

—the sets of variables $\{x_1, \ldots, x_k\}$ and $\{y_1, \ldots, y_l\}$ are disjoint;

—each of the variables x_i and y_j occurs in the body $B(\bar{w})$ of the query.

We call $\{x_1, \ldots, x_k\}$ the *grouping variables* and $\{y_1, \ldots, y_l\}$ the *aggregation variables* of the query. Similarly as for conjunctive queries, we distinguish between relational aggregate queries and arbitrary aggregate queries, which may contain comparisons in the body. Instead of (2), we use as a shorthand the notation

$$q(\bar{x}, \bar{\alpha}(\bar{y})) \leftarrow B(\bar{w}).$$

Note that, to simplify our presentation, our aggregate queries do not have constants in their heads. Standard conjunctive (nonaggregate) queries can have constants in their heads (see Eq. (1)). We will never directly consider linearizations of an aggregate query, since linearizations may introduce constants into the head of a query. Instead, we will consider linearizations of the *cores* of aggregate queries. Essentially, the core of an aggregate query is a conjunctive query derived by stripping off the aggregate functions in the query. This notion will be defined formally later on.

3.2. SEMANTICS OF AGGREGATE QUERIES. Consider the aggregate query $q(\bar{x}, \bar{\alpha}(\bar{y})) \leftarrow B(\bar{w})$. For a database $\mathcal{D}$, the query yields a new relation $q^{\mathcal{D}}$. To define the relation $q^{\mathcal{D}}$, we proceed in two steps.

First, we partition the set of assignments satisfying the body $B(\bar{w})$ of the query into equivalence classes. Two assignments γ_1, γ_2 are equivalent, $\gamma_1 \sim_q \gamma_2$, if they agree on $\bar{x}$, that is, $\gamma_1(\bar{x}) = \gamma_2(\bar{x})$. We denote the equivalence class of γ under this relation as $[\gamma]_q$. Obviously, the class $[\gamma]_q$ is uniquely determined by the tuple $\bar{d} := \gamma(\bar{x})$. If q is clear from the context, we drop the subscript.

For each such $\bar{d}$ the group $[\gamma]_q$ gives rise to the multiset of values assigned to the variables $\bar{y}$ by assignments in that group,

$$\{\!\{\, \gamma(\bar{y}) \mid \gamma(\bar{x}) = \bar{d} \text{ and } \gamma \text{ satisfies the body of } q \,\}\!\}.$$

We will refer to this multiset as the *group* of $\bar{y}$-values for $\bar{d}$ w.r.t. q, or simply, if no misunderstanding can arise, as the group of $\bar{d}$.

Next, we define how to evaluate an aggregate term $\alpha(y)$ on a class of assignments $[\gamma]$, written $\alpha(y).[\gamma]$:

$$\begin{aligned} \mathit{max}(y).[\gamma] &:= \max_{\gamma' \in [\gamma]} \gamma'(y) \\ \mathit{min}(y).[\gamma] &:= \min_{\gamma' \in [\gamma]} \gamma'(y) \\ \mathit{cntd}(y).[\gamma] &:= |\{\gamma'(y) \mid \gamma' \in [\gamma]\}| \end{aligned}$$

$$\begin{aligned} \mathit{count}.[\gamma] &:= |[\gamma]| \\ \mathit{sum}(y).[\gamma] &:= \sum_{\gamma' \in [\gamma]} \gamma'(y). \end{aligned}$$

In order for these definitions to make sense in a general setting, assignments have to be suitably typed so that maxima and minima are taken over an ordered set of values and sums are taken over numbers. Obviously, well-typedness of assignments can be guaranteed if database relations and queries are well-typed. Since it is obvious how to define and test such well-typedness, we do not dwell on this issue in this paper. Note that the function $\mathit{cntd}(y)$ returns the number of distinct values to which y is bound.

A tuple of aggregate terms is evaluated on an assignment class by evaluating each component. That is, if $\bar{\alpha}(\bar{y}) = (\alpha_1(y_1), \ldots, \alpha_l(y_l))$ is a tuple of aggregate terms, then $\bar{\alpha}(\bar{y}).[\gamma]$ is defined as the tuple $(\alpha_1(y_1).[\gamma], \ldots, \alpha_l(y_l).[\gamma])$.

Finally, the query $q(\bar{x}, \bar{\alpha}(\bar{y})) \leftarrow B(\bar{w})$ is evaluated on $\mathcal{D}$ by first partitioning the assignments satisfying the body into equivalence classes and then, for each class $[\gamma]_q$, concatenating the characteristic tuple of values $\gamma(\bar{x})$ with the evaluation of $\bar{\alpha}(\bar{y})$ on $[\gamma]_q$. Formally,

$$q^{\mathcal{D}} := \{(\gamma(\bar{x}),\ \bar{\alpha}(\bar{y}).[\gamma]_q) | \gamma \text{ satisfies } B(\bar{w})\},$$

where $(\gamma(\bar{x}),\ \bar{\alpha}(\bar{y}).[\gamma]_q)$ denotes the concatenation of the tuples $\gamma(\bar{x})$ and $\bar{\alpha}(\bar{y}).[\gamma]_q$.

The class of aggregate queries considered in this paper corresponds to unnested SQL queries with aggregations, in which (1) the `where` clause consists of a conjunction of comparisons, (2) all attributes in the `group by` clause also appear in the `select` clause, and (3) there is no `having` clause. SQL queries, satisfying the above description, can be translated into our notation similarly to the way that SQL queries without aggregation are translated into conjunctive queries. We demonstrate this translation with an example.

Example 3.1. Let $R(X, Y)$ and $S(U, V, W)$ be relations. Consider the following SQL query.

```
  select R.X, count, sum(S.V), avg(S.W)
    from R, S
   where R.Y = S.U AND S.V < 10
group by R.X
```

This query is written in our notation as

$$q(x, \mathit{count}, \mathit{sum}(v), \mathit{avg}(w)) \leftarrow r(x, y) \wedge s(u, v, w) \wedge y = u \wedge v < 10\,.$$

3.3. EQUIVALENCE OF AGGREGATE QUERIES. Consider two aggregate queries

$$\begin{aligned} q(\bar{x}, \bar{\alpha}(\bar{y})) &\leftarrow B(\bar{w}) \\ q'(\bar{x}', \bar{\alpha}'(\bar{y}')) &\leftarrow B'(\bar{w}'). \end{aligned}$$

We say that q and q' are *equivalent* if for every database $\mathcal{D}$ the two queries define the same relation, that is, $q^{\mathcal{D}} = q'^{\mathcal{D}}$.

There are cases where two queries are equivalent, even if they contain different aggregation functions, and the tuples of grouping variables have different length.

As an example consider the following two queries:

$$\begin{aligned} q(x, y, max(z)) &\leftarrow r(x, y, z) \wedge x = y \wedge y = z \\ q'(x, sum(y), min(z)) &\leftarrow r(x, y, z) \wedge x = y \wedge y = z. \end{aligned}$$

To exclude such pathological cases, we restrict ourselves to *comparable* queries. We say that two aggregate queries $q(\bar{x}, \bar{\alpha}(\bar{y}))$ and $q'(\bar{x}', \bar{\alpha}'(\bar{y}'))$ are *comparable* if the tuples of grouping variables are the same, that is, $\bar{x} = \bar{x}'$, the tuples of aggregate terms $\bar{\alpha}(\bar{y})$ and $\bar{\alpha}'(\bar{y}')$ have the same length, say l, and corresponding aggregate terms have the same aggregation function, that is, $\alpha_j = \alpha'_j$ for $j = 1, \ldots, l$.

In the following, we will show that in order to study the equivalence of aggregate queries, we can concentrate on queries with a single aggregate term. The intuitive reason is that the values of the aggregate terms in a query result are functionally dependent on the values of the grouping variables. Thus, we can treat each aggregate term in a query separately.

Let $q(\bar{x}, \bar{\alpha}(\bar{y})) \leftarrow B(\bar{w})$ be an aggregate query. For each aggregate term $\alpha_j(y_j)$ appearing in the head of q we define a new query q_j, the *jth kernel* of q, by

$$q_j(\bar{x}, \alpha_j(y_j)) \leftarrow B(\bar{w}).$$

Essentially, q_j is obtained from q by projecting out all aggregate terms different from $\alpha_j(y_j)$. The next proposition reduces the equivalence problem for arbitrary aggregate queries to the equivalence problem of their kernels.

PROPOSITION 3.2 (REDUCTION TO KERNELS). *Let q and q' be two comparable aggregate queries. Then q and q' are equivalent if and only if the kernels q_j and q'_j are equivalent for all j.*

PROOF. Obviously, if q and q' are equivalent, then so are q_j and q'_j, since the relations they define are projections of those defined by q and q'. For the converse claim, it suffices to show that q is contained in q'. The containment of q' follows by a symmetric argument.

Suppose that q and q' are defined as $q(\bar{x}, \bar{\alpha}(\bar{y})) \leftarrow B(\bar{w})$ and $q'(\bar{x}, \bar{\alpha}(\bar{y}')) \leftarrow B'(\bar{w}')$, respectively. Let $\mathcal{D}$ be a database and γ be an assignment that satisfies $B(\bar{w})$. Then, $q^{\mathcal{D}}$ contains the tuple $(\gamma(\bar{x}),\ \bar{\alpha}(\bar{y}).[\gamma]_q)$, and $q_1^{\mathcal{D}}$, the set of answers returned by the first kernel of q, contains the tuple $(\gamma(\bar{x}),\ \alpha_1(y_1).[\gamma]_{q_1})$. Since q_1 and q'_1 are equivalent, they return the same answers over $\mathcal{D}$. Thus, there is an assignment γ' such that γ' satisfies $B'(\bar{w}')$ and $(\gamma(\bar{x}),\ \alpha_1(y_1).[\gamma]_{q_1}) = (\gamma'(\bar{x}),\ \alpha_1(y'_1).[\gamma']_{q'_1})$.

To prove our claim, it suffices to show that

$$\alpha_j(y_j).[\gamma]_q \ = \ \alpha_j(y'_j).[\gamma']_{q'} \tag{3}$$

for all j. To see this, first observe that for all j we have

$$[\gamma]_q = [\gamma]_{q_j} \quad \text{and} \quad [\gamma']_{q'} = [\gamma']_{q'_j}, \tag{4}$$

since a query and its kernels have the same body and the same grouping variables. Now, because $\gamma(\bar{x}) = \gamma'(\bar{x})$, the equivalence of the kernels of q and q' implies that

$$\alpha_j(y_j).[\gamma]_{q_j} \ = \ \alpha_j(y'_j).[\gamma']_{q'_j} \tag{5}$$

for all j, which, together with (4), yields (3). □

In the proof, we have only used that fact that two comparable queries have the same tuple of grouping variables. That corresponding aggregate terms have the same aggregation function was not important.

Proposition 3.2 shows us that in order to study the equivalence of aggregate queries we can concentrate on queries with a single aggregate term. We call such a query, which has the form $q(\bar{x}, \alpha(y))$, a *simple* aggregate query.

Our technique for studying the equivalence of such queries will be to associate to every simple aggregate query a conjunctive query and to reduce the equivalence of simple aggregate queries to syntactic properties of the associated queries.

If $q(\bar{x}, \alpha(y)) \leftarrow B(\bar{w})$ is an aggregate query with the aggregation function *max*, *min*, *cntd*, or *sum*, then the *core* of q is the conjunctive query

$$\breve{q}(\bar{x}, y) \;\leftarrow\; B(\bar{w}).$$

The *core* of a query $q(\bar{x}, \mathit{count}) \leftarrow B(\bar{w})$ is the conjunctive query

$$\breve{q}(\bar{x}) \;\leftarrow\; B(\bar{w}).$$

Observe that for aggregation functions with an argument, the argument appears in the head of the core, and that for *count*, which does not have an argument, the head of the core contains only the grouping variables.

4. *Reduced Queries*

In this section, we introduce a normal form for conjunctive queries with comparisons, called reduced queries, which we will need for our characterizations in later sections. We show that the normal form can be computed in polynomial time. Reduced queries are motivated by the following technical considerations.

In a relational query, it is always the case that, over a suitable database, a variable can be bound to more than one data element, and different variables can be bound to distinct constants. However, comparisons may entail that a variable is equal to a constant, or that two variables are equal. Moreover, comparisons may render a query unsatisfiable. For our technical arguments later on to be valid we have to exclude such queries. In this section, we show how to deal with this problem.

4.1. CONJUNCTIVE QUERIES. A set of comparisons C is *reduced* over a domain $\mathcal{I}$ if it is satisfiable, and

(1) for every two distinct variables y, z occurring in C, it holds that $C \not\models_{\mathcal{I}} y = z$, that is, there is an assignment γ satisfying C such that $\gamma(y) \neq \gamma(z)$;
(2) for every variable y occurring in C and every constant $d \in \mathcal{I}$, it holds that $C \not\models_{\mathcal{I}} y = d$, that is, there is an assignment γ satisfying C such that $\gamma(y) \neq d$.

In other words, a set of comparisons is reduced if it does not imply equality of distinct terms. A conjunctive query $q(\bar{x}) \leftarrow R \wedge C$ is *reduced* if C is a reduced set of comparisons.

In Guo et al. [1996a; 1996b], the problems of determining satisfiability and implication of comparisons over both the rationals and the integers, were considered. The comparisons considered in Guo et al. [1996b] were of exactly the same type as those considered here. These results can immediately be applied to find reduced queries. Before stating the relevant results of Guo et al. [1996b], we present some necessary definitions.

Let C be a set of comparisons. We are interested in two properties of C (which will be used in conjunction with the two requirements of reduced sets of comparisons).

—*Equivalence Relation among Variables.* As pointed out earlier, the comparisons C naturally imply an equivalence relation over the variables in C. Formally, we will say that x and y are in the same equivalence class according to C over the domain $\mathcal{I}$, denoted $x \equiv_{C,\mathcal{I}} y$ if $C \models_{\mathcal{I}} x = y$. If the domain is clear from the context, we omit $\mathcal{I}$ from the subscript.

—*Lower and Upper Bounds.* We associate each variable y appearing in C with two values $gll_C(y)$ and $lul_C(y)$, defined as follows

$$\begin{aligned} gll_C(y) &= \inf\{\gamma(y) \mid \gamma \text{ satisfies } C\} \\ lul_C(y) &= \sup\{\gamma(y) \mid \gamma \text{ satisfies } C\}. \end{aligned}$$

Note that by definition $gll_C(y) = -\infty$ if y is not bounded from below, and $lul_C(y) = \infty$ if y is not bounded from above. Note also that if C is satisfiable and $gll_C(y) = lul_C(y)$, then $C \models y = gll_C(y)$.

For example, consider the set of comparisons

$$C = \{5 < x,\ x \leq y,\ y \leq x,\ x < z\}.$$

Over both the integers and the rationals, we have that $x \equiv_C y$. Over the integers, $gll_C(x) = 6$, $lul_C(x) = \infty$, (and therefore $gll_C(y) = 6$ and $lul_C(y) = \infty$), and $gll_C(z) = 7$, $lul_C(z) = \infty$. Over the rationals, $gll_C(x) = gll_C(y) = gll_C(z) = 5$ and $lul_C(x) = lul_C(y) = lul_C(z) = \infty$.

The following result is from Guo et al. [1996b].

THEOREM 4.1 (PROPERTIES OF COMPARISONS). *Let C be a set of comparisons ranging over the rationals or the integers.*

(1) *It is possible to determine if C is satisfiable in time $\mathcal{O}(|C|)$.*
(2) *It is possible to compute the equivalence relation $\equiv_C$ in time $\mathcal{O}(|C|)$.*
(3) *It is possible to compute $gll_C(y)$ and $lul_C(y)$, for all variables y in C, in time $\mathcal{O}(|C|)$.*
(4) *Suppose that C ranges over the integers. Then there exist assignments $\gamma^{\downarrow}$ and $\gamma^{\uparrow}$ satisfying C such that*
   —$\gamma^{\downarrow}(y) = gll_C(y)$, *whenever* $gll_C(y) \neq -\infty$
   —$\gamma^{\uparrow}(y) = lul_C(y)$, *whenever* $lul_C(y) \neq \infty$.
(5) *Let $s\ \rho\ t$ be a comparison. Then, it is possible to determine whether $C \models s\ \rho\ t$ in time $\mathcal{O}(|C|^2)$.*

Using Theorem 4.1 we can show that it is possible to compute a reduced version of a query efficiently.

PROPOSITION 4.2. *For every satisfiable conjunctive query one can compute in polynomial time an equivalent reduced conjunctive query.*

PROOF. Let $q(\bar{s}) \leftarrow R \wedge C$ be a conjunctive query. We compute the equivalence relation $\equiv_C$ and the values $gll_C(y)$ and $lul_C(y)$, for all variables y in C. This can be accomplished in linear time by Theorem 4.1. For each y, if $gll_C(y) = lul_C(y)$, then we replace y, and all the other variables y' such that $y \equiv_C y'$, with the value $gll_C(y)$. We loop over all remaining variables (in an arbitrary order) and check, for each variable z, whether there is another variable z' such that $z \equiv_C z'$. If so, we replace all such variables z' with z.

Let q' be the query resulting from the process described above. It is easy to see that q' is both equivalent to q and is reduced. $\square$

As a result of Theorem 4.1 and Proposition 4.2, we will often assume that a given query is reduced. This assumption is without loss of generality, since it is possible to check if a query is satisfiable in linear time (Theorem 4.1), and to compute an equivalent reduced version in polynomial time (Proposition 4.2).

It is often convenient to consider reduced queries, since they have certain useful properties. We discuss such a property now. A constant occurring in a query is a *relational constant* if it occurs in a relational atom. We want to show that equivalent reduced conjunctive queries have the same relational constants. For this purpose, we need the following lemma.

