FREIE UNIVERSITÄT BOZEN
LIBERA UNIVERSITÀ DI BOLZANO
FREE UNIVERSITY OF BOZEN - BOLZANO

Faculty of Computer Science, Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy
Tel: +39 04710 16000, fax: +39 04710 16009, http://www.inf.unibz.it/krdb/

KRDB Research Centre Technical Report:

Equivalence of Aggregate Queries with Incomplete Information and Dependencies

Camilo Thorne

Affiliation	KRDB Research Centre, Faculty of Computer Science Free University of Bozen-Bolzano Piazza Domenicani 3, 39100, Bolzano, Italy
Corresponding author	Camilo Thorne cthorne@inf.unibz.it
Keywords	Query languages, incomplete databases, aggregate queries
Number	KRDB08-05
Date	1-7-08
URL	http://www.inf.unibz.it/krdb/pub/

Equivalence of Aggregate Queries with Incomplete Information and Dependencies

Camilo Thorne

January 2008

KRDB Research Centre
Free University of Bozen-Bolzano
3, Piazza Domenicani, 39100 (Italy)
cthorne@inf.unibz.it

Contents

Introduction

Information systems handle typically large repositories of heterogeneous data, structured following different, if overlapping, data models. Relational databases, temporal databases, information retrieval systems (targeting document repositories, whether raw text or the web), data warehouses, object databases, etc. In many, if not in most, cases, this will involve storing, removing, updating or accessing numerical data or in performing numerical computations out of the information the repository may contain. Numerical data and ordered domain types will be widespread, if not ubiquitous in the data repositories. Their common denominator being their containing information about some domain. Think in university professors. They earn a salary, have an age, a name, may be married or single, teach courses, supervise students, write scientific papers, etc. Data sources from such a domain may contain strings of characters, .pdf documents, integers, urls and so forth.

Particularly, in the case of accessing information, we may want to mine the data for decision support (a typical use case). We may want to automatically classify it (whether by supervides or usupervised means), perform regression, concept learning. Or, simply, retrieve some piece of information we might be interested in knowing (e.g. "How many computer science professors are members of W3C?"). In either case, retrieval should be optimal, in the sense that the greatest amount of data should be retrieved (computed by the system) with the least space and time system resources (cf. [Elmasri2004]), which is critical, given the massive amounts of data that we may have, ultimately, to process. This constraint argues in favor of systems based on Relational database (DB) systems and database views and ultimately, we believe, in *ontology-based data integration*.

In ontology-based data integration, an ontology Σ (actually, a set of data integrity constraints) layer serves as a very high level representation or description of the data, which remains hidden to the end user. Queries q, therefore, are formulated w.r.t. this conceptual model and run over *incomplete instances* $\mathbf{I}$: ontologies can be seen as view definitions over the sources and the incomplete instances as (incomplete) retrieved global databases, that is, as the sets of corresponding virtual tables or views that may not necessarily contain all the data of the sources (which may be infinitely many). This can be performed following the global-as-view (GAV) approach to data-integration (cf.[Cali2004, Calvanese2007D]). Logically, this amounts to considering $\langle \Sigma, \mathbf{I} \rangle$ an *incomplete database* (or a *knowledge base* $\mathbf{K}$, a closely akin notion, cf. [Rosati2007, Calvanese2007D]), where query answering works under the *open world assumption* (OWA) and is thus modelled as the following logical entailment problem:

$$\langle \Sigma, \mathbf{I} \rangle \models q[\overline{\boldsymbol{t}}/\overline{\boldsymbol{x}}].$$

We believe, moreover, that this effort can be coupled with that of building controlled natural language (say, English) interfaces (NLIDBs). Which entails, to a great extent (cf. [Thorne2007E]), defining fragments of natural language (English), whose declarative sentences compositionally translate into facts and constraints (and might be used to, for instance, add or update data) and whose questions compositionally translate into formal queries, as has been done for OWL ontologies by the Attempto project (cf. Attempto Controlled English, [Fuchs]) and as was proposed time ago by Clifford in [Clifford1988]. Why? First, because using English is simpler than using, say, SQL (at least for a non-expert, i.e. a non-IT enginneer). Second, because we use views. Therefore, the source schemas can be disregarded when building the question/query, since the rewriting of these queries, built over the ontology or global schema, w.r.t. these source schemas can be delegeated to the data integration system (based on relational database technologies) running in the back end. The can enhance, therefore, both the portability and the usability of the front end.

In this report we will look to two of the (many) logical problems that arise in this setting. Namely, at the problems of query containment (QC) and query answering (QA) for conjunctive aggregate queries (CQs + agg) in an incomplete information setting. These problems have been studied for relational databases with *complete information* by, above all, Cohen *et. al.* (cf. [Cohen2007]) and SQL-like bag semantics, which makes no sense in a data integration setting inasmuch as we have to collapse different source tables into a single virtual table in the retrieved incomplete instance (cf. [Cali2004]). We plan to look at their counterparts for incomplete databases, check for their decidability and, if so, for their complexity class. Furthermore, as some corpora analysis shows, natural language questions (to NLIDBs) that target databases with numerical data frequently express aggregate queries: we may ask for an average, a sum, etc. (cf. [Thorne2007D]). Not to speak about how basic they are in other settings, like data mining, data warehouses, etc (cf. [Elmasri2004]).

1 Relational Databases

What is a relational database (DB)? If we think for a moment in commercial database systems we can say, to the best of our intuition: it is a set of records (or data) about a certain domain stored (or structured) in tables, defined over a database schema (a set of so-called relation names) and satisfying a certain number of integrity constraints or dependencies and over which queries (information requests) are run or evaluated. Interestingly, this state of affairs can be modelled by logic, through the *relational data model* (RDM) using the so-called unnamed view on relations (cf. [Abiteboul1995]), where database states become finite first order interpretation structures. Queries and constraints, on the other hand, can be captured by first order logic formulas. Thus, running a query against a database becomes nothing else than evaluating a formula over one of the finite models of the dependencies.

1.1 Databases

A *database schema* $\mathbf{R} = \{R_1, ..., R_n\}$ is a finite set of *relation names* with an associated *arity* (an integer $n \geq 0$). In addition, let **Dom** denote a (possibly) countably infinite non-empty set of constants called *domain*, then: a *tuple* over **Dom** is every $\overline{\boldsymbol{t}} \in \mathbf{Dom}^n$, for some $n \in \mathbb{N}$ and a *relation instance* of a relation name R of arity n is a relation $R(\mathbf{I}) \subseteq \mathbf{Dom}^n$ (a set of tuples). In other words, database records are modelled as tuples of contants and tables as n-ary relations. Given this, we can define what a database is: a *database instance* $\mathbf{I}$ over schema $\mathbf{R}$ is a finite set of relation instances whose *active domain* is the finite set $adom(\mathbf{I}) \subsetneq \mathbf{Dom}$ of (pairwise distinct) domain constants occuring among the relations in $\mathbf{I}$. In other words, it is a finite first order structure $\mathbf{I} = \langle adom(\mathbf{I}); \{R_i(\mathbf{I})\}_{i\in[1,n]}\rangle$ over the schema $\mathbf{R}$ which is, essentially, a (finite) first order logic signature (cf. [Abiteboul1995, Vardi1982]). The integer $\#(adom(\mathbf{I}))$ is called the *size* of the instance $\mathbf{I}$. We will denote by $Inst(\mathbf{R})$ the class of all such structures. In brief, database records are modelled as tuples of contants, tables as n-ary relations, schemas as first order signatures and, last but not least, database states as schema instances. Domains model, on the other hand, their domains of interest. By assumption, a DB is meant to exhaust all the information of its domain of interest, i.e., to be *complete*.