LEMMA 4.3 (AVOIDING A CONSTANT). *Let C be a reduced set of comparisons and d be a constant. Then there is a satisfying assignment γ of C such that $\gamma(y) \neq d$ for all variables y occurring in C.*

PROOF. First, we consider the case that C ranges over the rationals. Since C is reduced, there exists a satisfying assignment γ. If $\gamma(y) \neq d$ for all variables y occurring in C, then we are done.

Suppose therefore that $\gamma(y) = d$ for some variable y. Let Y be the set of all variables y such that $\gamma(y) = d$. We will change γ for the variables in Y. However, we cannot do so uniformly, but have to distinguish between those variables to which we may assign a smaller value and those to which we cannot do so.

Let Y^- be the subset of Y consisting of variables y such that $C \models y \leq d$, and let $Y^+ := Y \setminus Y^-$. Now, define d^- and d^+ by

$$d^- := \max(\{d' \mid d' \text{ is a constant in } C \text{ and } d' < d\} \cup \{\gamma(z) \mid \gamma(z) < d\})$$
$$d^+ := \min(\{d' \mid d' \text{ is a constant in } C \text{ and } d' > d\} \cup \{\gamma(z) \mid \gamma(z) > d\}).$$

We define a new assignment γ' by $\gamma'(y) := (d^- + d)/2$ for $y \in Y^-$, $\gamma'(y) := (d + d^+)/2$ for $y \in Y^+$, and $\gamma'(y) := \gamma(y)$ otherwise. Then γ' satisfies all comparisons between variables in C, and, since C is reduced, it satisfies as well all comparisons between variables and constants.

Next, we consider the case that C ranges over the integers. Let $\gamma^\downarrow$ be a limit assignment, as introduced in Theorem 4.1. If $\gamma^\downarrow(y) \neq d$ for all y, then we are done. Otherwise, we change $\gamma^\downarrow$ into an assignment γ' that will do the job.

If $gll_C(y) = -\infty$, then we define $\gamma'(z) := \gamma^\downarrow(z) - 1$ for all z with $\gamma^\downarrow(z) \leq \gamma^\downarrow(y)$, and $\gamma'(z) := \gamma^\downarrow(z)$ otherwise. Note that this definition implies $\gamma'(y) = d - 1$. It is easy to see that γ' satisfies C, because γ' continues to satisfy all comparisons between variables. It also satisfies all comparisons between variables and constants, because $gll_C(z) = -\infty$ for all variables z for which we have changed the values.

If $gll_C(y)$ is finite, then we define $\gamma'(z) := \gamma^\downarrow(z)$ for all z with $\gamma^\downarrow(z) < \gamma^\downarrow(y)$, and $\gamma'(z) := \gamma^\downarrow(z) + 1$ otherwise. Note that this definition implies $\gamma'(y) = d + 1$. Then γ' satisfies all comparisons between variables. It also satisfies all comparisons between variables and constants, because $gll_C(z) < lul_C(z)$ implies that $\gamma^\downarrow + 1 = gll_C(z) + 1 \leq lul_C(z)$. Hence, γ' satisfies C. $\square$

PROPOSITION 4.4 (RELATIONAL CONSTANTS AND CONTAINMENT). *Let q and q' be two reduced conjunctive queries. If q is contained in q', then every relational constant of q' is also a relational constant of q.*

PROOF. Assume that q is contained in q' and that there is a constant d' that occurs in a relational atom of q', but does not occur in a relational atom of q.

Let q be defined as $q(\bar{s}') \leftarrow R \wedge C$, and q' as $q'(\bar{s}') \leftarrow R' \wedge C'$. Obviously, since the queries are reduced, then every relational constant appearing in the head of q' must also appear in the head of q. In addition, since q is reduced, by Lemma 4.3, there is an assignment γ to the variables in q such that γ satisfies C and $\gamma(y) \neq d'$ for all variables y of q. We define a database $\mathcal{D}$ by $\mathcal{D} := \gamma R$, that is, $\mathcal{D}$ consists of the relational atoms of q instantiated by γ. Obviously, γ is an assignment over $\mathcal{D}$ that satisfies the body of q, and therefore q returns the answer $\gamma(\bar{s})$.

However, there is no assignment over $\mathcal{D}$ that satisfies the body of q'. To see this, suppose that γ' is such an assignment. Then $\gamma' R' \subseteq \gamma R$. This implies that there is an atom a of R such that γa contains the constant d'. But this is impossible, since no atom in R contains d', and γ does not introduce d'. $\square$

COROLLARY 4.5 (RELATIONAL CONSTANTS AND SET-EQUIVALENCE). *Reduced conjunctive queries that are equivalent under set semantics have the same relational constants.*

4.2. LINEAR EXPANSIONS. We now focus on ensuring that the linear expansion of a conjunctive query will be in normal form, that is, reduced. Intuitively, the problem that must be addressed is that a linearization of a query contains additional comparisons (not in the original query), which may imply equality between a variable and a constant not previously considered. Such newly introduced constants are called *virtual constants* and are considered below. We note that virtual constants are only needed for our characterization of equivalence among *sum*-queries (Section 9).

Let $q(\bar{s}) \leftarrow R \wedge C$ be a query, W be the set of variables of q, and D be a set of constants containing the constants of q. Let L be a linearization of $D \cup W$, and $q_L(\phi(\bar{s})) \leftarrow \phi R \wedge \phi L$ be a linearization of q with respect to L, where ϕ is a canonical substitution for L. Over the rationals, each such linearization is reduced.

Over the integers, however, this need not be the case. If, for instance, $q \leftarrow R$ contains the constants 0, 3, and the variables z_1, z_2, then $L = \{0 < z_1 < z_2 < 3\}$ is a linearization that is compatible with the comparisons of q (since there are none). Thus, $q_L \leftarrow R \wedge L$ is a linearization of q. However, q_L is not reduced, because every satisfying assignment maps z_1 to 1 and z_2 to 2.

If we transform the query into reduced normal form, we introduce new constants, 1 and 2 in our example. The question arises whether, right from the beginning, we can choose the constants in the linearized comparisons in such a way that the reduced normal form of a linearized query does not contain any additional constants.

In the following, we assume that all comparisons range over the integers and that all constants are integers. Let C be a set of comparisons, W be the set of variables of C, and D be a set of constants comprising the constants of C. We say that d is a *virtual constant* of C with respect to D if there is a linearization L of $D \cup W$, such that L is consistent with C and $L \models s = d$ for some term $s \in D \cup W$. We denote the set of virtual constants of C with respect to D as $vc_C(D)$.

The following lemma shows that virtual constants are located between two elements d^-, d^+ of D if the space between d^- and d^+ can consistently be filled with a strict chain of variables of C.

LEMMA 4.6 (FILLING UP SPACES). *Let C be a set of comparisons and D be a set of constants comprising the constants of C. Then d is a virtual constant of C with respect to D if and only if $d \in D$, or there are $d^-, d^+ \in D$ and $k := d^+ - d^- - 1$ variables $w_1, \ldots, w_k \in W$ such that*

—$d^- < d < d^+$, and

—$\{d^- < w_1,\ w_1 < w_2, \ldots,\ w_k < d^+\} \cup C$ is satisfiable.

PROOF

"$\Rightarrow$" Let d be a virtual constant of C with respect to D. Then there is a linearization L of $D \cup W$, such that L is consistent with C, and $L \models s = d$ for some term $s \in D \cup W$.

If s is a constant, then $d = s \in D$.

If s is a variable, say $s = y$, such that $\gamma(y) = d$ for all assignments γ satisfying L, then $d = gll_L(y) = lul_L(y)$. By the definition of gll_L and lul_L, the L implies $d \leq y$ and $y \leq d$. If $d \in D$ then we are finished. Assume otherwise.

We show that there is a constant $d^- \in D$ such that L implies $d < w_1, w_1 < w_2, \ldots, w_{j-1} < w_j, w_j < y$ for $j = d - d^- - 1$ and some variables $w_1, \ldots, w_j$. Let d^- be the greatest constant appearing in L that is less than d. There must be such a constant since y is bounded from below. Let $w_1, \ldots, w_j$ be all the variables in L such that L implies $d^- < w_i$ and $w_i < y$. (If there are several such variables that are equated by L, we only take one representative for each equivalence class of variables.) These variables are strictly ordered in L. Assume, without loss of generality, that $w_1 < w_2 < \cdots < w_j$. If $j = d - d^- - 1$, then we are finished.

The comparisons $d^- < w_1 < \cdots < w_j < y$ imply that $d^- + j + 1 \leq y$. Therefore, if $j > d - d^- - 1$, then $d < y$, in contradiction to the fact that $lul_L(y) = d$. If $j < d - d^- - 1$, then clearly $gll_L(y) < d$ in contradiction to the given. Therefore, $j = d - d^- - 1$ as required.

In a similar fashion, one can show that there is a constant $d^+ \in D$ such that the L implies $y < w_1, w_1 < w_2, \ldots, w_{j-1} < w_j, w_j < d^+$ for $j = d^+ - d - 1$ and some variables $w_1, \ldots, w_j$.

This means, there are variables $w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_k \in W$, where $i = d - d^-$, such that $\{d^- < w_1, \ldots, w_{i-1} < y,\ y < w_{i-1}, \ldots, w_k < d^+\} \subseteq L$. Since L is consistent with C, it follows that $\{d^- < w_1, \ldots, w_{i-1} < y,\ y < w_{i-1}, \ldots, w_k < d^+\} \cup C$ is satisfiable.

"$\Leftarrow$" We show that the above conditions are sufficient. Obviously, if $d \in D$, then it is a virtual constant.

Suppose that there are d^-, $d^+ \in D$ as in the statement of the lemma. Let γ be an assignment that satisfies $\{d^- < w_1 < \cdots < w_k < d^+\} \cup C$. Then, $d = \gamma(w_i)$ for some $i \in 1, \ldots, k$. Moreover, γ determines a linearization L that is obtained by first grouping terms into equivalence classes if γ maps them to the same number, and then ordering the classes according to their values under γ. All assignments satisfying L map w_i to d. Thus, d is a virtual constant of C with respect to D. $\square$

One may wonder, whether and when the process of adding virtual constants to a set D terminates. The following lemma shows that we just have to add them once, and after that no new virtual constants come into existence.

LEMMA 4.7 (ADDING VIRTUAL CONSTANTS IS A CLOSURE OPERATION). *Let C be a set of comparisons and D be a set of constants comprising the constants of C. Then we have*

(1) $D \subseteq vc_C(D)$

(2) $vc_C(D) = vc_C(vc_C(D))$.

PROOF.

(1) This has already been stated in Lemma 4.6.

(2) The inclusion "$\subseteq$" follows from Part (1) of the lemma. To show the inclusion "$\supseteq$", let $d_1 < d_2 < \cdots < d_n$ be the constants in D. Then, by Lemma 4.6, $vc_C(D)$ is obtained from D, by filling up spaces between some pairs d_i, d_{i+1} of adjacent elements of D. Similarly, $vc_C(vc_C(D))$ is obtained by filling up spaces between some pairs of adjacent elements of $vc_C(D)$.

Suppose $d \in vc_C(vc_C(D))$. Then, there are $d^-, d^+ \in vc_C(D)$ such that $d^- \leq d \leq d^+$, and it is consistent with C to completely fill the space between d^- and d^+ by a strict chain of variables.

Since $d^-, d^+ \in vc_C(D)$, there are elements d_i, d_{i+1} and $d_j, d_{j+1} \in D$, where $1 \leq i \leq j \leq n$, such that $d_i \leq d^- \leq d_{i+1}$ and $d_j \leq d^+ \leq d_{j+1}$. In addition to d^- and d^+, all elements between d_i and d_{i+1}, and all elements between d_j and d_{j+1} are elements of $vc_C(D)$, because $vc_C(D)$ is obtained by filling up spaces between pairs of adjacent elements of D.

In order to show that $d \in vc_C(D)$, we distinguish between three cases. If $d \leq d_{i+1}$, then $d_i \leq d^- \leq d \leq d_{i+1}$, and hence $d \in vc_C(D)$, since d is between d_i and d_{i+1}. If $d_j \leq d$, we conclude in a similar way that $d \in vc_C(D)$, It remains to consider the case that $d_{i+1} < d < d_j$. Since $d \in vc_C(vc_C(D))$, there is a strict chain from d^- to d^+ that has length $d^+ - d^- - 1$, that involves variables in C, and that is consistent with C (see the proof of Lemma 4.6). If this chain is shortened at both ends, it yields a strict chain from d_{i+1} to d_j of length $d_{i+1} - d_j + 1$, that involves variables in C and is consistent with C. Hence, $d \in vc_C(D)$. □

Computing the virtual constants of two sets of comparisons can be done by computing the virtual constants independently for each set.

LEMMA 4.8 (TWO SETS OF COMPARISONS). *Let C, C' be two sets of comparisons and D be a set of constants comprising the constants of C and C'. Then we have*

(1) $vc_C(vc_{C'}(D)) = vc_{C'}(vc_C(D)) = vc_C(D) \cup vc_{C'}(D)$

(2) $vc_C(vc_C(D) \cup vc_{C'}(D)) = vc_{C'}(vc_C(D) \cup vc_{C'}(D)) = vc_C(D) \cup vc_{C'}(D)$.

PROOF. By Lemma 4.6, both $vc_C(D)$ and $vc_{C'}(D)$ are obtained from D by filling up the spaces between certain pairs of adjacent elements of D. Similarly, as in the proof of Lemma 4.7, one can show that the fact that certain spaces are filled does not facilitate the filling of other spaces.

Therefore, it does not matter in which order vc_C and $vc_{C'}$ are applied. In either order, both operators fill the spaces that they would fill if they were applied alone.

Therefore, the result of their combined application is the same as the union of the results of their single applications. This yields Part (1).

For the same reason, applying either operator to $vc_C(D) \cup vc_{C'}(D)$, does not result in filling up additional spaces. This yields Part (2). □

We now arrive at the main conclusion of this section. Let $q(\bar{s}) \leftarrow R \wedge C$ be a conjunctive query. We say that the linear expansion $(q_L)_{L \in \mathcal{L}_D(q)}$ of q over D is *reduced* if every query q_L is reduced. A sufficient criterion for a linear expansion to be reduced is that it be taken over a set of constants that contains all virtual constants of the comparisons of q.

THEOREM 4.9 (REDUCED LINEAR EXPANSIONS). *Let $q(\bar{s})$ be a conjunctive query, D be a set of constants that contains the constants of q, and $(q_L)_{L \in \mathcal{L}_D(q)}$ be the linear expansion of q over D. If D contains all virtual constants of C with respect to D, that is, if $vc_C(D) = D$, then every query q_L is reduced.*

PROOF. A query $q_L(\phi_L(\bar{s})) \leftarrow \phi_L R \wedge \phi_L L$ is reduced if the set of comparisons $\phi_L L$ is reduced. The set $\phi_L L$ is reduced if for any terms s, t, the fact that $\phi_L L \models s = t$ entails that s and t are syntactically equal.

The set $\phi_L L$ does not entail the equality of syntactically distinct variables, because for any two such variables a strict inequality holds. If $\phi_L L \models s = d$, then $L \models s = d$ as well, hence, d is a virtual constant of C with respect to D. However, D is closed under vc_C, thus $d \in D$, and s and d are syntactically equal, because L is a linearization over D. □

5. *Survey of Results*

In this section, we briefly survey the results of this article. We discuss, for each aggregation function, the general technique used to determine equivalence, and the complexity bounds derived. Throughout this section, we refer to the sections and theorems in which the results are discussed in detail. Thus, this section serves as a gentle introduction and a road-map to the equivalence results presented in this article.

5.1. EQUIVALENCE CHARACTERIZATIONS.

Max-Queries. We consider equivalence of *max*-queries in Section 6. We characterize equivalence in terms of *dominance mappings* between the cores of *max*-queries. Intuitively, a dominance mapping from $q'(\bar{s}', t')$ to $q(\bar{s}, t)$ is a type of homomorphism from q' to q, which has the restriction that t must be mapped to a term at least as large as t'. For relational *max*-queries, we show that dominance is containment, i.e., there is a dominance mapping from q' to q if and only if q is contained in q' (Proposition 6.2). For *max*-queries with comparisons, our characterization of equivalence is more intricate, and requires the existence of dominance mappings for each linearization in the linear expansions of each of the queries (Theorem 6.4).

Count-Distinct-Queries. Equivalence of *cntd*-queries is considered in Section 7. Clearly, given two *cntd*-queries q and q', set-equivalence of the cores of q and q' is a sufficient condition for equivalence of *cntd*-queries, since two queries that return the same results must also return the same number of results. We show that set-equivalence of the cores is a complete characterization of equivalence for relational *cntd*-queries (Corollary 7.4) and for an additional special case (Theorem 7.3). We note that for general *cntd*-queries, which may contain arbitrary comparisons, determining equivalence is an open problem. In Section 7, we discuss further the reasons that seem to make this problem elusive.

Count-Queries. In Section 8, we characterize equivalence of *count*-queries. This is essentially the same problem as determining equivalences among conjunctive queries under bag-set semantics. We show that relational *count*-queries are equivalent if and only if they are isomorphic (Theorem 8.5). The proof is based on an analysis of counting functions that count how often a particular tuple is returned over a parameterized family of databases. We show that *count*-queries with comparisons are equivalent if and only if their linear expansions are isomorphic (Theorems 8.7 and 8.8).

Sum-Queries. We characterize equivalence among *sum*-queries in Section 9. We first consider the special case in which the *sum*-queries do not contain constants. For such queries we show that equivalence can be reduced to bag-set-equivalence of the cores (Theorem 9.5). As discussed above, bag-set-equivalence was characterized in Section 8.

It is possible for *sum*-queries with constants to be equivalent even if their cores are not bag-set-equivalent. Intuitively, this holds since sums of different constants may add up to the same value. Therefore, for this case, our characterization is much more intricate and appears in Theorems 9.8 and 9.13. Interestingly, for the special case of *sum*-queries with constants, but without comparisons, equivalence can once again be characterized in terms of bag-set-equivalence of the cores.