1.2 Conjunctive Queries

Now that we have characterized databases, we must do the same with queries. Queries are a way of specifying, declaratively, a request for information stored in a database. As such, they are meant to (so to speak) unambigously define the data (a set of tuples) we seek to retrieve, independently of the algorithms set to actually carry on with these tasks and which remain hidden to the end user, as is the case with SQL, the industry standard for relational query languages. A first order logic abstraction of the most common and simplest SQL queries is provided by the class of conjunctive queries (CQs), which correspond to the SELECT-FROM-WHERE fragment of SQL with, eventually, inequalities and numerical comparisons. This class lies, basically, within the existential-positive fragment of FOL (cf. [Abiteboul1995]). By covering boolean disjunction we will extend this class to the class of unions of conjunctive queries (UCQs), corresponding to the SELECT-FROM-WHERE-UNION fragment of SQL.

1.2.1 Syntax

A *conjunctive query* (CQ) q with comparisons over a relational schema $\mathbf{R}$ is an expression of the form:

$$q(\overline{\boldsymbol{x}}) \leftarrow \exists \overline{\boldsymbol{y}}(\Psi(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}) \wedge \Upsilon(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}))$$

where $\overline{\boldsymbol{x}}$ is a possibly empty finite sequence of *distinguished variables*. The length of the sequence $\overline{\boldsymbol{x}}$ is called the *arity* of the query. The expression to the left of $\leftarrow$ is called the *head* of the query and the expression to the left its *body*. Distinguished variables occur free in the body. We denote by $Var(q)$ the set of all the variables in q, distinguished or other. $\Psi(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}})$ is a conjunction of atoms and $\Upsilon(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}})$ a conjunction of comparisons of the form $t\rho t'$, with $\rho \in \{<, >, =, \leq, \geq\}$. A query without comparisons is said to be *relational*. The *size* of q, denoted $size(q)$, is simply $|\exists\overline{\boldsymbol{y}}(\Psi(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}) \wedge \Upsilon(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}))|$, i.e., its *rank* or, put otherwise, the depth of its parse tree. When the sequence $\overline{\boldsymbol{x}}$ of distinguished variables is empty we say that the query is *boolean*. Conjunctive queries can be extended to cover boolean disjunction giving way to *unions of conjuntive queries* (UCQs) of the form:

$$q(\overline{\boldsymbol{x}}) \leftarrow \bigvee_{i=0}^{k} \exists\overline{\boldsymbol{y}}_i(\Psi_i(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}_i) \wedge \Upsilon_i(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}_i))$$

of size $size(q) = |(\Psi_1(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}_1) \wedge \Upsilon_1(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}_1))| + ... + |(\Psi_k(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}_k) \wedge \Upsilon_k(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}_k))|$. Note that CQs $\subsetneq$ UCQs.

1.2.2 Semantics

Finally, the *result set* or *semantics* of a CQ or UCQ q on input database $\mathbf{I}$, denoted $q(\mathbf{I})$, is built by collecting the values of all the substitutions $\sigma\colon Var(q) \rightarrow \mathbf{Dom}$ under which $\mathbf{I}$ is a model of $q\sigma$. This is formally defined as follows:

$$q(\mathbf{I}) := \{\sigma(\overline{\boldsymbol{x}}) | \mathbf{I} \models q\sigma\}$$

Note that if q is boolean, then: (i) If q evaluates to truth in $\mathbf{I}$, $q(\mathbf{I}) = \{\langle\rangle\}$. (ii) If q evaluates to false in $\mathbf{I}$, $q(\mathbf{I}) = \emptyset$. In other words, the empty tuple and the empty set simulate the behavior of truth values in the database setting. Inspired by this analogy, it is standard to say a CQ or UCQ q is *satisfiable* if $q(\mathbf{I}) \neq \emptyset$ for some instance $\mathbf{I}$ and *unsatisfiable* otherwise.

Example 1.1. Let $\mathbf{R} := \{Loves^2\}$ be a schema. An instance over $\mathbf{R}$ will then be

$$\mathbf{I} := \{Loves(Julian, Mary), Loves(Lucas, Mary)\}.$$

A CQ an expression

$$q := q(y, x) \leftarrow Loves(x, y)$$

of semantics $q(\mathbf{I}) := \{\langle Mary, Julian\rangle, \langle Mary, Lucas\rangle\}$ and of size $|Loves(x, y)| = 1$. It corresponds to the SQL query

SELECT Loves.2, Loves.1
FROM Loves

and is read "list all the people who love each other".

1.3 Basic Properties of Conjunctive Queries

In this section we recall some well-known basic properties of CQs that will be using later. Queries in CQ are always satisfiable and monotone and their result set is always finite (and hence is computable). We introduce also the notion of containment among CQs, upon which equivalence is defined. We will devote some lines also to the computational complexity of query answering over relational databases and of query containment, since these two problems provide a baseline for all the other settings of query containment and answering (which are a generalization of this one). These proeprties also hold for UCQs, since they depend on properties true within the existential postive fragment of FOL, to which both classes belong (cf. [ChangKeisler]).

Proposition 1.1. *Let q, q' be* CQs *over* $\mathbf{R}$ *and* $\mathbf{I}, \mathbf{J} \in Inst(\mathbf{R})$. *Then:*

- *If* $\mathbf{I} \subseteq \mathbf{J}$, *then* $q(\mathbf{I}) \subseteq q(\mathbf{J})$. **(Monotonicity)**.
- $q(\mathbf{I})$ *is always finite* **(Finiteness)**.

Proposition 1.2. (Satisfiability) *Let q be a* CQ *over* $\mathbf{R}$*; then there exists* $\mathbf{I} \in Inst(\mathbf{R})$ *s.t.* $q(\mathbf{I}) \neq \emptyset$.

Remark 1.1. Proposition 1.1 (i.e. finiteness) has a very important consequence, namely that, when evaluating a query q over an instance $\mathbf{I}$, we need only to pay attention to those substitutions that range over $adom(\mathbf{I})$ – the finite set of functions ${}^{Var(q)}adom(\mathbf{I})$ of cardinal $\#(adom(\mathbf{I}))^{\#(Var(q))}$ which is polynomial on $\#(adom(\mathbf{I}))$ and exponential on $\#(Var(q))$ (and hence on $size(q)$).

1.3.1 Containment and Equivalence

Containment is a fundamental relation that holds among formal queries, and in particular among CQs (and UCQs). A conjunctive query q over is said to be *contained* by a conjunctive query q, in symbols $q \sqsubseteq q'$, iff for every $\mathbf{I} \in Inst(\mathbf{R})$ it holds that $q(\mathbf{I}) \subseteq q'(\mathbf{I})$. They are said to be *equivalent*, denoted $q \equiv q'$, whenever $q \sqsubseteq q'$ and $q' \sqsubseteq q$. In general, the problem of query containment is typically formulated as follows:

- **Input:** Two arbitrary queries q and q' over a schema $\mathbf{R}$.
- **Question:** Does $q(\mathbf{I}) \subseteq q'(\mathbf{I})$, for all $\mathbf{I} \in Inst(\mathbf{R})$?

As it is well known (cf. [Abiteboul1995]), containment and equivalence are decidable though untractable. Furthermore, the addition of comparisons yields containment and *a fortiori* equivalence even more difficult:

Proposition 1.3. (Complexity of Containment) *Query containment (and hence equivalence) for relational* CQs *is* **NP-Complete** *and is* $\Pi^2_{\mathbf{P}}$**-Complete** *for* CQs *with comparisons.*

1.3.2 Complexity of QA

To study the computational complexity of conjunctive query evaluation we need to consider the decision problem defined as follows: given (i) a tuple $\overline{\boldsymbol{t}}$ over **Dom**, (ii) a CQ q and (iii) an instance $\mathbf{I}$, check whether $\overline{\boldsymbol{t}} \in q(\mathbf{I})$. Following Vardi (cf. [Vardi1982]) we can subdivide this decision problem into three by considering some of its inputs constant:

- When we fix the size of q, we speak about *data complexity*, where computational complexity is a function on $adom(\mathbf{I})$.
- When we fix the size of $\mathbf{I}$, we speak about *query complexity*, where computational complexity is a function on $size(q)$.
- When neither is assumed to be constant, we speak about *combined complexity*, where computational complexity is a function on both $adom(\mathbf{I})$ and $size(q)$.