5.2. LINEAR AGGREGATE QUERIES. In this article, we develop a theory of equivalence for general aggregate queries. For those queries, we show that equivalence is at least as difficult as deciding graph isomorphism, a task for which no polynomial time algorithm is known, and for which it is considered highly implausible that one exists. In fact, for *count* and *sum*-queries equivalence, we show that checking equivalence amounts to checking whether two queries are isomorphic, a problem at least as difficult as graph isomorphism. Moreover, for *max*-queries, we show that equivalence is even Π_2^{P}-hard.

We briefly consider here a special class of aggregate queries for which equivalence can be checked in polynomial time. An aggregate query is *linear* if does not contain two relational atoms with the same predicate, that is, if it does not have any self-joins. In practice, linear queries occur frequently.

In Cohen et al. [2005], the notion of a *singleton-determining* aggregation functions was introduced. Formally, a singleton bag is a bag that contains a single constant. An aggregation function α is singleton-determining if, for any two singleton bags B and B' it holds that $\alpha(B) = \alpha(B')$ if and only if $B = B'$. Cohen et al. [2005] showed that linear aggregate queries with singleton-determining aggregation functions are equivalent if and only if they are isomorphic. It immediately follows that linear aggregate queries with the aggregation functions *max*, *count* and *sum* are equivalent if and only if they are isomorphic.[5]

In Theorem 7.3, we show that for a special class of *cntd*-queries, equivalence can be reduced to set-equivalence of the cores. This holds, in particular, for relational *cntd*-queries. Thus, for linear relational *cntd*-queries, (or, more generally, for linear *cntd*-queries in the class of Theorem 7.3), equivalence is isomorphism.

5.3. COMPLEXITY. In this section, we summarize the results on the complexity of aggregate queries that we obtain in this paper. We consider three cases: the general case of arbitrary aggregate queries, which may contain comparisons, and the two special cases of queries without comparisons (called *relational queries*) and of *linear queries*, which again may contain comparisons. The complexity results are shown in Table I.

TABLE I. SUMMARY OF COMPLEXITY RESULTS

Aggregation Function	Queries with Comparisons	Relational Queries	Linear Queries
max	Π_2^P-complete	NP-complete	PTIME
cntd	not known	NP-complete	PTIME (for relational queries)
count	GI-hard,[6] in PSPACE	GI-complete	PTIME
sum	GI-hard, in PSPACE	GI-complete	PTIME

For queries with the function *max*, the first two entries in the row follow from Theorem 6.6. The results for *max*-queries hold as well for *min*-queries.

For *count*-queries without comparisons, equivalence amounts to isomorphism (see Theorem 8.5). It is easy to show that the problem of query isomorphism and the graph isomorphism problem are many-one reducible to each other. Hence, query isomorphism is complete for the problem class GI, consisting of decision problems that are many-one-reducible to the graph isomorphism problem. This gives the second entry in the row for *count* and also GI-hardness for the general case. The PSPACE upper bound is shown in Theorem 8.9.

For equivalence of general *sum*-queries, PSPACE is an upper bound by Theorem 9.15. As for *count*-queries, equivalence of *sum*-queries without comparisons amounts to isomorphism (see Theorem 9.14), which gives us the second entry and the lower bound in the first entry.

All entries in the last column follow from the discussion in Section 5.2, since isomorphism of linear queries can be determined in polynomial time.

6. *Max-Queries*

In this section, we consider aggregate queries that contain an aggregate term with the function *max*. All the results for *max* can easily be translated into results for

[5] Technically, *count* is singleton-determining since the only constant in the bags to which it is applied is the empty tuple.

[6] We denote as GI the class of problems that are many-one-reducible to the graph isomorphism problem.

min. A *max-query* is a simple aggregate query of the form

$$q(\bar{x}, max(y)) \leftarrow B(\bar{w}).$$

We will reduce the equivalence of max-queries to syntactic properties of their cores

$$\breve{q}(\bar{x}, y) \leftarrow B(\bar{w}).$$

6.1. DOMINANCE OF CONJUNCTIVE QUERIES. Let $q(\bar{s}, t)$ and $q'(\bar{s}', t')$ be two conjunctive 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 $d' \geq d$. We say that q and q' *dominate each other* if q is dominated by q' and q' is dominated by q. Actually, in order to be precise, one has to define dominance with respect to an ordered domain, as we did in the case of containment and equivalence. However, to simplify the exposition, we do not mention the domain if it is clear from the context or not important.

Note that we define dominance for queries that have arbitrary terms in their head, although the aggregate queries for which we investigate containment have only head variables. The reason is that we will have to consider dominance also for queries that belong to linear expansions and therefore may have head constants.

PROPOSITION 6.1 (EQUIVALENCE AND DOMINANCE). *Two* max-*queries are equivalent if and only if their cores dominate each other.*

PROOF

"$\Rightarrow$" Suppose that $q(\bar{x}, max(y))$ and $q'(\bar{x}, max(y))$ are equivalent. It suffices to show that $\breve{q}$ is dominated by $\breve{q}'$. Consider a fixed database and suppose that $\breve{q}$ returns the tuple $(\bar{d}, d)$ over this database. Let d' be the maximal y-value in the group of $\bar{d}$ with respect to q. Then, $\breve{q}$ returns the tuple $(\bar{d}, d')$, too. The equivalence of q and q' implies that d' is also the maximal y-value in the group of $\bar{d}$ with respect to q'. Thus, $\breve{q}'$ returns $(\bar{d}, d')$ and $d \leq d'$.

"$\Leftarrow$" Suppose $\breve{q}$ and $\breve{q}'$ dominate each other. Consider a fixed database and suppose that q returns the tuple $(\bar{d}, d_{\max})$ over this database, that is, $d_{\max}$ is the maximal y-value in the group of $\bar{d}$ with respect to q. Obviously, also $\breve{q}$ returns this tuple. Since $\breve{q}$ is dominated by $\breve{q}'$, there is a tuple $(\bar{d}, d')$ with $d' \geq d_{\max}$ that is returned by $\breve{q}'$. Thus, d' is in the group of $\bar{d}$ with respect to q'. Let $d'_{\max}$ be the maximal element in that group. Then $\breve{q}'$ as well as q' return the tuple $(\bar{d}, d'_{\max})$. Moreover, $d'_{\max} \geq d' \geq d_{\max}$, hence $d'_{\max} \geq d_{\max}$. Reversing the roles of q and q', we can show that $d_{\max} \geq d'_{\max}$ and thus $d_{\max} = d'_{\max}$. Hence, also q' returns the tuple $(\bar{d}, d_{\max})$.

This shows that q is contained in q'. By a symmetric argument, it follows that also q' is contained in q, and thus q and q' are equivalent. $\square$

Obviously, if a conjunctive query q is contained in another conjunctive query q', then q is dominated by q'. If the queries are the cores of relational *max*-queries, then also the converse holds.

PROPOSITION 6.2 (DOMINANCE AND CONTAINMENT). *Let $q(\bar{x}, y)$ and $q'(\bar{x}, y)$ be relational conjunctive queries. Then, q is dominated by q' if and only if q is contained in q'.*

PROOF. It suffices to show that dominance implies containment. Suppose that q is dominated by q'. We will construct a database $\mathcal{D}$ from q, and then exploit the fact that q is dominated by q' over $\mathcal{D}$ to construct a containment mapping from q' to q.

Suppose q and q' are defined by $q(\bar{x}, y) \leftarrow R(\bar{w})$ and $q'(\bar{x}, y) \leftarrow R'(\bar{w}')$. Let δ be an injective mapping from $\bar{w}$, the variables in the body of q, to constants in an ordered domain such that $\delta(y)$ is greater than any constant occurring in the body of q and such that $\delta(y) \geq \delta(w)$ for all $w \in \bar{w}$.

Now, define $\mathcal{D}$ as

$$\mathcal{D} := \{\delta r(\bar{u}) \mid r(\bar{u}) \text{ is an atom in } R(\bar{w})\}.$$

In other words, each relation $r^{\mathcal{D}}$ consists of the tuples $\delta(\bar{u})$ for all conjuncts $r(\bar{u})$ in the body of q. Note that $\delta(y)$ is the maximal constant occurring in $\mathcal{D}$.

Clearly, δ is an assignment that satisfies q over $\mathcal{D}$, hence, q returns the tuple $(\delta(\bar{x}), \delta(y))$ over $\mathcal{D}$. Since q is dominated by q', there is an assignment δ' that satisfies q' over $\mathcal{D}$ such that $\delta(\bar{x}) = \delta'(\bar{x})$ and $\delta'(y) \geq \delta(y)$. The constant $\delta(y)$ is maximal in $|\mathcal{D}|$, which implies that $\delta(y) = \delta'(y)$.

Next, we define a substitution θ for the variables in q' by $\theta z := \delta^{-1}\delta'(z)$, which is well-defined because δ is injective. Obviously, $\theta\bar{x} = \bar{x}$ and $\theta y = y$. Moreover, for every relational atom $r(\bar{z})$ in the body of q', we have $\theta r(\bar{z}) = \delta^{-1}\delta' r(\bar{z}) \in R(\bar{w})$, since $\delta' r(\bar{z}) \in \mathcal{D}$ and $\mathcal{D}$ is defined as $\delta R(\bar{w})$. Hence, θ is a containment mapping. □

Note that the above proposition only holds because no constants can occur in the heads of the queries. It is easy to check, however, that arbitrary relational queries are equivalent if they dominate each other.

It is not true that dominance implies containment if queries may contain comparisons.

Example 6.3. Consider the queries

$$\begin{aligned} q(y) &\leftarrow p(y) \wedge p(z_1) \wedge p(z_2) \wedge z_1 < z_2 \\ q'(y) &\leftarrow p(y) \wedge p(z) \wedge z < y. \end{aligned}$$

Both queries return answers if there are at least two elements in p. If this is the case, then q returns all elements of p, while q' returns all elements but the least. Thus, the two queries dominate each other. However, they are not set-equivalent, since q contains q', but q' does not contain q.

6.2. Deciding Dominance.

Since for queries with comparisons, dominance is more general than containment, we have to come up for dominance checking with a more general technique than finding homomorphisms. We therefore generalize homomorphisms to dominance mappings.

Let $q(\bar{s}, t) \leftarrow R \wedge C$ and $q'(\bar{s}', t') \leftarrow R' \wedge C'$ be conjunctive queries with comparisons, ranging over the domain $\mathcal{I}$. A *dominance mapping* from q' to q is a substitution θ of the variables in q' by terms in q such that

(1) $\theta\bar{s}' = \bar{s}$;
(2) $\theta R' \subseteq R$;
(3) $C \models_{\mathcal{I}} \theta C'$;
(4) $C \models_{\mathcal{I}} \theta t' \geq t$.

THEOREM 6.4 (DOMINANCE WITH COMPARISONS). *Let q and q' be two conjunctive queries with comparisons, and let $(q_L)_L$ be a linearization of q. Then, q*

is dominated by q' if and only if for every q_L in $(q_L)_L$ there exists a dominance mapping from q' to q_L.

PROOF. Suppose that q and q' are queries over $\mathcal{I}$, defined as $q(\bar{s}, t) \leftarrow R \wedge C$ and $q'(\bar{s}', t') \leftarrow R' \wedge C'$, respectively.

"$\Rightarrow$" Suppose that q is dominated by q', and let $q_L(\phi\bar{s}, \phi t) \leftarrow \phi R \wedge \phi L$ be a linearization of q. We show that there exists a dominance mapping from q' to q_L.

To do so, we construct a database $\mathcal{D}$ from q_L. Since L is satisfiable over $\mathcal{I}$, so is ϕL. Let δ be an assignment for q_L that satisfies ϕL. Since q_L is a linearization of q, the assignment δ is injective, that is $\delta(z_1) \neq \delta(z_2)$ for any two distinct variables z_1, z_2 in q_L, and hence the inverse mapping δ^{-1} is well defined. Moreover, we have that $\delta(z_1) < \delta(z_2)$ if and only if $\phi L \models_{\mathcal{I}} z_1 < z_2$.

Now, define $\mathcal{D}$ as

$$\mathcal{D} := \{\delta\phi r(\bar{u}) \mid r(\bar{u}) \text{ is a relational atom of } q\}.$$

In other words, each relation $r^{\mathcal{D}}$ consists of the tuples $\delta\phi(\bar{u})$ for all conjuncts $r(\bar{u})$ in the body of q.

By construction of $\mathcal{D}$, the mapping $\delta\phi$ is an assignment that satisfies the body of q, and thus q returns the tuple $(\delta\phi(\bar{s}), \delta\phi(t))$ over $\mathcal{D}$. Since q' dominates q, there is an assignment δ' over $\mathcal{D}$ that satisfies the body of q' such that

$$\delta'(\bar{s}') = \delta\phi(\bar{s}) \quad \text{and} \quad \delta'(t') \geq \delta\phi(t). \tag{6}$$

We define a substitution θ for the variables w' in q' by

$$\theta w' := \begin{cases} \delta^{-1}\delta'(w') & \text{if } \delta'(w') \text{ is a constant introduced by } \delta \\ \delta'(w') & \text{if } \delta'(w') \text{ is a constant in the body of } q_L. \end{cases}$$

The definition makes sense because, first, every constant in $\mathcal{D}$ is by construction the image $\delta(v)$ of a term v in q_L and, second, all the values of δ' appear in $\mathcal{D}$, since q' is safe. We show that θ is a dominance mapping.

(1) We have $\theta\bar{s}' = \phi\bar{s}$ because of Eq. (6).

(2) Let $r(\bar{v})$ be a relational atom of q'. Then, $\delta' r(\bar{v}) \in \mathcal{D}$, and therefore $\theta r(\bar{v}) \in \phi R$ by definition of θ.

(3) Let $v_1 \leq v_2$ be a comparison of q'. Since δ' satisfies the comparisons in q', we have $\delta'(v_1) \leq \delta'(v_2)$. If $\delta'(v_1) = \delta'(v_2)$, then $\theta v_1 = \theta v_2$, and $\phi L \models_{\mathcal{I}} \theta v_1 \leq \theta v_2$ holds trivially. If $\delta'(v_1) < \delta'(v_2)$, then $\delta\theta v_1 < \delta\theta v_2$ by definition of θ, and hence $\phi L \models_{\mathcal{I}} \theta v_1 \leq \theta v_2$. With similar arguments, we can show that $\phi L \models_{\mathcal{I}} \theta v_1 \, \rho \, \theta v_2$ for other comparison $v_1 \, \rho \, v_2$ of q'. Thus, we conclude that $\phi L \models_{\mathcal{I}} \theta C'$

(4) Because of Eq. (6) and by definition of θ we have $\delta\theta t' = \delta'(t') \geq \delta\phi(t)$, which yields $\phi L \models_{\mathcal{I}} \theta t' \geq \phi(t)$.

Thus, θ fulfills all the conditions of a dominance mapping.

"$\Leftarrow$" Suppose that for every linearization q_L of $(q_L)_L$ there exists a dominance mapping from q' to q_L.

Let $\mathcal{D}$ be a database and γ be an assignment that satisfies the body of q. To prove dominance of q by q', we show that there is an assignment γ' that satisfies the body of q' such that $\gamma'(\bar{s}') = \gamma(\bar{s})$ and $\gamma'(t') \geq \gamma(t)$.

The assignment γ induces a linearization L on the terms of q where $u_1\ \rho\ u_2 \in L$ if and only if $\gamma(u_1)\ \rho\ \gamma(u_2)$ holds. Since γ satisfies C, it follows that L is compatible with C. Let $q_L(\phi s, \phi t) \leftarrow \phi R \wedge \phi L$ be the linearized version of q in $(q_L)_L$ that corresponds to L. There is a unique assignment γ_L for q_L such that $\gamma = \gamma_L\phi$.

Let θ be a dominance mapping from q' to q_L. Then, $\gamma_L\theta$ is an assignment that satisfies each relational atom $r(\bar{v})$ in the body of q', which can be seen as follows: $\theta r(\bar{v}) = \phi r(\bar{u})$ for some atom $r(\bar{u})$ in R, hence $\gamma_L\theta r(\bar{v}) = \gamma_L\phi r(\bar{u}) = \gamma r(\bar{u}) \in \mathcal{D}$.

Moreover, $\gamma_L\theta$ satisfies the comparisons C' of q': we have $L \models_{\mathcal{I}} \theta C'$ and γ_L satisfies ϕL, which implies that γ_L satisfies L, hence γ_L satisfies $\theta C'$, and therefore $\gamma_L\theta$ satisfies C'. In addition, $\gamma_L\theta\bar{s}' = \gamma_L\phi(\bar{s}) = \gamma(\bar{s})$, since $\theta\bar{s}' = \phi(\bar{s})$. Finally, since $\phi_L \models_{\mathcal{I}} \theta t' \geq \phi(t)$, we have $\gamma_L\theta t' \geq \gamma_L\phi(t) = \gamma(t)$. Thus, $\gamma' := \gamma\theta$ is the desired assignment. □

Next, we are interested in the complexity of determining dominance. The statement of the theorem holds for queries over the rationals as well as over the integers.

THEOREM 6.5 (COMPLEXITY OF DOMINANCE).

(1) *Dominance of conjunctive queries is* Π_2^{P}*-complete.*

(2) *Dominance of relational conjunctive queries is* NP*-complete.*

PROOF

(1) We first show that non-dominance is in Σ_2^{P}. We note that the existence of a dominance mapping from one query to another can be decided in nondeterministic polynomial time. Let q, q' be conjunctive queries, possibly with comparisons. In order to prove that q is not dominated by q', we guess a linearized version q_L of q and call an NP-oracle to verify that there is *no* dominance mapping from q' to q_L. This shows that dominance is in Π_2^{P}.

We show Π_2^{P}-hardness by a reduction of the containment problem for arbitrary conjunctive queries (which may contain comparisons), which has been proven in van der Meyden [1992] to be Π_2^{P}-complete.

Let $q_0(\bar{s}) \leftarrow B(\bar{w})$ and $q_0'(\bar{s}') \leftarrow B'(\bar{w}')$ be two conjunctive queries, possibly with comparisons. Let r be a unary predicate neither occurring in q_0 nor in q_0'. Define the queries q and q' by $q(\bar{s}, y) \leftarrow B(\bar{w}) \wedge r(y)$ and $q'(\bar{s}', y) \leftarrow B'(\bar{w}') \wedge r(y)$. Note that y is not involved in any comparison in q and q'.

Let L be a linearization of the terms in q, and let $q_L(\phi\bar{s}, \phi y) \leftarrow B(\phi\bar{w}) \wedge r(\phi y)$ be the corresponding linearized version of q. By construction of q and q', a dominance mapping from q' to q_L has to map y to ϕy.

Now, for every dominance mapping θ from q' to q_L, the restriction of θ to the variables of q_0' is a containment mapping from q_0' to q_{0M}, where M is the restriction of L to the terms of q_0. Since every linearization of the terms of q_0 is a restriction of a linearization of the terms of q, this shows that dominance of q by q' implies containment of q_0 in q_0'.

To see the converse, let again M be the restriction of L to the terms of q_0. Any containment mapping from q_0' to q_{0M} can be extended to a dominance mapping from q' to q_L by mapping y to ϕy, since y does not occur in any comparison in q'. Since L was arbitrary, this shows that containment of q_0 in q_0' implies dominance of q by q'.

Thus, we have reduced containment to dominance of conjunctive queries, which proves the Π_2^{P}-completeness of the latter.

(2) The second claim is obvious, since by Proposition 6.2, dominance is equivalent to containment for relational conjunctive queries. □

We can also state precisely the complexity of equivalence, which turns out to be the same as that of dominance. Again, the theorem below holds for both queries over the rationals and over the integers.

THEOREM 6.6 (COMPLEXITY OF EQUIVALENCE).

(1) *Equivalence of max-queries is* Π_2^{P}*-complete.*

(2) *Equivalence of relational max-queries is* NP*-complete.*

PROOF. We first prove the upper bounds. To show that two *max*-queries, say q and q', are not equivalent it suffices to show that the core of one, say $\breve{q}$, is not dominated by the core of the other, that is $\breve{q}'$. This is the case, by Theorem 6.4, if there exists a linearization $\breve{q}_L$ of $\breve{q}$ such that there is no dominance mapping from $\breve{q}'$ to $\breve{q}_L$. Thus, existence of a dominance mapping can be decided by an NP-oracle. Since a linearization $\breve{q}_L$ can be guessed in polynomial time, nonequivalence of *max*-queries is in Σ_2^{P} and equivalence is in Π_2^{P}.

For any two *max*-queries q, q', equivalence amounts to mutual dominance of the cores by Proposition 6.1, which by Proposition 6.2 amounts to mutual containment if the two queries are relational. Hence, two relational *max*-queries are equivalent if and only if their cores are equivalent. Since equivalence of relational conjunctive queries is known to be NP-complete, this proves that equivalence is in NP for relational *max*-queries.

We prove the lower bounds by reducing equivalence of conjunctive queries to equivalence of *max*-queries. The reduction is similar to the one in the proof of the preceding theorem. Let $q_0(\bar{x}) \leftarrow B(\bar{w})$ and $q_0'(\bar{x}) \leftarrow B'(\bar{w}')$ be two conjunctive queries, r be a unary predicate neither occurring in q_0 nor in q_0', and let y be a new variable. We define two *max*-queries q and q' by $q(\bar{x}, max(y)) \leftarrow B(\bar{w}) \wedge r(y)$ and $q'(\bar{x}, max(y)) \leftarrow B'(\bar{w}') \wedge r(y)$.

We want to prove that q_0 and q_0' are equivalent if and only if q and q' are equivalent. Let $\mathcal{D}$ be a database. Since the variable y does not occur in B, the query q returns a tuple $(\bar{d}, d)$ over $\mathcal{D}$ if and only if $\bar{d} \in q_0^{\mathcal{D}}$ and d is the maximal element in $r^{\mathcal{D}}$. An analogous statement holds for q'. This proves that the *max*-queries q and q' are equivalent if and only if the conjunctive queries q_0 and q_0' are equivalent. Note that q, q' are relational if q_0 and q_0' are relational.

Now, the upper bound for the general case follows from the fact that equivalence of conjunctive queries with comparisons is Π_2^{P}-complete [van der Meyden 1992] while for the case of relational queries, equivalence is NP-complete. □

7. *Count-Distinct-Queries*

In this section, we show that for queries with the aggregation function *cntd*, equivalence of the cores under set-semantics is a sufficient condition for equivalence and we give criteria for when it is also a necessary condition.

A *count-distinct-query* (also written *cntd*-query) is a simple aggregate query of the form

$$q(\bar{x}, cntd(y)) \leftarrow B(\bar{w}).$$

Its core is the query

$$\breve{q}(\bar{x}, y) \leftarrow B(\bar{w}).$$

If the cores of two *cntd*-queries are equivalent under set-semantics, then they return the same values for corresponding groups. Thus, in particular, they return they same number of distinct values. This gives us a sufficient condition for the equivalence of *cntd*-queries.

PROPOSITION 7.1 (SUFFICIENCY OF SET-EQUIVALENCE). *Two cntd-queries are equivalent if their cores are equivalent under set-semantics.*

The converse of Proposition 7.1 holds for relational queries (see Corollary 7.4), but not in the general case. In the remainder of this section, we identify a situation in which the converse of Proposition 7.1 holds even for queries with comparisons. First, we note in passing that there is no known complete characterization for equivalence of *cntd*-queries with comparisons. For the other types of queries considered in this article (*max*-queries, *count*-queries, *sum*-queries), we completely characterize equivalence by considering separately each linearization of each of the queries. Unfortunately, this approach cannot be applied to determine equivalence of *cntd*-queries since the result of a *cntd*-query cannot be evaluated from the results of each of its linearizations. It seems that this makes the problem of characterizing equivalence of *cntd*-queries more elusive.

The following example demonstrates that equivalence under set-semantics is not a necessary condition for queries that contain comparisons.

Example 7.2. Consider the queries

$$\begin{aligned} q(\mathit{cntd}(y)) &\leftarrow p(y) \wedge p(z) \wedge y < z \\ q'(\mathit{cntd}(y')) &\leftarrow p(y') \wedge p(z') \wedge z' < y'. \end{aligned}$$

Both queries give a result if there are at least two elements in p. The core of q returns all elements of p but the greatest, while the core of q' returns all but the least. Thus, both cores return the same number of elements, but they are not set-equivalent.

We present some definitions necessary to identify cases where set-equivalence of the cores is a necessary condition. Let q and q' be queries with comparisons C and C', respectively. Let w and w' be variables appearing in q and q', respectively. We say that the variable w' is a *possible preimage* of w if

—there are relational atoms $p(s_1, \ldots, s_n)$ and $p(s'_1, \ldots, s'_n)$ in the bodies of q and q', respectively (i.e., relational atoms with the same predicate) *and*
—there is an index $j \leq n$

such that $C \models w = s_j$ and $C' \models w' = s'_j$. In other words, w and w' are equal to terms in the same positions in relational atoms with the same predicates. As a special case, we say that w' in q' is a direct possible preimage of w in q if w and w' are distinguished variables and appear in the same places, respectively, in the heads of q and q'.

In Example 7.2, observe that the variables y' and z' in q' are both possible preimages of the aggregation variable y in q. Observe also that y' and z' are involved in a comparison. In the following theorem, we show that when such cases are ruled out (i.e., possible preimages of the aggregation variables are not involved in comparisons), then query equivalence can be reduced to set equivalence of the query cores.

THEOREM 7.3 (REDUCTION TO SET-SEMANTICS). *Suppose that $q(\bar{x}, cntd(y))$ and $q'(\bar{x}, cntd(y'))$ are cntd-queries. Suppose also that there is no variable in q' that is both a possible preimage of y in q and involved in any comparison. Similarly, suppose that there is no variable in q that is both a possible preimage of y' in q' and involved in any comparison. Then q and q' are equivalent if and only if their cores are set-equivalent.*

PROOF. By Proposition 7.1, q and q' are equivalent if their cores are set-equivalent. Assume therefore, that the cores $\breve{q}$ and $\breve{q}'$ are not equivalent under set semantics. Without loss of generality, assume that $\breve{q}$ is not contained in $\breve{q}'$. We will show that q is not equivalent to q'.

Let D be the set of constants appearing in q or q'. By Theorem 2.3, there is a linearization $\breve{q}_L$ in the linear expansion of $\breve{q}$ over D such that there is no homomorphism from $\breve{q}'$ to $\breve{q}_L$. We construct two databases $\mathcal{D}$ and $\mathcal{D}'$ from $\breve{q}_L$ such that for some tuple $\bar{d}$, the query q' counts the same number of distinct values in the group of $\bar{d}$ over both databases, while q does not.

Let W be the set of variables occurring in q. Suppose that the variables in q range over the domain $\mathcal{I}$ and that D are constants in $\mathcal{I}$. Since L is satisfiable with respect to $\mathcal{I}$ and q, there is an embedding δ from $W \cup D$ into $\mathcal{I}$. We define $\mathcal{D}$ as

$$\mathcal{D} := \{\delta r(\bar{t}) \mid r(\bar{t}) \text{ is a relational atom of } q\},$$

that is, each relation $r^{\mathcal{D}}$ consist of the tuples $\delta(\bar{t})$ for all conjuncts $r(\bar{t})$ in the body of q.

Let $\bar{d} := \delta(\bar{x})$ and $d := \delta(y)$. By construction of $\mathcal{D}$, the mapping δ, if restricted to W, is an assignment that satisfies the body of q, and thus d is an element of the group of $\bar{d}$ with respect to q. However, d is not an element of the group of $\bar{d}$ with respect to q'. Otherwise, there would be an assignment γ satisfying the body of q' such that $\gamma(\bar{x}, y) = (\bar{d}, d)$. From such an assignment one could construct a homomorphism θ from $\breve{q}'$ to $\breve{q}_L$ by choosing for every non-distinguished variable z of $\breve{q}'$ a term s in $\breve{q}_L$ with $\delta(s) = \gamma(z)$ and by defining $\theta(z) := s$.

We now construct the database $\mathcal{D}'$ by adding some atoms to $\mathcal{D}$. Let d' be an arbitrary number not appearing in D or $\mathcal{D}$. Let δ' be identical with δ, except that $\delta'(y) = d'$. Then, for every atom a in q, we add the atom $\delta' a$ to $\mathcal{D}$, thus obtaining $\mathcal{D}'$. (New atoms are actually added only if a contains y. Otherwise, $\delta' a = \delta a$ and is already in $\mathcal{D}$.) Formally,

$$\mathcal{D}' := \mathcal{D} \cup \{\delta' r(\bar{t}) \mid r(\bar{t}) \text{ is a relational atom of } q\}.$$

By construction, δ' is an assignment that satisfies the relational atoms of q. Clearly, δ' also satisfies the comparisons of q, since it differs from δ only on the value for y, and y does not participate in any comparisons (by the assumption of the theorem). Thus, d' is an element of the group of $\bar{d}$ with respect to q over $\mathcal{D}'$, in addition to those that were already in the group of $\bar{d}$ over $\mathcal{D}$.

We show that there are no new elements in the group of $\bar{d}$ over $\mathcal{D}'$ with respect to the query q'. To see this, assume that γ' is an assignment over $\mathcal{D}'$ with $\gamma'(\bar{x}) = \bar{d}$ that satisfies the body of q'. Let γ be obtained from γ' by defining $\gamma(z) := d$, whenever $\gamma'(z) = d'$, and $\gamma(z) := \gamma'(z)$ otherwise. Then γ satisfies every relational atom in q'.

We show that γ also satisfies all comparisons in q'. Let $s_1 \ \rho \ s_2$ be a comparison in q'. By the assumptions of the theorem, no possible preimage of y is involved in a comparison in q'. Therefore, it must be that $\gamma'(s_1) \neq d'$ and $\gamma'(s_2) \neq d'$. Hence, $\gamma(s_1) = \gamma'(s_1) \ \rho \ \gamma'(s_2) = \gamma(s_2)$ as required.

As a consequence, if $\gamma'(y) = d'$, then $\gamma(y) = d$, and γ satisfies the body of q'. Hence, d is in the group of $\bar{d}$ with respect to q' over $\mathcal{D}$, which contradicts our initial assumption. If $\gamma'(y) = e \neq d'$, then we also have $\gamma(y) = e$, and e is already in the group of $\bar{d}$ over $\mathcal{D}$.

That q returns different counts for the group of $\bar{d}$ over $\mathcal{D}$ and $\mathcal{D}'$, while q' does not, contradicts the fact that q and q' are equivalent. Thus, the assumption the $\breve{q}$ and $\breve{q}'$ are not equivalent under set-semantics is wrong. □

Observe that the conditions of Theorem 7.3 imply that y and y' are not equal to constants (since they are not involved in any comparisons). It is easy to show that if y is equal to a constant, and y' is not, then $q(\bar{x}, \mathit{cntd}(y))$ and $q'(\bar{x}, \mathit{cntd}(y))$ are not equivalent. Similarly, it is easy to show that if both y and y' are equal to (possibly different) constants, then q and q' are equivalent if and only if the conjunctive queries $p(\bar{x})$ and $p(\bar{x})$ derived by stripping off $\mathit{cntd}(y)$ and $\mathit{cntd}(y')$, are equivalent under set semantics.

We can simplify the conditions in Theorem 7.3 if the queries are relational.

COROLLARY 7.4 (RELATIONAL CNTD-QUERIES). *Suppose that $q(\bar{x}, \mathit{cntd}(y))$ and $q'(\bar{x}, \mathit{cntd}(y'))$ are relational cntd-queries. Then, q and q' are equivalent if and only if their cores are equivalent under set-semantics.*

We immediately derive the following complexity bound for relational *cntd*-queries.

COROLLARY 7.5 (COMPLEXITY OF EQUIVALENCE). *The problem of determining equivalence of relational cntd-queries is* NP-*complete.*

8. *Count-Queries and Bag-Set-Equivalence*

In this section, we consider aggregate queries that contain aggregate terms with the function *count*. We show that the equivalence problem for *count*-queries can be rephrased as the problem to decide bag-set-equivalence.

For relational queries, we prove that two such queries are bag-set-equivalent if and only if they are isomorphic. This result has already been stated before by Chaudhuri and Vardi [1993], but they did not supply a proof. For queries with comparisons, we prove that they are bag-set-equivalent if and only if they have isomorphic linear expansions.

8.1. COUNT-QUERIES.

A *count-query* is a simple aggregate query of the form

$$q(\bar{x}, \mathit{count}) \leftarrow B(\bar{w}).$$

We recall that the core of such a *count*-query is the query

$$\breve{q}(\bar{x}) \leftarrow B(\bar{w}).$$

According to the semantics of *count*-queries, the query $q(\bar{x}, \mathit{count})$ returns a tuple $(\bar{d}, d)$ if and only if $d \geq 1$ and there are d assignments γ with $\gamma(\bar{x}) = \bar{d}$ that satisfy the body of q. This yields an immediate characterization of the equivalence of *count*-queries.

PROPOSITION 8.1. *Two count-queries are equivalent if and only if their cores are bag-set-equivalent.*

It is easy to see that isomorphism of conjunctive queries is a sufficient condition for bag-set-equivalence.

PROPOSITION 8.2 (ISOMORPHISM IMPLIES BAG-SET-EQUIVALENCE). *Isomorphic conjunctive queries are bag-set-equivalent.*

PROOF. Let θ be an isomorphism from q' to q. Then for all assignments γ and γ' we have that γ satisfies the body of q if and only if $\gamma\theta$ satisfies the body of q', and γ' satisfies the body of q' if and only if $\gamma'\theta^{-1}$ satisfies the body of q. This establishes a one-to-one correspondence between the answers to q and the answers to q'. $\square$

8.2. BAG-SET-EQUIVALENCE OF RELATIONAL QUERIES.

We now prove the converse of Proposition 8.2 for relational conjunctive queries. The result will also follow from Theorem 8.8 in the next subsection, which characterizes arbitrary conjunctive queries. However, for the relational case, we can apply a different proof technique that is interesting by itself.

Without loss of generality, we can assume that bag-set equivalent queries have heads that have the form $q(\bar{x})$, $q'(\bar{x})$: if queries are bag-set equivalent, then they are equivalent, and hence their heads are isomorphic.

Let $q(\bar{x}) \leftarrow R$ and $q'(\bar{x}) \leftarrow R'$ be relational conjunctive queries. To show that bag-set-equivalence of q and q' implies isomorphism, we prove that bag-set-equivalence implies the existence of a surjective homomorphism from q' to q. Then, by symmetry, there is also a surjective homomorphism from q to q'. From this, we can conclude isomorphism, also for arbitrary conjunctive queries.

LEMMA 8.3. *Let q and q' be relational conjunctive queries such that there are surjective homomorphisms θ from q' to q and θ' from q to q'. Then, q and q' are isomorphic.*

We will now show that if $q(\bar{x})$, $q'(\bar{x})$ are relational conjunctive queries such that there is no surjective homomorphism from q' to q, then there is a database where they return some tuple with different multiplicities. To this end we construct from the query q a family of databases $(\mathcal{D}_{\bar{N}})_{\bar{N}\in\mathbf{N}^l}$, where l is the number of existential variables in q, and study how many times each of the queries returns a certain tuple $\bar{d}$ over these databases. More precisely, we consider the functions Γ and Γ' that count how often $\bar{d}$ is returned over $\mathcal{D}_{\bar{N}}$ by q and q', respectively, and show that they have the following properties:

(1) Γ and Γ' are polynomials over $\mathbf{N}^l$;
(2) Γ contains a monomial $c_{1,\dots,1}N_1^1 \cdots N_l^1$;
(3) Γ' contains only monomials $c_{e_1,\dots,e_l}N_1^{e_1} \cdots N_l^{e_l}$, where $e_i = 0$ for some exponent e_i.