Remark 1.1 immediately tells us that the first one is tractable and the other two, intracatable. More precisely:

Proposition 1.4. (Complexity of QA) *Query answering for relational* CQs *is in* **LOGSPACE** *w.r.t. data complexity and* **NP-Complete** *w.r.t. both query and combined complexity.*

1.4 Dependencies

Dependencies are ways of imposing some constraints on the possible states that a database may assume, in other words, in the way data is to be structured and are an essential part of database practice as well as of incomplete information settings as we shall soon see. Formally, a *dependency* δ is a first order sentence of the form:

$$\forall \overline{\boldsymbol{x}} \forall \overline{\boldsymbol{y}} (\Phi(\overline{\boldsymbol{x}}) \rightarrow \exists \overline{\boldsymbol{z}} \Psi(\overline{\boldsymbol{y}}, \overline{\boldsymbol{z}}))$$

where $\overline{\boldsymbol{x}}$ and $\overline{\boldsymbol{y}}$ are sequences of variables and $\overline{\boldsymbol{z}}$ is the sequence of variables occuring in $\overline{\boldsymbol{x}}$ but not in $\overline{\boldsymbol{y}}$. Both Φ and Ψ are conjunctions of atoms of which Φ might be possibly empty. Φ and Ψ contain *relational atoms* of the form $R(\overline{\boldsymbol{w}})$ or *equality atoms* of the form $w = w', w \neq w'$. There are many kind of dependencies. For our present purposes we will limit ourselves the following:

- Inclusion dependencies (IDs), which are sentences of the form:

$$\forall \overline{\boldsymbol{x}} \forall \overline{\boldsymbol{y}} \forall \overline{\boldsymbol{z}} (\Phi(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}) \rightarrow \Psi(\overline{\boldsymbol{x}}, \overline{\boldsymbol{z}})).$$

- Functional dependencies (FDs), which are sentences of the form :

$$\forall \overline{\boldsymbol{x}} \forall \overline{\boldsymbol{y}} \forall \overline{\boldsymbol{z}} \forall \overline{\boldsymbol{w}} (\Phi(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}, \overline{\boldsymbol{w}}) \wedge \Phi(\overline{\boldsymbol{x}}, \overline{\boldsymbol{z}}, \overline{\boldsymbol{w}}) \rightarrow \overline{\boldsymbol{y}} = \overline{\boldsymbol{z}}).$$

- Key dependencies (KD), which are sentences of the form:

$$\forall \overline{\boldsymbol{y}} \forall \overline{\boldsymbol{z}} \forall \overline{\boldsymbol{w}} (\Phi(\overline{\boldsymbol{y}}, \overline{\boldsymbol{w}}) \wedge \Phi(\overline{\boldsymbol{z}}, \overline{\boldsymbol{w}}) \rightarrow \overline{\boldsymbol{y}} = \overline{\boldsymbol{z}}).$$

- If Ψ contains no occurrences of $=$ or $\neq$ is ts called a *tuple generating dependency* (TGD).

Note that KDs are a special case of FDs. Dependencies are included in the undecidable (w.r.t. satisfiability) $\forall^* \exists^*$ prefix class of FOL formulas - and IDs in the decidable $\forall^*$ class. A key dependency $\forall \overline{\boldsymbol{y}} \forall \overline{\boldsymbol{z}} \forall \overline{\boldsymbol{w}} (\Phi(\overline{\boldsymbol{y}}, \overline{\boldsymbol{w}}) \wedge \Phi(\overline{\boldsymbol{z}}, \overline{\boldsymbol{w}}) \rightarrow \overline{\boldsymbol{y}} = \overline{\boldsymbol{z}})$ and an inclusion dependency $\forall \overline{\boldsymbol{x}} \forall \overline{\boldsymbol{y}} \forall \overline{\boldsymbol{z}} (\Phi'(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}) \rightarrow \Psi'(\overline{\boldsymbol{x}}, \overline{\boldsymbol{z}}))$ are said to be *non conflicting* whenever $\Psi'(\overline{\boldsymbol{x}}, \overline{\boldsymbol{z}})$ and neither $\Phi(\overline{\boldsymbol{y}}, \overline{\boldsymbol{w}})$ nor $\Phi(\overline{\boldsymbol{z}}, \overline{\boldsymbol{w}})$ coincide. When one puts together conflicting IDs and KDs one gets the so-called *foreign key dependencies* (FKDs). Last, but not least, IDs are required to be TGDs.

Example 1.2. The following are two examples of dependencies over the relational schema $\mathbf{R} := \{Man^1, MarriedMan^1, Loves^2\}$, two IDs and a KD, respectively:

$$\forall x \forall y (Man(x) \rightarrow Loves(x, y)), \tag{1}$$

$$\forall x (MarriedMan(x) \rightarrow Man(x)), \tag{2}$$

$$\forall x \forall y (Man(x) \wedge Man(y) \rightarrow x = y). \tag{3}$$

In (1) as the reader may remark, only the variable x is involved in the inclusion. Note moreover that ID (1) and KD (3) are non conflicting, but that ID (2) and KD (3) are conflicting. As the reader may well see, (1) and (2) are TGDs too.

When dependencies are present, as in the case in database systems, containment has to be relativized to them, which has the effect of rendering the problem computationally more expensive. Two queries q and q' over $\mathbf{R}$ are said to be *contained w.r.t. a set of dependencies* Σ over $\mathbf{R}$, denoted $q \subseteq_\Sigma q'$, iff for every instance $\mathbf{I}$ such that $\mathbf{I} \models \Sigma$ it is the case that $q(\mathbf{I}) \subseteq q'(\mathbf{I})$. This problem is computationally very difficult and, in general, undecidable:

Proposition 1.5. (Containment with Dependencies) *Query containment w.r.t.* IDs *is* **NP-Complete**, *w.r.t. non-conflicting* IDs *and* KDs *it is* **PSPACE-Complete** *and undecidable w.r.t. arbitrary dependencies.*

1.5 Genericity

The property of generitity conveys the idea that the semantics of a query is invariant under certain specific families of mappings (cf. [LibkinC]). Let ϕ: **Dom** $\rightarrow$ **Dom** be a function. The function ϕ can be extended to relation and database instances as follows: $\phi R(\mathbf{I}) := \{\phi(\overline{t})|\overline{t} \in R(\mathbf{I})\}$ and $\phi\mathbf{I} := \{\phi R(\mathbf{I})|R(\mathbf{I}) \in \mathbf{I}\}$. Assume now that $\langle \mathbf{Dom} \leq_{\mathbf{Dom}} \rangle$ is an ordered domain and let ϕ be as above. Let $\mathbf{I}$ be an instance over $\mathbf{R}$ and q a query. Then:

- The query q is said to be *generic* iff ϕ is bijective and $q(\phi\mathbf{I}) = q(\mathbf{I})$.
- The query q is said to be *monotone generic* iff ϕ is strictly monotone increasing w.r.t. $\leq_{\mathbf{Dom}}$ and $q(\phi\mathbf{I}) = q(\mathbf{I})$.
- The query q is said to be *locally generic* iff ϕ is strictly monotone increasing (again, w.r.t. $\leq_{\mathbf{Dom}}$) from $adom(\mathbf{I})$ to **Dom** and $q(\phi\mathbf{I}) = q(\mathbf{I})$.