This implies that Γ and Γ' are different functions and therefore q and q' return $\bar{d}$ with different multiplicities over some database $\mathcal{D}_{\bar{N}}$.

The idea behind this proof is as follows. The databases $\mathcal{D}_{\bar{N}}$ will be constructed in such a way that each assignment over $\mathcal{D}_{\bar{N}}$ that satisfies q or q' corresponds to a unique homomorphism from q to q, or from q' to q, respectively. Each homomorphism contributes to the entire counting function a sum of monomials that counts the assignments corresponding to that homomorphism. The monomials for a homomorphism θ contain a variable N_i if and only if the i-th existential variable y_i of q occurs in the range of θ. Then, Property (2) reflects the fact that there is a surjective homomorphism from q to q, namely the identity, and Property (3) reflects the fact that there is no surjective homomorphism from q' to q.

We now construct the databases $\mathcal{D}_{\bar{N}}$. Let D_q be the set of constants occurring in q, and let $D = \{d_1, \dots, d_k\}$ be a set with as many constants as there are output variables. Let $\bar{d}$ denote the tuple $(d_1, \dots, d_k)$. For every tuple of natural numbers $\bar{N} \in \mathbf{N}^l$, $\bar{N} = (N_1, \dots, N_l)$, let

$$D_{\bar{N}} = \left\{ d_i^{(j)} \mid i \in 1, \dots, l,\ j \in 1, \dots, N_i \right\}$$

be a set consisting of $N_1 + \cdots + N_l$ distinct constants. Intuitively, there are N_i copies $d_i^{(j)}$ for every existential variable y_i of q. We assume that D_q, D, and $D_{\bar{N}}$ are mutually disjoint. The set $D_q \cup D \cup D_{\bar{N}}$ will be the carrier of $\mathcal{D}_{\bar{N}}$. We say that an assignment $\gamma\colon (\bar{x}, \bar{y}) \to D_q \cup D \cup D_{\bar{N}}$ is *nice* if $\gamma(\bar{x}) = \bar{d}$ and $\gamma(y_i) = d_i^{(j)}$ for some $j \in 1, \dots, N_i$. Now, $\mathcal{D}_{\bar{N}}$ consists of all images of the body of q under nice assignments, i.e.,

$$\mathcal{D}_{\bar{N}} := \{\gamma a \mid \gamma \text{ is a nice assignment, and } a \text{ is an atom in the body of } q\}.$$

Intuitively, $\mathcal{D}_{\bar{N}}$ has been constructed by "blowing up" the query q.

We now introduce a *collapsing function* π from the constants in the database $\mathcal{D}_{\bar{N}}$ to the terms in q that maps each constant in the database back to its corresponding variable or constant in the query. The function π is defined by

$$\begin{aligned} \pi(d) &:= d && \text{for } c \in C \\ \pi(d_i) &:= x_i && \text{for } i \in 1, \dots, k \\ \pi(d_i^{(j)}) &:= y_i && \text{for } i \in 1, \dots, k \text{ and } j \in 1, \dots, N_i. \end{aligned}$$

Due to the construction of $\mathcal{D}_{\bar{N}}$, the collapsing function behaves like a surjective homomorphism. In fact, for every predicate p of arity m and every m-tuple $\bar{d}'$ of constants in $\mathcal{D}_{\bar{N}}$, we have that $\pi(p(\bar{d}'))$ is an atom of the body of q if and only if $p(\bar{d}')$ is an atom in $\mathcal{D}_{\bar{N}}$. As a consequence, for every assignment $\gamma\colon (\bar{x}, \bar{y}) \to |\mathcal{D}_{\bar{N}}|$ with $\gamma(\bar{x}) = \bar{d}$, if γ satisfies q, then $\pi\gamma$ is a homomorphism from q to q. Similarly, for assignments $\gamma'\colon (\bar{x}, \bar{z}) \to |\mathcal{D}_{\bar{N}}|$ with $\gamma'(\bar{x}) = \bar{d}$, if γ' satisfies q', then $\pi\gamma'$ is a homomorphism from q' to q.

Next, we want to count how many times $\bar{d}$ is returned over $\mathcal{D}_{\bar{N}}$ by q and q', respectively. To this end we define the counting functions Γ and Γ' as

$$\begin{aligned} \Gamma(\bar{N}) &:= |\{\gamma\colon (\bar{x}, \bar{y}) \to |\mathcal{D}_{\bar{N}}| \,|\, \gamma(\bar{x}) = \bar{d}, \text{ and } \gamma \text{ satisfies } q\}| \\ \Gamma'(\bar{N}) &:= |\{\gamma'\colon (\bar{x}, \bar{z}) \to |\mathcal{D}_{\bar{N}}| \,|\, \gamma'(\bar{x}) = \bar{d}, \text{ and } \gamma' \text{ satisfies } q'\}|, \end{aligned}$$

where $\bar{y} = (y_1, \dots, y_l)$ consists of the existential variables of q and $\bar{z} = (z_1, \dots, z_l)$ consists of the existential variables of q'.

LEMMA 8.4 (PROPERTIES OF COUNTING FUNCTIONS). *Let $q(\bar{x})$ and $q'(\bar{x})$ be two relational conjunctive queries, and let Γ and Γ' be the corresponding counting functions over $\mathcal{D}_{\bar{N}}$. Then:*

(1) *Γ and Γ' are polynomials over $\mathbf{N}^l$, where l is the number of existential variables in q;*
(2) *Γ contains a monomial $c_{1,\ldots,1}N_1^1 \cdots N_l^1$;*
(3) *if there is no surjective homomorphism from q' to q, then Γ' contains only monomials $c_{e_1,\ldots,e_n}N_1^{e_1} \cdots N_l^{e_l}$, where $e_i = 0$ for some exponent e_i.*

PROOF.
(1) We show the first claim for Γ. For Γ' the proof is analogous. For every homomorphism θ from q to q, let

$$\Gamma_\theta(\bar{N}) := |\{\gamma\colon (\bar{x}, \bar{y}) \to |\mathcal{D}_{\bar{N}}| \mid \gamma(\bar{x}) = \bar{d}, \gamma \text{ satisfies } q, \text{and } \pi\gamma = \theta\}|,$$

that is, Γ_θ counts the satisfying assignments γ that are collapsed by π to θ. Obviously,

$$\Gamma(\bar{N}) = \sum_{\theta \in Hom(q,q)} \Gamma_\theta(\bar{N}).$$

If we show that each Γ_θ is a polynomial, then it follows that Γ is a polynomial.

We classify the assignments γ with $\pi\gamma = \theta$ according to which variables y_i are mapped to the same constant in $\mathcal{D}_{\bar{N}}$. Let V be the set of existential variables of q. Each γ induces a partition $\mathcal{P}_\gamma$ of V that groups variables into the same class if γ maps them to the same constant in the database $\mathcal{D}_{\bar{N}}$. We say that the partition $\mathcal{P}_\gamma$ is the *pattern of* γ. We also say that a partition $\mathcal{P}$ is a *pattern* if $\mathcal{P} = \mathcal{P}_\gamma$ for some satisfying assignment γ. For every homomorphism θ and pattern $\mathcal{P}$ we define

$$\Gamma_{\theta,\mathcal{P}}(\bar{N}) := |\{\gamma\colon (\bar{x}, \bar{y}) \to |\mathcal{D}_{\bar{N}}| \mid \gamma(\bar{x}) = \bar{d}, \gamma \text{ satisfies } q, \pi\gamma = \theta, \text{and } \mathcal{P}_\gamma = \mathcal{P}\}|,$$

that is, $\Gamma_{\theta,\mathcal{P}}$ counts the satisfying assignments γ that are collapsed by π to θ and have the pattern $\mathcal{P}$. Obviously,

$$\Gamma_\theta(\bar{N}) = \sum_{\mathcal{P}} \Gamma_{\theta,\mathcal{P}}(\bar{N}).$$

If we show that each $\Gamma_{\theta,\mathcal{P}}$ is a polynomial, then it follows that Γ_θ and hence Γ is a polynomial.

Now, we count how many ways there are to construct an assignment γ with pattern $\mathcal{P}$ such that $\pi\gamma = \theta$. There is no choice in mapping the output variables, since $\pi\gamma(\bar{x}) = \theta(\bar{x}) = \bar{x}$ enforces that $\gamma(\bar{x}) = \bar{d}$. Similarly, if θ maps an existential variable y to an output variable x_i or to a constant c, then there is no other choice for γ than to map y to d_i or to c, respectively.

However, there is a choice if γ maps an existential variable to another existential variable, say y_j. For every $y_j \in V$, let $V_j^\theta := \theta^{-1}(y_j)$ be the set of variables that θ maps to y_j. Evidently, only existential variables are mapped to y_j. The set V_j^θ is the disjoint union of m_j classes of $\mathcal{P}$, that is,

$$V_j^\theta = P_{j,1} \uplus \cdots \uplus P_{j,m_j},$$

where two variables are in the same class $P_{j,h}$ if γ maps them to the same constant in the database $\mathcal{D}_{\bar{N}}$. The above fragment of the pattern $\mathcal{P}$ can be realized in

$N_j(N_j - 1)\cdots(N_j - m_j + 1)$ ways by an assignment: each variable in V_j^θ has to be mapped to some constant $d_i^{(j)}$, where $i \in 1, \ldots, N_j$. However, variables in the same class $P_{j,h}$ are mapped to the same $d_i^{(j)}$. Thus, m_j distinct constants out of N_j must be chosen. Since for every $j \in 1, \ldots, l$, there are N_j constants $d_i^{(j)}$ in $\mathcal{D}_{\bar{N}}$, we have

$$\Gamma_{\theta,\mathcal{P}}(\bar{N}) = \prod_{j=1}^{l} N_j(N_j - 1)\cdots(N_j - m_j + 1). \tag{7}$$

This shows that $\Gamma_{\theta,\mathcal{P}}$ is a polynomial in $\bar{N}$. Hence Γ is a polynomial. An analogous argument shows that Γ' is a polynomial. This proves Claim (1).

(2) To verify Claim (2), observe that the identity *id* is a homomorphism from q to q. Since *id* is injective, assignments corresponding to *id* can only have the pattern $\mathcal{P}_{id} = \{\{y_1\}, \ldots, \{y_l\}\}$. Thus,

$$\Gamma_{id}(\bar{N}) = \Gamma_{id,\mathcal{P}_{id}}(\bar{N}) = N_1^1 \cdots N_l^1. \tag{8}$$

Equation (7) implies that the degree of every polynomial $\Gamma_{\theta,\mathcal{P}}$ is at most l, and that the coefficient of a monomial of degree l in $\Gamma_{\theta,\mathcal{P}}$ is always positive. Thus, Eq. (8) implies that there is a monomial $c_{1,\ldots,1}N_1^1 \cdots N_l^1$ in Γ.

(3) To see Claim (3), note that, by Eq. (7), a variable N_j appears in $\Gamma_{\theta,\mathcal{P}}$ if and only if $\theta(z) = y_j$ for some existential variable z of q'. By assumption, there is no surjective homomorphism from q' to q. Hence, for every homomorphism θ from q' to q, there is some y_j such that $\theta(z) \neq y_j$ for all existential variables of q'. Hence, in each $\Gamma_{\theta,\mathcal{P}}$, one variable N_j is missing. As a consequence, all monomials in Γ' are of the form $c_{e_1,\ldots,e_n}N_1^{e_1} \cdots N_l^{e_l}$, where $e_i = 0$ for some exponent e_i. □

The preceding lemma implies that there exist surjective homomorphisms between bag-set-equivalent relational conjunctive queries. Otherwise, there would be a tuple of numbers $\bar{N}$ such that the counting functions Γ and Γ' differ for $\bar{N}$, that is, $\Gamma(\bar{N}) \neq \Gamma'(\bar{N})$. This would mean that over the database $\mathcal{D}_{\bar{N}}$, the queries q and q' return the tuple $\bar{d}$ with different multiplicities.

Thus, together with Lemma 8.3, we can conclude the main result of this subsection.

THEOREM 8.5 (BAG-SET-EQUIVALENCE IS ISOMORPHISM). *Relational conjunctive queries are bag-set-equivalent if and only if they are isomorphic.*

8.3. BAG-SET-EQUIVALENCE OF QUERIES WITH COMPARISONS.

If queries have comparisons, then it is not true that bag-set-equivalence entails isomorphism.

Example 8.6. Consider the queries

$$\begin{aligned} q &\leftarrow p(x) \wedge p(y) \wedge p(z) \wedge x < y \wedge x < z \\ q' &\leftarrow p(x) \wedge p(y) \wedge p(z) \wedge x < z \wedge y < z \end{aligned}$$

Clearly, they are not isomorphic. However, they are bag-set-equivalent, which can be seen as follows.

Suppose, $d_1 < d_2$ is a pair of two distinct constants in the relation $p^{\mathcal{D}}$ in some database $\mathcal{D}$. Such a pair gives rise to the assignment $\gamma_1 = \{x \mapsto d_1,\ y \mapsto d_2,\ z \mapsto d_2\}$, which satisfies q. If $d_1 < d_2 < d_3$ is a triple of distinct constants in $p^{\mathcal{D}}$, it gives rise to two assignments satisfying q, namely $\gamma_2 = \{x \mapsto d_1,\ y \mapsto d_2,\ z \mapsto d_3\}$, and $\gamma_3 = \{x \mapsto d_1,\ y \mapsto d_3,\ z \mapsto d_2\}$. If $p^{\mathcal{D}}$ contains n elements, then there are $\binom{n}{2}$ pairs and $\binom{n}{3}$ triples of distinct elements. Hence, q returns $\binom{n}{2} + 2\binom{n}{3}$ answers over $\mathcal{D}$. With similar arguments, it can be seen that q' returns the same number of answers over $\mathcal{D}$.

The comparisons in q and q' contain only incomplete information about the relationships between x, y, and z. They can each be completed in three ways to linearizations of $\{x, y, z\}$ compatible with those comparisons. In the case of q, those three linearizations are

$$x < y = z, \qquad x < y < z, \qquad x < z < y,$$

and in the case of q', they are

$$x = y < z, \qquad x < y < z, \qquad y < x < z.$$

Based on the linearizations for q, we can construct a linear expansion over the empty set of constants, consisting of the three queries

$$\begin{aligned} q_1 &\leftarrow p(x) \wedge p(y) \wedge x < y \\ q_2 &\leftarrow p(x) \wedge p(y) \wedge p(z) \wedge x < y \wedge y < z \\ q_3 &\leftarrow p(x) \wedge p(y) \wedge p(z) \wedge x < z \wedge z < y. \end{aligned}$$

Similarly, for q' we can construct the linear expansion

$$\begin{aligned} q'_1 &\leftarrow p(x) \wedge p(z) \wedge x < z \\ q'_2 &\leftarrow p(x) \wedge p(y) \wedge p(z) \wedge x < y \wedge y < z \\ q'_3 &\leftarrow p(x) \wedge p(y) \wedge p(z) \wedge y < x \wedge x < z. \end{aligned}$$

Although the queries q and q' are not isomorphic, one readily checks that the linearizations q_i and q'_i are pairwise isomorphic.

In the sequel of this subsection, we will show that isomorphism of linear expansions completely characterizes bag-set-equivalence. It is not too difficult to show that isomorphism of the linear expansions of two queries is a sufficient criterion for bag-set-equivalence.

THEOREM 8.7 (ISOMORPHISM IMPLIES BAG-SET-EQUIVALENCE). *Let $q(\bar{x})$ and $q'(\bar{x})$ be two conjunctive queries. If q and q' have isomorphic linear expansions, then they are bag-set-equivalent.*

PROOF. Let $(q_L)_L$ and $(q'_M)_M$ be isomorphic linear expansions of q and q'. Then, there is a bijection $\mu\colon \mathcal{L}_D(q) \to \mathcal{L}_D(q')$ such that for every linearization $L \in \mathcal{L}_D(q)$ there is an isomorphism θ_L from $q'_{\mu(L)}$ to q_L.

Let $\mathcal{D}$ be a database. We show that there is a bijection between the assignments over $\mathcal{D}$ that satisfy q and those that satisfy q'. Moreover, if under this bijection γ' corresponds to γ, then $\gamma'(\bar{x}) = \gamma(\bar{x})$. This implies that both queries return the same answers with the same multiplicities.

Suppose that γ satisfies q. Then γ satisfies exactly one linearization $L \in \mathcal{L}_D(q)$, and γ also satisfies the query q_L. Let $M := \mu(L)$, let θ_L be the isomorphism from q'_M to q_L, and let ϕ_M be the canonical substitution for M and q' that produced q'_M. Since θ_L and ϕ_M are homomorphisms, the assignment $\gamma' := \gamma\theta_L\phi_M$ satisfies q'.

The association of γ' to γ can be inverted. We show that $\gamma = \gamma'\theta_L^{-1}\phi_L$. Observe that ϕ_M is the identity for the variables of q'_M. Hence, $\phi_M\theta_L^{-1} = \theta_L^{-1}$. Now, using the definition of γ', we obtain $\gamma'\theta_L^{-1}\phi_L = \gamma\theta_L\phi_M\theta_L^{-1}\phi_L = \gamma\theta_L\theta_L^{-1}\phi_L = \gamma\phi_L = \gamma$, where the last identity holds because γ satisfies L. This shows that we have indeed defined a bijection between the assignments satisfying q and those satisfying q'.

We now check that γ and γ' produce the same answers for q and q'. Let $\bar{s} := \phi_M(\bar{x})$ be the distinguished terms of q'_M, and $\bar{t} := \phi_L(\bar{x})$ be those of q_L, where ϕ_M and ϕ_L are the canonical substitutions that produced q'_M and q_L. Since γ satisfies L, we have $\gamma\phi_L(\bar{x}) = \gamma(\bar{x})$. Combining these identities, we obtain $\gamma'(\bar{x}) = \gamma\theta_L\phi_M(\bar{x}) = \gamma\theta_L(\bar{s}) = \gamma(\bar{t}) = \gamma\phi_L(\bar{x}) = \gamma(\bar{x})$, which yields the claim. □

For the converse of the preceding theorem, we have to be careful with the set D over which we construct the linear expansion: it must comprise the constants in both queries.

THEOREM 8.8 (BAG-SET-EQUIVALENCE IMPLIES ISOMORPHISM). *Let $q(\bar{x})$ and $q'(\bar{x})$ be conjunctive queries and let D be the set of constants occurring in q or q'. If q and q' are bag-set-equivalent, then their linear expansions over D are isomorphic.*

PROOF. Let $(q_L)_L$ and $(q'_M)_M$ be linear expansions of q and q' over D. We have to show that there is a bijection $\mu: \mathcal{L}_D(q) \to \mathcal{L}_D(q')$ such that q_L and $q'_{\mu(L)}$ are isomorphic for all $L \in \mathcal{L}_D(q)$.

Let $\mathcal{Q}$ be the set of all queries occurring in $(q_L)_L$ and $\mathcal{Q}'$ be the set of those occurring in $(q'_M)_M$. Isomorphism of queries is an equivalence relation on $\mathcal{Q}$ and on $\mathcal{Q}'$. Let $Q \subseteq \mathcal{Q}$ and $Q' \subseteq \mathcal{Q}'$ be equivalence classes of isomorphic queries. We say that Q and Q' are *partners* if there is a $q_L \in Q$ and a $q'_M \in Q'$ such that q_L and q'_M are isomorphic. In this case, all elements of Q are isomorphic to all elements of Q'. If Q or Q' does not have a partner, we say that the empty set is its partner. Let

$$\mathcal{P} := \{(Q, Q') \mid Q \text{ and } Q' \text{ are partners}\}$$

be the collection of all pairs of classes of isomorphic queries that are partners of each other.

The theorem follows if we can show that for each pair $(Q, Q') \in \mathcal{P}$, we have $|Q| = |Q'|$, that is, two partners have the same cardinality. Because then, we can define μ locally for each pair (Q, Q') by mapping their elements bijectively to each other in an arbitrary manner.

Assume that there is a pair in $\mathcal{P}$ whose components have different cardinalities. We call such pair a *counter-example*. We show that under this assumption, q and q' are not bag-set-equivalent.

We first choose all counter-examples such that the number of variables in their queries is minimal. Among those, we choose all counter-examples such that the number of relational atoms in their queries is minimal. Let (Q, Q') be such a counter-example. Without loss of generality, we can assume that $|Q| > |Q'|$.

Let $q_L(\bar{s}) \leftarrow R_L \wedge L$ be an element of Q and let δ be an assignment that satisfies L. We construct a database $\mathcal{D}$ out of q_L by defining $\mathcal{D} := \delta R_L$, that is, $\mathcal{D}$ consists of the ground atoms obtained by instantiating the atoms of R_L by δ.

The query q_L returns the answer $\bar{d} := \delta(\bar{s})$ over $\mathcal{D}$. We show that it returns it exactly once. The comparisons in L force any satisfying assignment to map the variables to values distinct from the constants in D. We call the constants in $\mathcal{D}$ that are not in D *fresh* constants. The only fresh constants are the values introduced by δ. Moreover, L enforces a strict linear order on the assigned values, which excludes any assignment other than δ. Similarly, all queries that are isomorphic to q_L return $\bar{d}$ exactly once.

For $\mathcal{D}$, as for any database, we have

$$q^{\mathcal{D}} = \bigcup_{q_L \in Q} q_L^{\mathcal{D}} \qquad \text{and} \qquad q'^{\mathcal{D}} = \bigcup_{q'_M \in Q'} q'^{\mathcal{D}}_M.$$

The queries in Q contribute the answer $\bar{d}$ exactly $|Q|$ times to the overall result $q^{\mathcal{D}}$, and those in Q' exactly $|Q'|$ times. Since $|Q| > |Q'|$ and q and q' are bag-set-equivalent, this difference has to be compensated. Hence, there is a pair $(P, P') \in \mathcal{P}$ such that the elements of P and P' return the answer $\bar{d}$, and $|P| < |P'|$. The pair (P, P') is a counter-example.

Let $q'_M(\bar{t}) \in P'$ and let γ be an assignment over $\mathcal{D}$ that satisfies q'_M and returns $\bar{d}$. Our goal is to show that q'_M and q_L are isomorphic by showing that $\delta^{-1}\gamma$ is an isomorphism from q'_M to q_L. This will imply that $(P, P') = (Q, Q')$, contradicting the fact that $|P| < |P'|$.

We observe that the comparisons in M force γ to map

(1) variables to fresh constants in $\mathcal{D}$, and

(2) distinct variables to distinct constants.

From Observation (1) and (2), we conclude that $\delta^{-1}\gamma$ maps variables to variables and is injective, because δ maps variables to fresh constants of $\mathcal{D}$ and is injective.

From Observation (1) and (2), we conclude also that q'_M has no more variables than there are fresh constants in $\mathcal{D}$. Since the fresh constants have been introduced by δ, we conclude that q'_M has no more variables than q_L. Hence, since (Q, Q') is a counter-example with a minimal number of variables, q'_M and q_L have the same number of variables. This implies that $\delta^{-1}\gamma$ is a bijection between the variables of q'_M and q_L.

Since γ returns $\bar{d}$, we have $\gamma(\bar{t}) = \bar{d} = \delta(\bar{s})$, and therefore, $\delta^{-1}\gamma(\bar{t}) = \bar{s}$. Hence, $\delta^{-1}\gamma$ maps distinguished terms of q'_M to their counterparts in q_L. In addition, γ maps the relational atoms of q'_M to the ground atoms in $\mathcal{D}$, which are bijective images under δ of the relational atoms of q_L. Thus, $\delta^{-1}\gamma$ is a relational homomorphism.

Since L is a linearization over D and δ satisfies L, the values $\delta(y)$ are related to each other and to the constants in D in exactly the same way as the variables y themselves. Since q'_M has as many variables as there are fresh constants in $\mathcal{D}$, the same holds for M and γ. Thus, we have $L \models \delta^{-1}\gamma M$. This yields that $\delta^{-1}\gamma$ is a homomorphism from q'_M to q_L.

It remains to prove that $\gamma^{-1}\delta$, the inverse of $\delta^{-1}\gamma$, is also a homomorphism. In a similar way as above, we can show that $\delta^{-1}\gamma(\bar{s}) = \bar{t}$ and that $M \models \gamma^{-1}\delta L$.

Now, the proof is complete if we show that $\gamma^{-1}\delta$ maps each relational atom of q_L to one of q'_M. To do so, it suffices to show that each relational atom of q_L is the image of a relational atom of q'_M under $\delta^{-1}\gamma$. This will follow, if we show that q'_M and q_L have the same number of relational atoms.

From Observation (2), we conclude, that γ maps distinct relational atoms in q'_M to distinct relational atoms in $\mathcal{D}$. This implies that q'_M has no more relational atoms than $\mathcal{D}$, and therefore no more than q_L. However, since (Q, Q') is a counter-example with a minimal number of relational atoms, q'_M and q_L have the same number of relational atoms. This yields the claim. $\square$

Our characterization of bag-set-equivalence gives us immediately an upper complexity bound.

THEOREM 8.9 (UPPER COMPLEXITY BOUND). *Bag-set-equivalence of conjunctive queries with comparisons can be decided with polynomial space.*

PROOF. Let q, q' be two conjunctive queries with comparisons. Let $(q_L)_L$, $(q'_M)_M$ be the linear expansions of q and q', respectively, $\mathcal{Q}$ be the set of all queries occurring in $(q_L)_L$ and $\mathcal{Q}'$ be the set of those occurring in $(q'_M)_M$.

The two linear expansions are isomorphic if and only if, for every $q_L \in \mathcal{Q}$, there are as many isomorphic queries in $\mathcal{Q}$ as there are in $\mathcal{Q}'$, and, similarly, for every $q'_M \in \mathcal{Q}'$, there are as many isomorphic queries in $\mathcal{Q}'$ as there are in $\mathcal{Q}$.

Each of the two conditions can be checked with polynomial space as follows. In an outer loop, we enumerate all elements of $\mathcal{Q}$. During the enumeration, for each $q_L \in \mathcal{Q}$, we enumerate in an inner loop all elements of $\mathcal{Q}$ and count how many are isomorphic to q_L. Then, in a subsequent inner loop, we enumerate all elements of $\mathcal{Q}'$, count those that are isomorphic to q_L, and check that there are at least as many as there are isomorphic ones in $\mathcal{Q}$. In a second outer loop, we check the analogous condition for elements $q'_M \in \mathcal{Q}'$.

At each stage of the computation, there are at most two nested loops, each of which needs no more than polynomial space. Thus, the entire algorithm can be executed with polynomial space. $\square$

9. *Sum-Queries*

In this section, we consider aggregate queries that contain an aggregate term with the function *sum*. A *sum-query* is a simple aggregate query of the form

$$q(\bar{x}, sum(y)) \leftarrow B(\bar{w}).$$

We recall that the core of the above query is the conjunctive query

$$\breve{q}(\bar{x}, y) \leftarrow B(\bar{w}).$$

We will give a complete characterization of when two *sum*-queries are equivalent. We first consider the case of queries without constants. We give a simple proof showing that such queries are equivalent if and only if their cores are bag-set-equivalent. For queries with constants, this is not true. We will characterize equivalence in the general case in Section 9.2.

9.1. SUM-QUERIES WITHOUT CONSTANTS.

The main result of this subsection is that equivalence of *sum*-queries without constants can be reduced to bag-set-equivalence of their cores. One half of the reduction is straightforward, and holds, in fact, for arbitrary *sum*-queries.

PROPOSITION 9.1 (BAG-SET-EQUIVALENCE IMPLIES SUM-EQUIVALENCE). *Two sum-queries are equivalent if their cores are equivalent under bag-set-semantics.*

PROOF. Consider two *sum*-queries and a fixed database. If their cores are equivalent under bag-set-semantics, the cores return over the database the same tuples $(\bar{d}, d)$ with the same multiplicity. In particular, for every $\bar{d}$, the groups of y-values for every core contain the same numbers with the same multiplicity. Hence, the sums over the two groups are the same and thus the *sum*-queries return the same result for $\bar{d}$. $\square$

However, from the fact that for two queries the sums over corresponding groups of y-values are the same over each database, we cannot always deduce that the y-values are the same and occur with the same multiplicities.

Example 9.2. Consider the two queries

$$\begin{aligned} q(sum(y)) &\leftarrow p(1) \wedge p(2) \wedge p(3) \wedge \\ &\quad\;\; p(y) \wedge 1 \leq y \leq 3 \\ q'(sum(y)) &\leftarrow p(1) \wedge p(2) \wedge p(3) \wedge \\ &\quad\;\; p(y) \wedge 1 \leq y \leq 2 \wedge \\ &\quad\;\; p(z) \wedge 1 \leq z \leq 2. \end{aligned}$$

where all variables range over the integers. Then, both queries return a result if and only if the database contains the atoms $p(1)$, $p(2)$, and $p(3)$. Moreover, both queries return the number 6, but the first query obtains it as $6 = 1+2+3$, while the second obtains it as $6 = 1 + 2 + 1 + 2$. Thus, the two *sum*-queries are equivalent, but their cores are not—neither under set- nor under bag-set-semantics.

To enforce in the first query that exactly the numbers 1, 2, and 3 are returned, we exploit the fact that there are no other integers y with $1 \leq y \leq 3$. In a similar vein, we enforce that the second query outputs exactly the numbers 1 and 2, and that each of them is output exactly twice.

Over databases that contain rational numbers, which have a dense ordering, the two queries are not equivalent. For example, over the database

$$\{p(1),\ p(1.5),\ p(2),\ p(3)\},$$

the first query returns $q(7.5)$, while the second returns $q'(13.5)$.

The technique to prove our result consists in transforming queries and databases by strictly monotonic mappings. A mapping $\phi : \mathcal{I} \to \mathcal{I}$ on an ordered domain $\mathcal{I}$ is *strictly monotonic* if $\phi(u) < \phi(v)$ for all $u, v \in \mathcal{I}$ with $u < v$. If a is an atom containing variables and constants from $\mathcal{I}$, then ϕa is the atom where each constant u is replaced with $\phi(u)$. Similarly, for a query q, database $\mathcal{D}$, and a multiset of atoms A, we define ϕq, $\phi \mathcal{D}$, and ϕA as the query, the database, and the multiset, respectively, that are obtained by replacing each occurrence of an atom a with ϕa.

PROPOSITION 9.3. *Let q be a conjunctive query and $\mathcal{D}$ be a database over the ordered domain $\mathcal{I}$. Let ϕ be a strictly monotonic mapping on $\mathcal{I}$. Then*

$$\{\!\{ \phi q \}\!\}^{\phi \mathcal{D}} = \phi(\{\!\{ q \}\!\}^{\mathcal{D}}).$$

PROOF. Since ϕ is injective, every assignment over $\phi\mathcal{D}$ can be written as $\phi\gamma$ for some assignment γ over $\mathcal{D}$. Therefore, the proposition follows if we show that an assignment γ over $\mathcal{D}$ satisfies the body of q if and only if $\phi\gamma$ satisfies the body of ϕq.

The mapping ϕ is injective. Hence, for every relational atom a, we have $\gamma a \in \mathcal{D}$ if and only if $\phi\gamma a \in \phi\mathcal{D}$. The mapping ϕ is strictly monotonic. Hence, γ satisfies a comparison $s\ \rho\ t$ if and only if $\phi\gamma$ satisfies $\phi\gamma(s)\ \rho\ \phi\gamma(t)$.

Thus, γ satisfies an atom a in the body of q if and only if $\phi\gamma$ satisfies ϕa in the body of ϕq. $\square$

COROLLARY 9.4. *Let q be a conjunctive query without constants and $\mathcal{D}$ be a database over an ordered domain $\mathcal{I}$. Let ϕ be a strictly monotonic mapping on $\mathcal{I}$. Then*

$$\{\!\!\{\, q \,\}\!\!\}^{\phi\mathcal{D}} = \phi(\{\!\!\{\, q \,\}\!\!\}^{\mathcal{D}}).$$

PROOF. This is an immediate consequence of Proposition 9.3, since $\phi q = q$, if q does not contain constants. $\square$

THEOREM 9.5 (EQUIVALENCE OF SUM-QUERIES W/O CONSTANTS). *Let q, q' be sum-queries without constants. If q and q' are equivalent, then their cores $\breve{q}$ and $\breve{q}'$ are bag-set-equivalent.*

PROOF. Assume that q and q' are equivalent. Let $\mathcal{D}$ be a database over the ordered domain $\mathcal{I}$. We have to show that

$$\{\!\!\{\, \breve{q} \,\}\!\!\}^{\mathcal{D}} = \{\!\!\{\, \breve{q}' \,\}\!\!\}^{\mathcal{D}}. \tag{9}$$

This claim is difficult to show for an arbitrary database, because the same sum of y-values may be produced in different ways, as can be seen in Example 9.2. We therefore transform $\mathcal{D}$ into another database, where the multiplicities of each y-value in a group with respect to q and q' can be read off the sum over the group.

First, we determine an upper bound for the multiplicities of y-values, i.e., for the number of times that a tuple $(\bar{d}, d)$ can be produced by the queries $\breve{q}$ and $\breve{q}'$. Such an upper bound is the number of assignments satisfying the bodies of $\breve{q}$ and $\breve{q}'$. Let l be the maximum of the numbers of variables appearing in the bodies of $\breve{q}$ and $\breve{q}'$, and let n be the number of constants appearing in $\mathcal{D}$. Then, n^l is an upper bound for the number of assignments over $\mathcal{D}$ that satisfy the bodies of $\breve{q}$ or $\breve{q}'$.

Let $u_1 < u_2 < \cdots < u_n$ be the constants appearing in $\mathcal{D}$, which are all either integers or rational numbers. Let $M > n^l$, and let ϕ be the strictly monotonic mapping on $\mathcal{I}$ defined by $\phi(u_i) := M^i$.

To prove Eq. (9), it suffices to show that

$$\{\!\!\{\, \breve{q} \,\}\!\!\}^{\phi\mathcal{D}} = \{\!\!\{\, \breve{q}' \,\}\!\!\}^{\phi\mathcal{D}},$$

since by Corollary 9.4 this implies that $\phi(\{\!\!\{\, \breve{q} \,\}\!\!\}^{\mathcal{D}}) = \phi(\{\!\!\{\, \breve{q}' \,\}\!\!\}^{\mathcal{D}})$, which implies that $\{\!\!\{\, \breve{q} \,\}\!\!\}^{\mathcal{D}} = \{\!\!\{\, \breve{q}' \,\}\!\!\}^{\mathcal{D}}$, because ϕ is a bijection between the carriers of $\mathcal{D}$ and $\phi\mathcal{D}$.

Since the *sum*-queries q and q' are equivalent, they have corresponding groups over $\phi\mathcal{D}$, that is, if $\breve{q}$ returns $(\bar{d}, d)$, then $\breve{q}'$ returns $(\bar{d}, d')$ for some d', and vice versa.

Now, consider a fixed $\bar{d}$. Let $m_1, \ldots, m_n$ be the multiplicities of the y-values $M^1, \ldots, M^n$ in the group of $\bar{d}$ with respect to q, and let $m'_1, \ldots, m'_n$ be their multiplicities in the group of $\bar{d}$ with respect to q'. Since q and q' are equivalent, the sums over the multisets of y-values in the groups of q and q' are the same, that is,

$$\sum_{i=1}^{n} m_i M^i = \sum_{i=1}^{n} m'_i M^i. \tag{10}$$

As over $\mathcal{D}$, the number of assignments over $\phi\mathcal{D}$ satisfying the bodies of q and q' is at most $n^l < M$. Thus, $m_i < M$ and $m'_i < M$ for all i. Hence, the sums in Eq. (10) are M-adic representations of the same number. The coefficients in such a representation are always uniquely determined, so that $m_i = m'_i$ for all i. Hence, over $\phi\mathcal{D}$, each y-value has the same multiplicity in the groups of $\bar{d}$ with respect to q and q'.