Proposition 1.6. *Let q, q' be two generic (resp. monotone and locally generic) queries over a schema $\mathbf{R}$ and let $\mathbf{I} \in Inst(\mathbf{R})$. Let furthermore ϕ be a function from* **Dom** *to* **Dom**. *Then $q(\mathbf{I}) = q'(\mathbf{I})$ iff $q(\phi\mathbf{I}) = q'(\phi\mathbf{I})$.*

Proposition 1.7. (Genericity of CQs and UCQs) *Relational* CQs *(and* UCQs*) are generic and* CQs *with comparisons but no constants are monotone generic.*

2 Incomplete Databases

As we said before, DBs contain *complete* information. Which is to say, any fact regarding any piece of data (and by extension all their properties) which is absent from the database is considered as false. Every dependency $\delta \in \Sigma$ must be thus relativised to $adom(\mathbf{I})$, which means that $\mathbf{I}$ must be the unique model of Σ up to isomorphism. This assumption is known as the *closed world assumption* (CWA). Incomplete databases arise when we drop this assumption and stick to the so-called *open world assumption* (OWA). When this happens, we start reasoning about classes of possible database states, namely all of those that, while satisfying the dependencies Σ, extend the (incomplete) database instance $\mathbf{I}$.

We define an *incomplete database* as a pair $\mathbf{K} = \langle \Sigma, \mathbf{I} \rangle$, where Σ is a set of dependencies and $\mathbf{I}$ a database instance. The *active domain* of $\mathbf{K}$, denoted $adom(\mathbf{K})$, is simply the active domain of its incomplete database instance, i.e., $adom(\mathbf{K}) := adom(\mathbf{I})$. Denote $Sen(\mathbf{R})$ the set of all possible FOL sentences (closed formulas) that can be built over schema $\mathbf{R}$. By analogy with the theory of logic databases, we can consider $\mathbf{K}$ as a logical theory over a schema (a FOL signature) $\mathbf{R}$, i.e., $\mathbf{K} \subseteq Sen(\mathbf{R})$. In other words, Σ can be seen as a universally quantified set of axioms or rules, the *intensional database* (IDB), and $\mathbf{I}$ as a set of facts, i.e., of ground atoms, the *extensional database* (EDB), as suggested by Abiteboul et. al. in [Abiteboul1995]. We will be exploiting this similarity frequently, by allowing instance $\mathbf{I}$ to lead a double life as a set of facts, an EDB, and a first order structure.

In this setting query answering is reformulated as a *logical entailment* problem. We must now check if the class of models $Mod(\mathbf{K})$ of a knowledge base $\mathbf{K} = \langle \Sigma, \mathbf{I} \rangle$ are contained (in set theoretical terms) by the models $Mod(q\sigma)$ of the closed query $q\sigma$, viz. whether $\mathbf{K} \models q\sigma$, which is to say, following the definition of entailment in FOL, to check if $Mod(\mathbf{K}) \subseteq Mod(q\sigma)$. This is basically a generalization of QA to deal with an incomplete information setting. Accordingly, the result set $q(\mathbf{K})$ of a UCQ q evaluated over an incomplete database $\mathbf{K}$ is built by collecting the values of all the closed substitutions

$\sigma\colon Var(q) \to \mathbf{Dom}$ under which q is logically entailed by $\mathbf{K}$, viz.,

$$q(\mathbf{K}) := \{\sigma(\overline{\boldsymbol{x}}) | \mathbf{K} \models q\sigma\}.$$

This definition corresponds to the *certain answers* semantics for queries under incomplete information (cf. [Abiteboul1995, Cali2003, Abiteboul1998]), as can e seen from its trivial characterization in terms of the intersections of the semantics of query q over every instance $\mathbf{J}$ containing $\mathbf{I}$ and satisfying Σ.

Example 2.1. Take the schema $\mathbf{R} := \{Loves^2, Man^1, MarriedMan^1\}$. An incomplete database can be defined over this schema by choosing as set of dependencies $\Sigma :=$

$$\begin{gathered}\{\forall x(MarriedMan(x) \to Man(x)),\\ \forall x(Man(x) \to \exists Loves(x,y) \wedge Woman(y)),\\ \forall x(Man(x) \to \neg Woman(x))\}.\end{gathered}$$

and as an incomplete instance the set $\mathbf{I} :=$

$$\begin{gathered}\{MarriedMan(Julian),\\ Man(Lucas),\\ Loves(Julian, Mary),\\ Loves(Lucas, Mary)\}.\end{gathered}$$

To compute $q(\mathbf{K})$, we compute first the answer set of each $\mathbf{J} \models \mathbf{K}$ and then we pick their intersection - a set that need not be necessarily finite and computable:

Proposition 2.1. *Let q be a query over $\mathbf{R}$ and $\mathbf{K} = \langle \Sigma, \mathbf{I}\rangle$ a* KB *over. $\mathbf{R}$. Then it holds that:*

$$q(\mathbf{K}) = \bigcap_{\substack{\mathbf{I}\subseteq\mathbf{J}\\ \mathbf{J}\models\Sigma}} q(\mathbf{J}).$$

But then, *quid* about containment and equivalence? We must redefine these two relations as well to make them fit this new framework. Let $\mathbf{R}$ be a schema. We say that a query q over is *(set) contained* by a query q' over under a incomplete information setting iff for all $\mathbf{K}$ over $\mathbf{R}$ it is the case that $q(\mathbf{K}) \subseteq q'(\mathbf{K})$. However, as the following lemma shows, there is no general increase in complexity when equivalence among queries is defined w.r.t. any arbitrary knowledge base. This only happends when we fix a particular set of dependencies and reason about containment and equivalence w.r.t. that fixed set:

Lemma 2.1. (Reduction to Instance Equivalence) *For every $\mathbf{K} = \langle \Sigma, \mathbf{I}\rangle$, q and q'* CQs *(or* UCQs*) over $\mathbf{R}$, it holds that $q(\mathbf{K}) = q(\mathbf{K})$ iff $q \equiv q'$.*

Proof. ($\Rightarrow$) The assumption holds in particular when $\Sigma = \emptyset$, in which case, by proposition 2.1, we need only to prove that:

$$\bigcap_{\mathbf{I}\subseteq\mathbf{J}} q(\mathbf{J}) = q(\mathbf{I})$$

because any instance $\mathbf{J}$ is a model of Σ, of the empty set. Let $\overline{\boldsymbol{t}} \in \bigcap\{q(\mathbf{J}) | \mathbf{I} \subseteq \mathbf{J}\}$; as $\mathbf{I} \subseteq \mathbf{I}$, $\overline{\boldsymbol{t}} \in q(\mathbf{I})$, i.e. $\bigcap\{q(\mathbf{J}) | \mathbf{I} \subseteq \mathbf{J}\} \subseteq q(\mathbf{I})$. Let $\overline{\boldsymbol{t}} \in q(\mathbf{I})$ and let $\mathbf{J}$ be an instance such that $\mathbf{I} \subseteq \mathbf{J}$. By monotonicity of CQs, $q(\mathbf{I}) \subseteq q(\mathbf{J})$ and therefore $\overline{\boldsymbol{t}} \in q(\mathbf{J})$ for every instance $\mathbf{J}$ s.t. $\mathbf{I} \subseteq \mathbf{J}$, i.e. $q(\mathbf{I}) \subseteq \bigcap\{q(\mathbf{J}) | \mathbf{I} \subseteq \mathbf{J}\}$.

Note that $\mathbf{I}$ in $\langle \emptyset, \mathbf{I} \rangle$ is an arbitrary instance. Hence, by applying the same reasoning to q', we deduce that $q(\mathbf{I}) = q'(\mathbf{I})$, for all $\mathbf{I}$, i.e., $q \equiv q'$.