Since $\bar{d}$ was chosen arbitrarily, this shows that $\{\!\{ \breve{q} \}\!\}^{\phi\mathcal{D}} = \{\!\{ \breve{q}' \}\!\}^{\phi\mathcal{D}}$, which implies our claim. □

As the proof shows, the preceding theorem relies upon the fact that for queries without constants renaming a database by a monotonic mapping results in the query returning a renamed set of answer tuples. This property still holds if instead of all databases we consider a class of databases restricted by functional dependency or referential integrity constraints. Consequently, the characterization is also true in these cases and for other integrity constraints with the same property.

9.2. SUM-QUERIES WITH CONSTANTS.

We will give a characterization of the equivalence of *sum*-queries that resembles the one for *count*-queries.

Intuitively, this is not surprising, since we can reduce the equivalence problem for *count*-queries to the one for *sum*-queries. Obviously, the queries $q(\bar{x}, count) \leftarrow B$ and $q'(\bar{x}, count) \leftarrow B'$ are equivalent, if and only if the queries $q_0(\bar{x}, sum(y)) \leftarrow B \wedge y = 1$ and $q'_0(\bar{x}, sum(y)) \leftarrow B' \wedge y = 1$ are equivalent. The count of the multiplicity of an answer $\bar{d}$ in the first queries is returned as the sum of the 1's in the last argument of the latter queries. Note, however, that the conditions in q_0 and q'_0 are not safe according to our definition in Subsection 2.1. Therefore, it will not be possible to transfer the results for *sum*-queries to *count*-queries in a straightforward manner.

Determining equivalence is more complicated for *sum*-queries than for *count*-queries. The group of y-values for a tuple $\bar{d}$ may contain several values. As can be seen from Example 9.2, in two queries, the groups for a given $\bar{d}$ may differ, but still result in the same sum. As we will see, this phenomenon can only occur if the groups contain constants that explicitly appear in the body of the query or, if the group ranges over the integers, there are variables that are constrained by the comparisons to some constants.

As for our characterization of bag-set-equivalence in Section 8.3, we consider linear expansions of queries. In the present case, however, we have to make an additional effort to control the constants that are explicitly or implicitly present in the query. We do so by making sure that

—each query in the linear expansion is reduced, and

—all constants that occur in the reduced queries appear already in the underlying linearization of comparisons.

As discussed in Section 4.2, this is achieved by computing the linear expansions with respect to the virtual constants of the queries.

9.2.1. *Equivalence of Sum-Queries with Constants.* We want to check whether two *sum*-queries of the form $q(\bar{x}, sum(y)) \leftarrow R \wedge C$ and $q'(\bar{x}, sum(y)) \leftarrow R' \wedge C'$, possibly containing constants, are equivalent. To this end, we consider reduced linear expansions $(\breve{q}_L)_{L \in \mathcal{L}_D(\breve{q})}$ and $(\breve{q}'_M)_{M \in \mathcal{L}_D(\breve{q}')}$ of the cores $\breve{q}$ and $\breve{q}'$ of q and q'.

We distinguish between those linearizations whose summation terms are constants and those whose summation terms are variables. We say that a query $\breve{q}_L(\bar{s}, s)$ in $(\breve{q}_L)_L$ is a *variable query* if the summation term s is a variable, and we say that it is a *constant query* if s is a constant.

Example 9.6. Consider again the queries q, q' introduced in Example 9.2. The query $\breve{q}$ has three linearizations

$$\breve{q}(d_i) \leftarrow p(1) \wedge p(2) \wedge p(3), \quad i = 1, 2, 3,$$

where $d_i = i$. The query $\breve{q}'$ has four linearizations

$$\breve{q}'(d'_{ij}) \leftarrow p(1) \wedge p(2) \wedge p(3), \quad i = 1, 2, \ j = 1, 2,$$

where $d'_{ij} = i$.

All linearizations are constant queries. If one of them returns a result over a database, then the others do so as well. However, the queries are not isomorphic, because they differ in the summation term.

We capture this relationship with the term of weak isomorphism. A substitution θ is a *weak homomorphism* from a query $p(\bar{s}, s) \leftarrow B$ to a query $p'(\bar{t}, t) \leftarrow B'$, if θ is a homomorphism from $\bar{p}(\bar{s}) \leftarrow B$ to $\bar{p}'(\bar{t}) \leftarrow B'$. Analogously, we define weak isomorphisms and weak isomorphism of queries. We write $p \sim p'$ if p and p' are weakly isomorphic. Intuitively, a weak homomorphism from $\breve{q}_L(\bar{s}, s)$ to $\breve{q}'_M(\bar{t}, t)$ is a homomorphism that does not pay attention to the summation terms s and t. We say that p and p' are *weakly set-equivalent* if $\bar{p}$ and $\bar{p}'$ are equivalent under set-semantics.

Isomorphism is an equivalence relation on the queries in $(\breve{q}_L)_L$ and in $(\breve{q}'_M)_M$. We denote the class of queries in $(\breve{q}_L)_L$ that are isomorphic to $\breve{q}_L$ as $[\breve{q}_L]$. Similarly, $[\breve{q}'_M]$ denotes the class of queries in $(\breve{q}'_M)_M$ that are isomorphic to $\breve{q}'_M$. Let $\mathcal{Q}$ be the set of all classes $[\breve{q}_L]$, and $\mathcal{Q}'$ be the set of all classes $[\breve{q}'_M]$. We say that $[\breve{q}_L]$ is weakly isomorphic to $[\breve{q}_{L'}]$, if $\breve{q}_L$ and $\breve{q}_{L'}$ are weakly isomorphic. We write in this case $[\breve{q}_L] \sim [\breve{q}_{L'}]$. We partition $\mathcal{Q}$ into the sets $\mathcal{Q}_V$ and $\mathcal{Q}_C$ that consist of the classes of variable queries and constant queries, respectively. There is a similar partition of $\mathcal{Q}'$.

Two classes of constant queries $Q_0 \in \mathcal{Q}_C$ and $Q'_0 \in \mathcal{Q}'_C$ are *associated* if they are weakly isomorphic. If for a class in $\mathcal{Q}_C$ or in $\mathcal{Q}'_C$ there is no associated class, then we say it is associated to the empty class. If $\breve{q}_L$ is a constant query, then we denote the summation constant as $\sigma(\breve{q}_L)$. We say that $(\breve{q}_L)_L$ and $(\breve{q}'_M)_M$ are *in balance* if for every pair Q_0, Q'_0 of associated classes of constant queries we have

$$\sum_{\substack{Q \in \mathcal{Q}_C \\ Q \sim Q_0}} \sum_{\breve{q}_L \in Q} \sigma(\breve{q}_L) = \sum_{\substack{Q' \in \mathcal{Q}'_C \\ Q' \sim Q'_0}} \sum_{\breve{q}'_M \in Q} \sigma(\breve{q}'_M).$$

The definition can be rephrased as follows: For every constant query $\breve{q}_L$, collect all weakly isomorphic constant queries in $(\breve{q}_L)_L$ and sum up their summation constants. Do the same for all constant queries in $(\breve{q}'_M)_M$ that are weakly isomorphic to $\breve{q}_L$. The resulting sums must be the same.

Example 9.7. The cores of the queries q, q' introduced in Example 9.2 have linear expansions that are in balance. All their linearizations are given in Example 9.6. They are all constant queries and weakly isomorphic to each other. The sum for $\breve{q}$ is 6, and the sum for $\breve{q}'$ is also 6.

As another example, consider the queries

$$\begin{aligned} p(sum(y)) &\leftarrow r(y) \wedge r(z) \wedge r(w) \wedge 0 < y \wedge 0 \leq z \wedge 0 < w \\ p'(sum(y)) &\leftarrow r(y) \wedge r(z) \wedge r(w) \wedge 0 \leq y \wedge 0 \leq z \wedge 0 < w. \end{aligned}$$

The core of the first query does not have any linearization that is a constant query. The core of the second has four, namely

$$\begin{aligned} p'_1(0) &\leftarrow r(0) \wedge r(z) \wedge 0 < z \\ p'_2(0) &\leftarrow r(0) \wedge r(z) \wedge 0 < z \\ p'_3(0) &\leftarrow r(0) \wedge r(z) \wedge r(w) \wedge 0 < z \wedge z < w \\ p'_4(0) &\leftarrow r(0) \wedge r(z) \wedge r(w) \wedge 0 < w \wedge w < z. \end{aligned}$$

Query p'_1 is obtained from $L_1 = \{0 = y = w < z\}$, while p'_2 is obtained from $L_2 = \{0 = y < z = w\}$. They form two equivalence classes of isomorphic queries, $\{p'_1,\ p'_2\}$ and $\{p'_3,\ p'_4\}$. Both classes are associated to the empty set. Since the summation constant is always 0, we obtain in both cases 0 as the sum. This is also the result that we obtain when summing over the empty set. Thus, the expansions of $\breve{q}$ and $\breve{q}'$ are in balance.

Let $\mathcal{L}^v_D(\breve{q})$ consist of those linearizations that do not identify y with a constant. We say that two linear expansions $(\breve{q}_L)_L$, $(\breve{q}'_M)_M$ are *variable isomorphic* if there is a bijection $\mu\colon \mathcal{L}^v_D(\breve{q}) \to \mathcal{L}^v_D(\breve{q}')$ such that $\breve{q}_L$ and $\breve{q}'_{\mu(L)}$ are isomorphic.

THEOREM 9.8 (SUFFICIENT CONDITION FOR EQUIVALENCE). *Let q and q' be sum-queries. Then q and q' are equivalent if*

—*$\breve{q}$ and $\breve{q}'$ are weakly set-equivalent, and*

—*$\breve{q}$ and $\breve{q}'$ have linear expansions that are in balance and variable isomorphic.*

PROOF. The proof is analogous to the proof of Theorem 8.7. We therefore just give an outline. Since $\breve{q}$ and $\breve{q}'$ are weakly set-equivalent, $\breve{q}$ returns a group for the tuple $\bar{d}$ if and only if $\breve{q}'$ does.

Let $(\breve{q}_L)_L$, $(\breve{q}'_M)_M$ be linear expansions as in the statement of the theorem. The fact that $(\breve{q}_L)_L$, $(\breve{q}'_M)_M$ are in balance, guarantees that those values in the groups of $\bar{d}$ that are equal to constants in the queries sum up to the same results in q and in q'. The isomorphism of $(\breve{q}_L)_{L\in\mathcal{L}^v_D(\breve{q})}$ and $(\breve{q}'_M)_{M\in\mathcal{L}^v_D(\breve{q}')}$ guarantees that values in the groups of $\bar{d}$ that are distinct from constants are returned with the same multiplicity by each core. Hence, both queries return the same sums. □

Example 9.9. Consider again the queries p and p' in Example 9.7. As shown before, the linear expansions of their cores are in balance. It is also easy to check that the linear expansions are variable isomorphic. Thus, by Theorem 9.8, p and p' are equivalent.

The condition in Theorem 9.8 does not take into consideration the set of constants over which the linear expansions are taken. In order to derive a necessary condition, care must be taken when choosing the constants for the linear expansions. In particular, we will always take the linear expansions over the set of virtual constants of the queries. By Theorem 4.9, this will ensure that the linear expansions are reduced.

We present a series of lemmas that will enable us to prove a necessary condition for *sum*-query equivalence.

LEMMA 9.10 (EQUIVALENCE IMPLIES WEAK SET-EQUIVALENCE). *Let q, q' be sum-queries. If q and q' are equivalent, then $\breve{q}$ and $\breve{q}'$ are weakly set equivalent.*

PROOF. Since q and q' are equivalent, q returns a sum for the tuple $\bar{d}$ if and only if q' does. Hence, $\breve{q}$ and $\breve{q}'$ are weakly set-equivalent. $\square$

LEMMA 9.11 (VARIABLE ISOMORPHISM IMPLIES BALANCE). *Let $q(\bar{x}, y) \leftarrow R \wedge C$, $q'(\bar{x}, y) \leftarrow R' \wedge C'$ be sum-queries, and let D be a set of constants such that $D = vc_C(D_0) \cup vc_{C'}(D_0)$, where D_0 comprises the constants of q and q'. Suppose that q and q' are equivalent and $\breve{q}$ and $\breve{q}'$ have linear expansions over D that are variable isomorphic. Then these linear expansions are in balance.*

PROOF. This proof is analogous to the proof of Theorem 8.8. We therefore just give an outline.

Suppose that q and q' are equivalent, and let $(\breve{q}_L)_L$ and $(\breve{q}'_M)_M$ be linear expansions of the cores of q and q' over D. Note that by the choice of D the linear expansions are reduced.

We give a proof by contradiction. Thus, assume that the linear expansions are not in balance. Then there is a pair of associated classes of constant queries Q_0, Q'_0 for which

$$\sum_{\substack{Q \in \mathcal{Q}_C \\ Q \sim Q_0}} \sum_{\breve{q}_L \in Q} \sigma(\breve{q}_L) \neq \sum_{\substack{Q' \in \mathcal{Q}'_C \\ Q' \sim Q'_0}} \sum_{\breve{q}'_M \in Q'} \sigma(\breve{q}'_M).$$

Let $\bar{Q}_0$ be the set of constant queries $\breve{q}_L$ that are weakly isomorphic to the queries in Q_0, that is, queries that are in some class Q such that $Q \sim Q_0$, and let $\bar{Q}'_0$ be the set of constant queries $\breve{q}'_M$ that is defined analogously. We can choose the pair of Q_0 and Q'_0 such that the queries in $\bar{Q}_0$ and $\bar{Q}'_0$ have a minimal number of variables and relational atoms.

Next, we construct a database $\mathcal{D}$ out of one of the queries. Over this database, $\bar{Q}_0$ and $\bar{Q}'_0$ return groups that sum up to distinct numbers. Because of the minimality of Q_0 and Q'_0, one can show similarly to the proof of Theorem 8.7, that the difference cannot be compensated by classes of constant queries that are not associated with Q and Q'. Thus, the difference can only be compensated by variable queries, which means that the linearizations of q and q' are not variable isomorphic. $\square$

LEMMA 9.12 (EQUIVALENCE IMPLIES VARIABLE ISOMORPHISM). *Consider the sum-queries $q(\bar{x}, y) \leftarrow R \wedge C$, $q'(\bar{x}, y) \leftarrow R' \wedge C'$ and let D be a set of constants such that $D = vc_C(D_0) \cup vc_{C'}(D_0)$, where D_0 comprises the constants of q and q'. If q and q' are equivalent, then the linear expansions of $\breve{q}$ and $\breve{q}'$ over D are variable isomorphic.*

PROOF. Again, we prove the claim by contradiction. Assume that $\breve{q}$ and $\breve{q}'$ have linear expansions over D that are *not* variable isomorphic. We show that q and q' are not equivalent.

Let $Q = [q_L]$, $Q' = [q'_M]$ be two associated classes of isomorphic variable queries that are not of the same cardinality. Such classes exist since q and q' are not variable isomorphic.

As in the proof of Theorem 8.8, we choose the classes Q, Q' in such a way that the number of variables and relational atoms in their queries is minimal. We construct two databases, $\mathcal{D}_1, \mathcal{D}_2$, from one of the queries, say $q_L(\bar{s}, y) \in Q$, using assignments γ_1, γ_2 that satisfy L. Moreover, we choose γ_1, γ_2 such that

—they differ on the summation variable, that is, $\gamma_1(y) \neq \gamma_2(y)$ *and*

—they agree on every other variable.

It is possible to choose such assignments since q_L is reduced and is not a constant query.

Note all queries in Q are isomorphic to $q_L(\bar{s}, y)$. Therefore, one can show with the same argument as in the proof of Theorem 8.8 that each query in Q is satisfied by exactly one assignment over $\mathcal{D}_i$ and that the summation variable y is always mapped to $\gamma_i(y)$ by those assignments.

We denote by $\sigma_1(Q)$ the total sum of values returned by the queries in class Q for the group $\gamma_1(\bar{s})$ over $\mathcal{D}_1$. Analogously, we define $\sigma_2(Q)$ as the sum of values returned by Q for the group $\gamma_2(\bar{s})$ over $\mathcal{D}_2$. Thus, for $i = 1, 2$, we have

$$\sigma_i(Q) = |Q| * \gamma_i(y),$$

where $|Q|$ denotes the cardinality of class Q. We define $\sigma_i(Q')$ in a similar way. It is not difficult to see that

$$\sigma_1(Q) - \sigma_1(Q') \neq \sigma_2(Q) - \sigma_2(Q'), \tag{11}$$

since $|Q| \neq |Q'|$ and $\gamma_1(y) \neq \gamma_2(y)$.

We show that for all other pairs of associated classes of isomorphic queries in the linear expansions, the differences of the corresponding sums are the same, that is, for all other classes R, R', we have

$$\sigma_1(R) - \sigma_1(R') = \sigma_2(R) - \sigma_2(R'). \tag{12}$$

Let R, R' be associated classes of isomorphic variable queries different from Q and Q'. Suppose that $q_K \in R$ and $q'_N \in R'$. Assume first that R and R' are classes of variable queries. We consider three cases.

Case 1. The query q_K has fewer variables than q_L or as many variables but fewer relational atoms. We chose Q and Q' to be minimal counter-examples. Therefore R and R' are of the same size and contribute equal amounts to the summations. Thus, for $i = 1, 2$, we have

$$\sigma_i(R) = \sigma_i(R').$$

Case 2. The query q_K has more variables than q_L or as many variables but more relational atoms. Then no query in R or R' can return any values at all when evaluated over $\mathcal{D}_1$ and $\mathcal{D}_2$, and hence, $\sigma_i(R) = \sigma_i(R') = 0$.