($\Leftarrow$) Assume that $q \equiv q'$ and let $\mathbf{I}$ be an instance. Then it holds that:

$$q(\mathbf{I}) = q'(\mathbf{I}) \text{ implies } \bigcap_{\mathbf{I} \subseteq \mathbf{J}} q(\mathbf{J}) = \bigcap_{\mathbf{I} \subseteq \mathbf{J}} q'(\mathbf{J}),$$

$$\text{implies } \bigcap_{\substack{\mathbf{I} \subseteq \mathbf{J} \\ \mathbf{J} \models \Sigma}} q(\mathbf{J}) = \bigcap_{\substack{\mathbf{I} \subseteq \mathbf{J} \\ \mathbf{J} \models \Sigma}} q'(\mathbf{J}), \text{ for every } \Sigma.$$

That is (proposition 2.1), $q(\mathbf{K}) = q'(\mathbf{K})$, for every $\mathbf{K}$, as we wanted. □

Remark 2.1. Answer sets may not be, in general, finite, because CQs are now being evaluated against the possibly infinite (even uncountably infinite) family of models of a set of dependencies. This prevents us from reasoning solely in terms of $adom(\mathbf{K})$. Denote, given q and $\mathbf{K}$, $q_a(\mathbf{K})$ the semantics of q where substitutions range not over **Dom**, but over the finite set $adom(\mathbf{K}) \subsetneq \mathbf{Dom}$. Then, clearly $q_a(\mathbf{K}) \subseteq q(\mathbf{K})$, but the converse is false. To see this, consider $\mathbf{K}$ and q defined as follows: put $\Sigma := \{\forall y \exists x \exists z (R(x, y) \rightarrow R(y, z))\}$, $\mathbf{I} := \{R(c_0, c_1)\}$ and $q := q(y) \leftarrow \exists x R(x, y)$. Then $q(\mathbf{K}) = \{\langle c_i \rangle\}_{i \geq 1}$, which is infinite, but $q(\mathbf{K}) = \{\langle c_1 \rangle\}$.

3 Aggregate Queries

In this section we will proceed to define the class of aggregate conjunctive queries and of their unions (CQs + agg and UCQs + agg resp.), which constitute a strict superclass of CQs and UCQs. But, why? For many reasons. Aggregate queries involve operations such as counting the number of a certain collection of records, computing an average, making a sum, etc. These kind of operations become commonplace as soon as the database domain is ordered or contains numerical data. Indeed, empirical studies show that they become very frequent in user questions and queries directed to, say, geographical databases (cf. [Thorne2007D]). An aggregate query in SQL looks as follows:

```
SELECT Man.1, COUNT(Loves.2)
    FROM Man, Loves
    WHERE Man.1 = Loves.1
    GROUP BY Man.1
```

where the GROUPBY clause defines a collection of tuples, the collection of women that each man loves, over which the aggregate function (COUNT) is computed. The attribute Man.1 is known as the grouping attribute. The query is read as "for each man, count the number of people he loves". We can also drop the GROUPBY clause and the grouping attribute and still have an aggregate query, e.g.

```
SELECT COUNT(Loves.2)
    FROM Loves
```

which reads "count all those who are loved by somebody". Indeed, the classes CQs + agg and UCQs + agg of queries correspond, respectively, to the SELECT-FROM-WHERE-GROUP BY-SUM-COUNT-MIN-MAX and to the SELECT-FROM-WHERE-UNION -GROUP BY-SUM-COUNT-MIN-MAX fragments of SQL (cf. [Abiteboul1995, LibkinB, Klug1982A]). As the reader may see, we need to enrich the syntax of CQs to capture this grouping feature together with the aggregate functions themselves. We will do this in the following section.

The main property we will be focusing on will be that of containment and equivalence of aggregate queries. Why? Because we want to know, ultimately, to what extent aggregate query equivalence can be reduced to conjunctive query equivalence, which will provide us with an efficient procedure for evaluating aggregate queries over incomplete data and, more precisely, over knowledge bases. This algorithm, akin to the algorithm defined in [Cali2003], will proceed by equivalence-preserving rewritings that compute a conjunctive query out of the aggregate query, rewrite it w.r.t. the dependencies (following a certain answers-preservings rewriting), evaluate it, and compute the values of the aggregates at the very end.

3.1 Syntax

Since the head of a query may now contain syntactic objects other (and more complex) than variables and constants, we start by defining terms and so-called aggregate terms. A *term* t is either a variable or a constant. An *aggregate* term is an expression of the form $\alpha(\overline{\boldsymbol{y}})$, where $\overline{\boldsymbol{y}}$ is a finite sequence of variables and α is either **sum**, **count**, **min** or **max**, viz., an aggregate function symbol. Given this, we can now define what an aggregate query is: An *aggregate query* (CQ + agg) is a query of the form:

$$q(\overline{\boldsymbol{x}}, \alpha(\overline{\boldsymbol{y}})) \leftarrow \exists \overline{\boldsymbol{z}}(\Psi(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}, \overline{\boldsymbol{z}}) \wedge \Upsilon(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}, \overline{\boldsymbol{z}}))$$

where $\overline{\boldsymbol{x}}$ are called the *grouping variables* and $\overline{\boldsymbol{y}}$, the aggregate variables. We further assume variables (and constants) to be adequately typed, so that they range either over individual constants or numbers (integers, rationals or reals). Moreover, the sequence $\overline{\boldsymbol{y}}$ is also allowed to occur alone in the head of q. We define analogously *unions of aggregate queries* (UCQs + agg). The notions of *arity*, *size*, *boolean queries*, etc., are defined in a way analogous to that of conjunctive queries.

The conjunctive query that underlies an aggregate query can be obtained in a very simple way by stripping it from its aggregate function symbol. We call this underlying conjunctive query the *core* of the aggregate query q and denote it $\breve{q}$: it is, formally, the conjunctive query obtained from q by replacing its aggregate term $\alpha(\overline{\boldsymbol{y}})$ by $\overline{\boldsymbol{y}}$ (i.e. by the aggregate variables).

Example 3.1. The following are some examples of (relational) conjunctive queries, unions of conjunctive queries and aggregate (**count**) queries:

$$q_1(x, y) \leftarrow Man(x) \wedge Loves(x, y)$$
$$q_2(x, y) \leftarrow Loves(y, x)$$
$$q_3(x, \mathbf{count}(y)) \leftarrow Man(x) \wedge Loves(x, y)$$
$$q_4(x, \mathbf{count}(y)) \leftarrow Loves(y, x)$$
$$q_5(x, y) \leftarrow (Man(x) \wedge Loves(x, y)) \vee Loves(y, x)$$
$$q_6(x, \mathbf{count}(y)) \leftarrow (Man(x) \wedge Loves(x, y)) \vee Loves(y, x)$$

As the reader may see q_1 is the core of q_3, q_2 the core of q_4 and q_5 is a UCQ built out of q_1 and q_2 (and the core of q_6).

3.2 Aggregate Terms

For each UCQ q and each incomplete database $\mathbf{K}$ we define an equivalence relation $\sim_q$ over the substitutions $\sigma\colon Var(q) \to adom(\mathbf{K})$ as follows: $\sigma \sim_q \sigma'$ iff $\sigma(x) = \sigma'(x)$, that is, whenever two substitutions map the grouping variable(s) onto the same constant(s). We denote $[\sigma]_q$ the equivalence class represented by σ and $Var(q)/_{\sim_q}$ the resulting quotient set. Equivalence classes intuitively collect

the groups associated to the values of each grouping variable. The equivalence relation $\sim_q$ gives rise, for every substitution σ over the grouping variable(s) of an aggregate query q to the following set:

$$\{\theta(y)|\theta \in [\sigma]_q\}.$$

Given a grouping assignment σ, we can define the *evaluation* of an aggregate term over a group $[\sigma]_q$, denoted $\alpha(y).[\sigma]_q$, for $\alpha \in \{\mathbf{sum}, \mathbf{count}, \mathbf{min}, \mathbf{max}\}$, as follows:

$$\mathbf{max}(y).[\sigma]_q := \max_{\theta \in [\sigma]_q} \theta(y)$$
$$\mathbf{min}(y).[\sigma]_q := \min_{\theta \in [\sigma]_q} \theta(y)$$
$$\mathbf{count}(y).[\sigma]_q := \#(\{\theta(y)|\theta \in [\sigma]_q\})$$
$$\mathbf{sum}(y).[\sigma]_q := \sum_{\theta \in [\sigma]_q} \theta(y)$$

Remark 3.1. One can also view semantically aggregate functions as set-valued functions of the form $\#\colon \mathcal{P}_f(\mathbf{Dom}) \to \mathbb{N}$ or $\alpha\colon \mathcal{P}_f(\mathbb{Q}) \to \mathbb{Q}$, where $\alpha \in \{\sum, \prod, \min, \max\}$ and $\mathcal{P}_f(A)$ denotes the set of finite subsets of a set A. Under this particular assumption, we can deduce some immediate properties. Note that by convention, $\mathbb{Q} \subsetneq \mathbf{Dom}$ (i.e., **Dom** is a set of numeric values and individual constants). Let $\{a_i\}_{i\in[0,n]}$ and $\{b_j\}_{j\in[0,m]}$ be two finite families of pairwise disjoint finite subsets of, respectively, $\mathbb{R}$ and **Dom**. Then it holds that:

$$\sum \bigcup_{i=0}^{n} a_i = \sum a_1 + ... + \sum a_n,$$
$$\prod \bigcup_{i=0}^{n} a_i = \prod a_1 * ... * \prod a_n,$$
$$\max \bigcup_{i=0}^{n} a_i = \max\{\max a_1, ..., \max a_n\},$$
$$\min \bigcup_{i=0}^{n} a_i = \min\{\min a_1, ..., \min a_n\} \text{ and}$$
$$\#(\bigcup_{j=0}^{m} b_j) = \#(b_1) + ... + \#(b_m).$$

3.3 Semantics, Containment and Equivalence

Aggregate queries are run over numerical data and ordered domains. That is to say, that they are not evaluated exclusively over a set **Dom** of individual constants, but also over totally ordered numerical domains - i.e., over **Dom** and $\mathbb{Q}$. We may further assume that $\mathbb{Q}$ has been given the structure of a ordered field (i.e., the ordered field $\mathcal{Q} = \langle \mathbb{Q}; \leq_{\mathbb{Q}}; +_{\mathbb{Q}}, \cdot_{\mathbb{Q}}, {}^{-1}\mathbb{R}; 0_{\mathbb{Q}}, 1_{\mathbb{Q}} \rangle$ of the rationals), as is common in everyday mathematics, and similarly for $\mathcal{N}$ or $\mathcal{Z}$. Thus, numerical variables will range over $\mathbb{Q}$ (or its subsets) and individual variables over **Dom** proper.

The *result set* for CQs + agg (resp. UCQs + agg) is defined in a manner analogous to that of CQs (resp. UCQs) , only we need to exploit the semantics of aggregate terms:

$$q(\mathbf{K}) := \{(\sigma(\overline{\boldsymbol{x}}), \alpha(\overline{\boldsymbol{y}}).[\sigma]_q)|\mathbf{K} \models \breve{q}\sigma)\}$$

where $(\sigma(\overline{\boldsymbol{x}}), \alpha(\overline{\boldsymbol{y}}).[\sigma]_q) \in adom(\mathbf{K})^n \times \mathbb{Q}$, for some $n \in \mathbb{N}$, is the concatenation of $\sigma(\overline{\boldsymbol{x}})$ and $\alpha(\overline{\boldsymbol{y}}).[\sigma]_q$. Two queries are said to be *comparable* iff they are of the same arity and agree on their distinguished variables. In the case of aggregate queries this means that they agree on all of their grouping and aggregate variables as well as on their aggregate terms.

Remark 3.2. The reader may have noticed that we are using a variant of the *certain answers* semantics with sound sources, as defined for conjunctive queries w.r.t. incomplete databases (cf. [Cali2004, Cali2003]). That is, given q and $\mathbf{K}$ over $\mathbf{R}$ we compute first the certain answers for $\breve{q}(\mathbf{K})$ and only afterwards, $[\sigma]_q$ and the value of the aggregate function (whether **sum**, **prod**, **count**,**min** or **max**). We notice that the certain answers semantics is perhaps one of the few, if not the only one that makes sense for conjunctive queries in this setting. Suppose we did otherwise, i.e., that we were to compute answer set of the aggregate query q itself over each $\mathbf{J}$ such that $\mathbf{J} \models \mathbf{K}$ and only after compute the intersection, instead of starting (as in our definition with $\breve{q}$. Such $\mathbf{J}$s are finite structures over a countably infinite universe **Dom** s.t. $\mathbb{Q} \subseteq \mathbf{Dom}$. As there are countably many such instances, we can enumerate them, i.e., collect them into a family $\{\mathbf{J}_i\}_{i\in\mathbb{N}}$. But then it would hold that for some $i, j \in \mathbb{N}$, $q(\mathbf{J}_i) \neq q(\mathbf{J}_j)$, whence $q(\mathbf{J}_i) \cap q(\mathbf{J}_j) = \emptyset$. Therefore, $q(\mathbf{K}) = \emptyset$, since for every set A, $A \cap \emptyset = \emptyset$ (the null set is absorbant for set intersection), and this would make no sense: all agreggate queries would be unsatisfiable.

Given this, we can now state the properties that interests us: containment and equivalence. We say that two comparable aggregate queries q, q' over a schema $\mathbf{R}$ are *contained under set semantics*, denoted $q \sqsubseteq_s q'$, iff for every incomplete database $\mathbf{K}$ over $\mathbf{R}$ we have that:

$$q(\mathbf{K}) \subseteq q'(\mathbf{K}).$$

Equivalence, denoted $\equiv_s$, is defined as the symmetric closure of $\sqsubseteq_s$.

Example 3.2. When we evaluate the queries from example 3.1 over the incomplete database of example 2.1 we get:

$$\begin{gathered}
q_1(\mathbf{K}) = \{\langle Lucas, Mary\rangle, \langle John, Mary\rangle\},\\
q_2(\mathbf{K}) = \{\langle Mary, John\rangle, \langle Mary, Lucas\rangle\},\\
q_3(\mathbf{K}) = \{\langle Lucas, 1\rangle, \langle John, 1\rangle\},\\
q_4(\mathbf{K}) = \{\langle Mary, 2\rangle\},\\
q_5(\mathbf{K}) = \{\langle Lucas, Mary\rangle, \langle John, Mary\rangle,\\
\langle Mary, John\rangle, \langle Mary, Lucas\rangle\}.\\
q_6(\mathbf{K}) = \{\langle Lucas, 1\rangle, \langle John, 1\rangle, \langle Mary, 2\rangle\}.
\end{gathered}$$

In the case of q_3, for instance, consider the substitutions $\sigma_1 := \{x \mapsto Lucas\}$ and $\sigma_2 := \{x \mapsto John\}$. If we look carefully at $adom(\mathbf{K})$ we can see that $\{\theta(y) | \theta \in [\sigma_1]_{q_3}\} = \{\theta'(y) | \theta' \in [\sigma_2]_{q_3}\} = \{Mary\}$, whence $\mathbf{count}(y).[\sigma_1]_{q_3} = \mathbf{count}(y).[\sigma_2]_{q_3} = 1$. Notice that $q_5(\mathbf{K}) = q_1(\mathbf{K}) \cup q_2(\mathbf{K})$ as one should expect.