Case 3. The query $q_K(\bar{s}', y')$ has as many variables and as many atoms as q_L. Let γ'_1 be a satisfying assignment over $\mathcal{D}_1$ that maps $\bar{s}'$ to $\gamma_1(\bar{s})$. Note that there can be at most one such assignment because $\mathcal{D}_1$ has been constructed from q_L, the query q_K has as many variables as q_L, and both K and L are linearizations that constrain variables to values distinct from the constants in q. (If no such assignment exists, then q_K does not return any summation value for the group $\gamma_1(\bar{s})$ at all.) Similarly, let γ_2' be a satisfying assignment over $\mathcal{D}_2$ that maps $\bar{s}'$ to $\gamma_2(\bar{s})$.

We prove that $\gamma_1'(y') \neq \gamma_1(y)$. Assume instead that $\gamma_1(y) = \gamma_1'(y')$. We show that in this case the composition $\gamma_1^{-1} \circ \gamma_1'$ is an isomorphism from q_K to q_L. The composition is a bijection because both, γ_1 and γ_1' are injective and map the variables of q_L and q_K to the same set of constants. It is a homomorphism in both directions due to the construction of $\mathcal{D}_1$ and because $\gamma_1(\bar{s}) = \gamma_1'(\bar{s}')$ and $\gamma_1(y) = \gamma_1'(y')$. However, the isomorphism of q_K and q_l implies that $R = Q$, in contradiction to our assumption about the class R.

We now prove that Eq. (12) holds in the present case as well. Recall that the assignments γ_1 and γ_2, from which $\mathcal{D}_1$ and $\mathcal{D}_2$ have been constructed, differ only in their values for the summation variable y. Thus, if $\gamma_1'(y') \neq \gamma_1(y)$, then $\gamma_2'(y') \neq \gamma_2(y)$ as well, and $\gamma_1'(y') = \gamma_2'(y')$. Hence, $\sigma_1(R) = \sigma_2(R)$ and $\sigma_1(R') = \sigma_2(R')$, which implies that the differences of the corresponding sums are the same.

We have shown the correctness of Eq. (12) for associated classes of variable queries. Now, suppose that R and R' are associated classes of constant queries. The linear expansions were taken over all the virtual constants. Therefore, by Theorem 4.9, the queries in the linear expansions are reduced. Hence, the constants in the summation terms of R and R' are different from $\gamma_1(y)$ and from $\gamma_2(y)$. Therefore, as in Case 3 above, $\sigma_1(R) = \sigma_2(R)$ and $\sigma_1(R') = \sigma_2(R')$. Hence, Eq. (12) holds for classes of constant queries as well.

Let $\sigma_1(q)$ be the summation value that q returns for $\gamma_1(\bar{s})$ over $\mathcal{D}_1$ and let $\sigma_1(q')$ be the value that q' returns. Let $\sigma_2(q)$ and $\sigma_2(q')$ be the summation values returned by q and q' over $\mathcal{D}_2$. From Eqs. (11) and (12), we conclude that

$$\sigma_1(q) - \sigma_1(q') \neq \sigma_2(q') - \sigma_2(q').$$

Therefore, even if the queries q, q' return the same sums over one of the databases $\mathcal{D}_1$, $\mathcal{D}_2$, they return different sums over the other database. Therefore, q is not equivalent to q', as required. □

THEOREM 9.13 (NECESSARY CONDITION FOR EQUIVALENCE). *Let $q(\bar{x}, y) \leftarrow R \wedge C$, $q'(\bar{x}, y) \leftarrow R' \wedge C'$ be sum-queries, and let D be a set of constants such that $D = vc_C(D_0) \cup vc_{C'}(D_0)$, where D_0 comprises the constants of q and q'. If q and q' are equivalent, then*

—*$\breve{q}$ and $\breve{q}'$ are weakly set-equivalent,*

—*$\breve{q}$ and $\breve{q}'$ have linear expansions over D that are in balance and variable isomorphic.*

PROOF. The theorem follows from Lemmas 9.10, 9.11, and 9.12. □

The two preceding theorems can be specialized for queries without comparisons.

THEOREM 9.14 (SUM-QUERIES WITHOUT COMPARISONS). *Let q and q' be sum-queries without comparisons and $\breve{q}$ and $\breve{q}'$ be their cores. Then, the following are equivalent:*

(1) *q and q' are equivalent;*

(2) *$\breve{q}$ and $\breve{q}'$ are bag-set-equivalent;*

(3) *$\breve{q}$ and $\breve{q}'$ are isomorphic.*

PROOF. Properties (2) and (3) are equivalent by Theorem 8.5. Property (2) implies Property (1) by Proposition 9.1. To complete the proof, it suffices to show that Property (1) implies Property (3).

If q and q' are equivalent, then by Theorem 9.13 there are linear expansions $(\breve{q}_L)_L$ and $(\breve{q}'_M)_M$ of $\breve{q}$ and $\breve{q}'$ that are variable isomorphic.

Let $\breve{q}_L$ be a linearization in this expansion such that L does not identify any two terms. Such a linearization always exists. Then, there is a linearization M such that $\breve{q}_L$ and $\breve{q}'_M$ are isomorphic. The linearization M does not identify any terms of q', either. Otherwise, q' would have more terms than q, which by Corollary 4.5 implies that q' has more variables than q. Hence, if M_0 is a linearization that does not identify terms of q', then there is no query in $(\breve{q}_L)_L$ that is isomorphic to $\breve{q}'_{M_0}$.

Let θ be an isomorphism from $\breve{q}_L$ to $\breve{q}'_M$. In particular, θ is a relational isomorphism. Hence, since L and M do not identify terms, θ is an isomorphism from $\breve{q}$ to $\breve{q}'$. □

Theorem 9.5 could as well have been obtained as a corollary to Theorem 9.13. To see this, note that the cores of queries without constants don't have virtual constants and that their linear expansions don't contain constant queries. Thus, the condition in the theorem boils down to requiring that the cores of the two queries have isomorphic linear expansions. By Theorem 8.7, this implies that the cores are bag-set-equivalent. However, it is not clear how to modify the characterization of Theorem 9.13 for the case of databases that satisfy integrity constraints. As mentioned earlier, the proof of Theorem 9.5 is also valid in that case, provided the constraints are invariant under strictly monotonic mappings.

The conditions in Theorem 9.13 are necessary for the equivalence of *sum*-queries, and by Theorem 9.8, they are also sufficient. This gives us an upper complexity bound for the problem of deciding equivalence of *sum*-queries.

THEOREM 9.15 (UPPER COMPLEXITY BOUND). *Equivalence of sum-queries can be decided with polynomial space.*

PROOF. Let q and q' be two *sum*-queries and D_0 be the set of constants occurring in q and q'.

We first observe that the set of virtual constants $D = vc_C(D_0) \cup vc_{C'}(D_0)$ is of polynomial size with respect to q and q' and can be computed with polynomial space. This follows from Lemma 4.6 and Item 5 of Theorem 4.1.

Then, we note that weak set-equivalence is in Π_2^P, and therefore can be decided with polynomial space.

Finally, with algorithms similar to the one described in the proof of Theorem 8.9, one can check that the cores $\breve{q}$, $\breve{q}'$ of q, q', respectively, have linear expansions that are in balance and variable isomorphic. □

10. *Conclusion*

We have established syntactic characterizations for equivalences among aggregate queries. Understanding why and how queries can be equivalent is a prerequisite for designing optimization algorithms and algorithms to rewrite queries using views. In this way, our work lays foundations for optimizing aggregate queries and for solving the view rewriting problem. Our main contributions can be summarized as follows:

Nonaggregate Queries. We provide a complete characterization for equivalence under bag-set semantics of conjunctive nonaggregate queries with constants and comparisons. Such queries are common in SQL, since bag-set semantics is the default semantics of SQL. Characterizing equivalence under bag-set semantics has been an open problem since bag-set semantics were first considered in Chaudhuri and Vardi [1993] and Ioannidis and Ramakrishnan [1995].

Aggregate Queries. We present complete characterizations for equivalences among *max*-queries, *count*-queries and *sum*-queries. For *cntd*-queries, we present a sufficient characterization for equivalence. Our results are unique in that we consider general and common types of queries.

We leave for future research the problem of finding a complete characterization for equivalence of *cntd*-queries. Finding tight upper and lower bounds for equivalence is another important open issue.

We have not addressed equivalences among aggregate queries with a `having` clause. The work of Ross et al. [1998] on aggregation constraints, which considers the satisfiability of a conjunction of aggregation constraints, may be relevant for this problem.

ACKNOWLEDGMENTS. The authors thank the anonymous referees for helpful suggestions and for pointing out important related work.

REFERENCES

AHO, A., SAGIV, Y., AND ULLMAN, J. 1979. Efficient optimization of a class of relational expressions. *ACM Trans. Datab. Syst. 4,* 4, 435–454.

ALBERT, J. 1991. Algebraic properties of bag data types. In *Proceedings of the 17th International Conference on Very Large Data Bases* (Barcelona, Spain). Morgan-Kaufmann, San Francisco, CA, 211–219.

BENEDIKT, M., AND KEISLER, H. 1997. Expressive power of unary counters. In *Proceedings of the 6th International Conference on Database Theory* (Delphi, Greece). F. Afrati and P. Kolaitis, Eds. Lecture Notes in Computer Science, vol. 1186. Springer-Verlag, New York, 291–305.

BENEDIKT, M., AND LIBKIN, L. 1999. Exact and approximate aggregation in constraint query languages. In *Proceedings of the 18th Symposium on Principles of Database Systems* (Philadelphia, PA). C. Papadimitriou, Ed. ACM, New York, 102–113.

CALVANESE, D., DE GIACOMO, G., LENZERINI, M., AND VARDI, M. Y. 2000. Containment of conjunctive regular path queries with inverse. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning* (Breckenridge, CO). A. G. Cohn, F. Giunchiglia, and B. Selman, Eds. Morgan-Kaufmann, San Francisco, CA.

CALVANESE, D., DE GIACOMO, G., LENZERINI, M., AND VARDI, M. Y. 2001. View-based query answering and query containment over semistructured data. In *Proceedings of the 8th International Workshop on Database Programming Languages* (Frascati, Italy). Springer-Verlag, New York, 40–61.

CALVANESE, D., DE GIACOMO, G., AND VARDI, M. Y. 2003. Decidable containment of recursive queries. In *Proceedings of the 9th International Conference on Database Theory* (Siena, Italy). Lecture Notes in Computer Science. Springer-Verlag, New York, 330–345.

CHANDRA, A., AND MERLIN, P. 1977. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing* (Boulder, CO). ACM, New York, 77–90.

CHAUDHURI, S., AND DAYAL, U. 1997. An overview of data warehousing and OLAP technology. *SIGMOD Record 26,* 1, 65–74.

CHAUDHURI, S., KRISHNAMURTHY, S., POTARNIANOS, S., AND SHIM, K. 1995. Optimizing queries with materialized views. In *Proceedings of the 11th International Conference on Data Engineering* (Taipei, China). P. S. Yu and A. Chen, Eds. IEEE Computer Society Press, Los Alamitos, CA.

CHAUDHURI, S., AND VARDI, M. 1993. Optimization of real conjunctive queries. In *Proceedings of the 12th Symposium on Principles of Database Systems* (Washington, DC). ACM, New York.

CHEKURI, C., AND RAJARAMAN, A. 2000. Conjunctive query containment revisited. *Theoret. Comput. Sci. 239,* 2, 211–229.

COHEN, S., NUTT, W., AND SAGIV, Y. 2001. Equivalences among aggregate queries with negation. In *Proceedings of the 20th Symposium on Principles of Database Systems* (Santa Barbara, CA). ACM, New York, 155–166.

COHEN, S., NUTT, W., AND SAGIV, Y. 2003. Containment of aggregate queries. In *Proceedings of the 9th International Conference on Database Theory* (Siena, Italy). Lecture Notes in Computer Science. Springer-Verlag, New york.

COHEN, S., NUTT, W., AND SAGIV, Y. 2005. Equivalences among aggregate queries with negation. *ACM Trans. Computat. Logic 6,* 2, 328–360.

COHEN, S., NUTT, W., AND SEREBRENIK, A. 1999. Rewriting aggregate queries using views. In *Proceedings of the 18th Symposium on Principles of Database Systems* (Philadelphia, PA). C. Papadimitriou, Ed. ACM, New york.

DOBRA, A., GAROFALAKIS, M. N., GEHRKE, J., AND RASTOGI, R. 2002. Processing complex aggregate queries over data streams. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data* (Madison, WI). ACM, New york, 61–72.

ETESSAMI, K., AND IMMERMAN, N. 2000. Tree canonization and transitive closure. *Inf. Comput. 157,* 1-2, 2–24.

GRUMBACH, S., RAFANELLI, M., AND TININI, L. 1999. Querying aggregate data. In *Proceedings of the 18th Symposium on Principles of Database Systems* (Philadelphia, PA). C. Papadimitriou, Ed. ACM, New York, 174–183.

GUO, S., SUN, W., AND WEISS, M. 1996a. On satisfiability, equivalence, and implication problems involving conjunctive queries in database systems. *IEEE Trans. Knowl. Data Eng. 8,* 4, 604–616.

GUO, S., SUN, W., AND WEISS, M. 1996b. Solving satisfiability and implication problems in database systems. *ACM Trans. Datab. Syst. 21,* 2, 270–293.

GUPTA, A., HARINARAYAN, V., AND QUASS, D. 1995. Aggregate query processing in data warehouses. In *Proceedings of the 21st International Conference on Very Large Data Bases* (Zurich, Switzerland). Morgan-Kaufmann, San Francisco, CA.

GUPTA, A., AND MUMICK, I. S. 1999. *Materialized Views: Techniques, Implementations, and Applications*. MIT Press, Cambridge, MA.

HELLA, L., LIBKIN, L., NURMONEN, J., AND WONG, L. 1999. Logics with aggregate operators. In *Proceedings of the 14th IEEE Symposium on Logic in Computer Science* (Trento, Italy). IEEE Computer Society Press, Los Alamitos, CA, 35–44.

IOANNIDIS, Y., AND RAMAKRISHNAN, R. 1995. Beyond relations as sets. *ACM Trans. Datab. Syst. 20,* 3, 288–324.

JOHNSON, D., AND KLUG, A. 1983. Optimizing conjunctive queries that contain untyped variables. *SIAM J. Comput. 12,* 4, 616–640.

KLUG, A. 1982. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *J. ACM 29,* 3, 699–717.

KLUG, A. 1988. On conjunctive queries containing inequalities. *J. ACM 35,* 1, 146–160.

LEVY, A., MUMICK, I. S., SAGIV, Y., AND SHMUELI, O. 1993. Equivalence, query-reachability, and satisfiability in datalog extensions. In *Proceedings of the 12th Symposium on Principles of Database Systems* (Washington, DC). ACM, New York, 109–122.

LEVY, A., MUMICK, I. S., SAGIV, Y., AND SHMUELI, O. 2001. Static analysis in datalog extensions. *J. ACM 48,* 5, 971–1012.

LEVY, A., AND SAGIV, Y. 1995. Semantic query optimization in datalog programs. In *Proceedings of the 14th Symposium on Principles of Database Systems* (San Jose, CA). ACM, New York, 163–173.

MADDEN, S., SZEWCZYK, R., FRANKLIN, M. J., AND CULLER, D. 2002. Supporting aggregate queries over ad-hoc wireless sensor networks. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications* (Callicoon, NY). IEEE Computer Society Press, Los Alamitos, CA, 49–58.

MIKLAU, G., AND SUCIU, D. 2002. Containment and equivalence for an XPath fragment. In *Proceedings of the 21st Symposium on Principles of Database Systems* (Madison, WI). ACM, New York, 65–76.

NUTT, W., SAGIV, Y., AND SHURIN, S. 1998. Deciding equivalences among aggregate queries. In *Proceedings of the 17th Symposium on Principles of Database Systems* (Seattle, WA). J. Paredaens, Ed. ACM, New York, 214–223. Long version as Report of Esprit LTR DWQ.

ÖZSOYOĞLU, G., ÖZSOYOĞLU, Z., AND MATOS, V. 1987. Extending relational algebra and relational calculus with set-valued attributes and aggregate functions. *ACM Trans. Datab. Syst. 12,* 4, 566–592.

POPA, L., AND TANNEN, V. 1999. An equational chase for path-conjunctive queries, constraints, and views. In *Proceedings of the 7th International Conference on Database Theory* (Jerusalem, Israel). C. Beeri and P. Buneman, Eds. Lecture Notes in Computer Science, vol. 1540. Springer-Verlag, New York, 39–57.

ROSS, K., SRIVASTAVA, D., STUCKEY, P., AND SUDARSHAN, S. 1998. Foundations of aggregation constraints. *Theoret. Comput. Sci. 193,* 1-2, 149–179.

SAGIV, Y., AND SARAIYA, Y. 1992. Minimizing restricted-fanout queries. *Disc. Appl. Math. 40*, 245–264.

SAGIV, Y., AND YANNAKAKIS, M. 1981. Equivalence among relational expressions with the union and difference operators. *J. ACM 27,* 4, 633–655.

SRIVASTAVA, D., DAR, S., JAGADISH, H., AND LEVY, A. 1996. Answering queries with aggregation using views. In *Proceedings of the 22nd International Conference on Very Large Data Bases* (Bombay, India). Morgan-Kaufmann, San Francisco, CA.

VAN DER MEYDEN, R. 1992. The complexity of querying indefinite data about linearly ordered domains. In *Proceedings of the 11th Symposium on Principles of Database Systems* (San Diego, CA). ACM, New York, 331–345.

RECEIVED JANUARY 2004; REVISED FEBRUARY 2006; ACCEPTED JANUARY 2007