Remark 3.3. Note that the following (strict) inclusions hold: $\mathsf{CQs} \subsetneq \mathsf{UCQs} \subsetneq \mathsf{UCQs} + \mathsf{agg}$ and $\mathsf{CQs} \subsetneq \mathsf{CQs} + \mathsf{agg}$.

4 Equivalence of Aggregate Queries

In this section state results regarding the reduction of the equivalence of aggregate queries to that of their cores (under set semantics). Core equivalence constitutes a sufficent but not necessary condition of equivalence. However, it becomes a necessary condition when we restrict ourselves to purely relational queries (i.e. without comparisons) or to relational queries with comparisons but no constants. In both cases we will exploit the genericity of their cores. This problem (namely, equivalence) has been thoroughly studied in particular by Cohen *et al.* in [Cohen2007] and by Chaudhuri and Vardi in [Vardi1993] for the so-called bag and bag-set semantics. As the reader may recall, commercial database engines and SQL itself rely on bags or multisets as main data-structures. Intuitively, a bag is like a set, only that it admits repetitions. Database tables and result sets are bags. In a bag-set semantics, tables become again, set-valued relations, but result sets, or, more exactly, groups, become bags. That is, instead of an assigment σ over the variables of an aggregate query q giving way to a set $\{\theta(y)|\theta \in [\sigma]_q\}$, it gives rise to a multiset $\{\!\{\theta(y)|\theta \in [\sigma]_q\}\!\}$. However, when dealing with incomplete databases and, more importantly, with knowledge bases (which can be considered a particular case of the former), we have to manipulate sets and set-valued relations.

Lemma 4.1. (Sufficiency of core equivalence) *Let q and q' be two relational aggregate queries with or without constants. Then $\breve{q} \equiv \breve{q}'$ implies $q \equiv_s q'$.*

Proof. We will show the result only for **sum** queries, the other cases are analogous. Let q, q' be two **sum** queries whose cores are equivalent. Then, for all $\mathbf{I}$:

$$\breve{q}(\mathbf{I}) = \breve{q}'(\mathbf{I}). \tag{4}$$

Assume w.l.o.g that $\{x\}$ is the set of grouping variables and $\{y\}$ that of aggregate variables. We have to prove that $q(\mathbf{K}) \subseteq q'(\mathbf{K})$ and $q'(\mathbf{K}) \subseteq q(\mathbf{K})$. We will only prove the first claim, the other being symmetrical. We claim that for every tuple $\langle c, n\rangle \in adom(\mathbf{K}) \times \mathbb{Q}$ with $\sigma(x) = c$ and $\sum_{\theta\in[\sigma]_q} \theta(y) = n$ and every substitution σ over $Var(q)$ s.t. $\mathbf{K} \models \breve{q}\sigma$ there exists a substitution σ' over $Var(q')$ s.t. $\mathbf{K} \models \breve{q}\sigma'$ and:

$$\langle \sigma(x), \sum_{\theta\in[\sigma]_q} \theta(y)\rangle = \langle \sigma'(x), \sum_{\theta'\in[\sigma']_{q'}} \theta'(y)\rangle,$$

which entails $q(\mathbf{K}) \subseteq q'(\mathbf{K})$. Let σ be a substitution over $Var(q)$ s.t. $\mathbf{K} \models \breve{q}\sigma$. Let $\mathbf{J} \models \mathbf{K}$, then $\mathbf{J}$ is an instance such that $\mathbf{J} \models \breve{q}\sigma$ and by (4) there exists a substitution σ' over $Var(q')$ that coincides with σ over x, that is, such that $\sigma(x) = \sigma'(x)$. Furthermore, since $\mathbf{J}$ was arbitrary, $\mathbf{K} \models \breve{q}'\sigma'$. We thus only need to show that:

$$\sum_{\theta\in[\sigma]_q} \theta(y) = \sum_{\theta'\in[\sigma']_{q'}} \theta'(y),$$

which is true whenever

$$\{\theta(y)|\theta \in [\sigma]_q\} = \{\theta'(y)|\theta' \in [\sigma']_{q'}\}.$$

As before, there are two symmetrical cases to consider, so we will prove only the first. Let $\theta \in [\sigma]_q$. We claim that there exists a $\theta' \in [\sigma']_q$ such that $\theta'(y) = \theta(y)$. Put $\theta' = \theta \restriction \{x, y\}$, i.e. θ' is the restriction of θ to $\{x, y\}$ and hence a substitution over $Var(q')$ too since $\{x, y\} \subseteq Var(q) \cap Var(q')$. Clearly, by definition, $\theta(y) = \theta'(y)$. Furthermore, $\sigma'(x) = \sigma(x) = \theta(x) = \theta'(x)$, whence $\theta' \in [\sigma']_{q'}$. This closes the proof. □

Cohen *et al.* in [Cohen2007] proved that for relational **max**, **min** or **count** queries, equivalence (under set semantics) is equivalent to core equivalence:

	Rel	Rel + Cons	Rel + $\leq$	Rel + $\leq$ + Cons
max(**min**)	Yes	Yes	Yes	No
count	Yes	Yes	Yes	No
sum	Yes	Yes	Yes	No
prod	Yes	Yes	Yes	No

Figure 1: reduction to core equivalence

Proposition 4.1. (Cohen *et. al.*) *Let q, q' be two comparable relational* **max** *(or* **min***) queries. Then $q \equiv_s q'$ if $\breve{q} \equiv \breve{q}'$.*

Proposition 4.2. (Cohen *et al.*) *Let q, q' be two comparable relational* **count** *queries. Then $q \equiv_s q$ if $\breve{q} \equiv \breve{q}'$.*

4.1 Queries with Comparisons but no Constants

We start by looking at the necessary conditions for equivalence of relational aggregate queries with numerical comparisons. The argument is based on the monotonic genericity and the local genericity of their cores, plus some algebraic properties of numerical domains. Note that by lemma 2.1 reasoning over incomplete databases (under OWA) can be reduced to reasoning over database instances (under CWA), when equivalence is at stake.

Lemma 4.2. *Let q, q' be two comparable relational* **sum** *queries with comparisons but without constants. Then $q \equiv_s q$ implies $\breve{q} \equiv \breve{q}'$.*

Proof. The proof is similar to the proof by Cohen *et al.* in [Cohen2007]. We reason by contraposition. Let q and q' be two comparable **sum** queries over a signature $\mathbf{R}$ and suppose that $\breve{q} \not\equiv \breve{q}'$, that is, that there exists some instance $\mathbf{I}$ over $\mathbf{R}$ such that $\breve{q}(\mathbf{I}) \neq \breve{q}'(\mathbf{I})$. Assume, moreover, that $\{x\}$ is the set of grouping variables $\{y\}$ and that of aggregate variables (i.e. q and q' are of arity $n = 2$). This means, w.l.o.g. that there is some tuple $\langle c, n\rangle \in adom(\mathbf{I}) \times adom(\mathbf{I})$ and some substitution σ over $Var(q)$ with $\sigma(x) = c, \sigma(y) = n$, such that $\langle c, n\rangle \in \breve{q}(\mathbf{I})$ but $\langle c, n\rangle \notin \breve{q}'(\mathbf{I})$. Now, the latter is true when, and only when, for every σ' over $Var(q')$ we have that:

(i) $\sigma(x) \neq \sigma'(x)$, or

(ii) $\sigma(x) = \sigma'(x)$ and $\sigma(y) \neq \sigma'(y)$.

If (i) holds, then, clearly, $q \not\equiv_s q'$. Otherwise, if (ii) holds, we have to show that

$$\sum_{\theta \in [\sigma]_q} \theta(y) \neq \sum_{\theta' \in [\sigma']_{q'}} \theta'(y) \tag{5}$$

for every substitution σ' over $Var(q')$ that coincides with σ over x. Suppose towards contradiction that the two sums coincide for some σ' over $Var(q')$. By (ii) we know that that

$$n \notin \{\theta'(y) | \theta' \in [\sigma']_{q'}\}. \tag{6}$$

	Rel + Cons	Rel + $\leq$
max(min)	**NP-Complete**	$\mathbf{\Pi^2_P}$**-Complete**
count	**NP-Complete**	$\mathbf{\Pi^2_P}$**-Complete**
sum	**NP-Complete**	$\mathbf{\Pi^2_P}$**-Complete**
prod	**NP-Complete**	$\mathbf{\Pi^2_P}$**-Complete**

Figure 2: complexity of equivalence

To derive an absurdity we exploit the local genericity of conjunctive queries (cf. [LibkinC]) over ordered domains. Let $m_0 < m_1 < ... < m_k$ be a chain of the numerical constants of $adom(\mathbf{I})$, with $n = m_{i_0}$, for some $i_0 \in [0, k]$. Let $M := m_k$ and define ϕ: $adom(\mathbf{I}) \to \mathbf{Dom}$ by putting

$$\phi(m_i) := M^i.$$

Recall that we are assuming that the sums of q and q' should agree for the group of c. This will be true whenever:

$$\sum_{i=0}^{k} \mu_i M^i = \sum_{i=0}^{k} \mu'_i M^i,$$

where $\mu_i, \mu'_i \in \{0, 1\}$, for $i \in [0, k]$, that is, their M-adic representations should agree. Moreover, by construction (i.e. by properties of M-adic representations of numbers), $\mu_i = \mu'_i$, for all $i \in [0, n]$. Now, $\theta(y) = n$, for some $\theta \in \sigma$, whence $\mu_{i_0} = \mu'_{i_0} = 1$. But then $\theta'(y) = n$ too, for some $\theta' \in [\sigma']$, which contradicts (6). Hence (5) should hold too. □

Lemma 4.3. *Let q, q' be two comparable relational* **prod** *queries without constants but with comparisons. Then $q \equiv_s q$ implies $\breve{q} \equiv \breve{q}'$.*

Proof. The proof is similar to that of **sum** queries, so we will omit some of the details. As before, We will assume that $\breve{q} \not\equiv \breve{q}'$, and hence that there exists an instance $\mathbf{I}$, a substitution σ over $Var(q)$ mapping x to c and y to n satisfying the same conditions as before. We need to define now a monotone function ϕ: $adom(\mathbf{I}) \to \mathbf{Dom}$ on the basis of some function ψ: $\mathbb{Q} \to \mathbb{Z}$. Let again So, let $m_0 < m_1 < ... < m_k$ be a chain of the numerical constants of $adom(\mathbf{I})$. As before, put $M := m_k$. Functions ψ and ϕ are defined by:

$$\psi(m_i) := M^i;$$
$$\phi(m_i) := p_i^{\psi(m_i)}.$$

Where p_i is the i-est prime number. As the reader may check, ϕ is monotone increasing. Moreover, the equivalence of the agreggate queries implies that

$$\prod_{i=0}^{k} p_i^{\mu_i \cdot \psi(m_i)} = \prod_{i=0}^{k} p_i^{\mu'_i \cdot \psi(m_i)}$$

with $\mu_i, \mu'_i \in \{0, 1\}$, for $i \in [0, k]$. Which, by the uniqueness of prime factorisation, implies that $\mu_i = \mu'_i$, for all $i \in [0, k]$. The argument then proceeds as before. □

Remark 4.1. Note that any finite sequence $\langle q_1, ..., q_k \rangle$ of rationals can be monotonically mapped by an order preserving function f onto a sequence $\langle f(q_1), ..., f(q_k) \rangle$ of integers. Rationals can be seen as

pairs of integers, i.e., $q_i = \frac{a_i}{b_i}$, with $a_i, b_i \in \mathbb{Z}$, for $i \in [1,k]$. Define $f\colon [\mathbb{Q}]^k \to [\mathbb{Z}]^k$, where $[A]^k$ denotes the set of finite sequences of length k of elements of set A, by putting:

$$f(q_i) := q_i \cdot \prod_{i=1}^{k} b_i.$$

Then, clearly, $q_1 \leq_{\mathbb{Q}} ... \leq_{\mathbb{Q}} q_k$ only if $f(q_1) \leq_{\mathbb{Z}} ... \leq_{\mathbb{Z}} f(q_k)$.

4.2 Relational Queries

Now we turn to relational queries. We will strongly rely on their genericity.

Proposition 4.3. *For any two finite sets $\Gamma := \{m_1, ..., m_k\} \subseteq \mathbb{Q}$ and $\Delta := \{n_1, ..., n_k\} \subseteq \mathbb{Q}$ of the same size, such that $\Gamma \neq \Delta$ and such that $\alpha\Gamma = \alpha\Delta$, for $\alpha \in \{\sum, \prod\}$, there exists a bijection $\phi\colon \mathbb{Q} \to \mathbb{Q}$ s.t.*

- $\{\phi(m_1), ..., \phi(m_k)\} \neq \{\phi(n_1), ..., \phi(n_k)\}$ *and*
- $\alpha\{\phi(m_1), ..., \phi(m_k)\} \neq \alpha\{\phi(n_1), ..., \phi(n_k)\}$.

Proof. Let $\Gamma := \{m_1, ..., m_k\}$ and $\Delta := \{n_1, ..., n_k\}$ be two finite sets of rationals of the same size s.t. $\alpha\Gamma = q \in \mathbb{Q}$, for $\alpha \in \{\sum, \prod\}$ and define $\phi\colon \mathbb{Q} \to \mathbb{Q}$ as follows:

$$\phi(m) := \begin{cases} q & \text{if } m = m_1, \\ m_1 & \text{if } m = q, \\ m & \text{otherwise.} \end{cases}$$

Clearly, ϕ is a bijection, $\{\phi(m_1), ..., \phi(m_k)\} \neq \{\phi(n_1), ..., \phi(n_k)\}$ and, finally, $\alpha\{\phi(m_1), ..., \phi(m_k)\} \neq \alpha\{\phi(n_1), ..., \phi(n_k)\}$ as wanted. □

Lemma 4.4. *Let q, q' be two comparable relational* **sum** *or* **prod** *queries with or without constants. Then $q \equiv_s q$ implies $\breve{q} \equiv \breve{q}'$.*

Proof. Let q and q' be two aggregate queries over a schema $\mathbf{R}$. Assume that $q \equiv q'$ and suppose for contradiction (see lemma 2.1) that for some $\mathbf{I} \in Inst(\mathbf{R}), \breve{q}(\mathbf{I}) \neq \breve{q}'(\mathbf{I})$. This means that there should be some σ over $Var(q)$ s.t. for every σ' over $Var(q')$, $\sigma(x) = \sigma'(x)$ and

$$\underset{\theta \in [\sigma]_q}{\alpha} \theta(y) = \underset{\theta' \in [\sigma']_{q'}}{\alpha} \theta'(y) \tag{7}$$

with $\alpha \in \{\sum, \prod\}$ and

$$\{\theta(y) | \theta \in [\sigma]_q\} \neq \{\theta'(y) | \theta' \in [\sigma']_{q'}\}. \tag{8}$$

We need to consider two cases:

- Either $\#(\{\theta(y) | \theta \in [\sigma]_q\}) > \#(\{\theta(y)' | \theta' \in [\sigma']_{q'}\})$. Recall that $\breve{q}, \breve{q}'$ are CQs, they are generic. Define a mapping $\phi\colon$ **Dom** $\to$ **Dom** by putting:

$$\phi(m) := \begin{cases} m+1 & \text{if } m \in \mathbb{Q}, \\ m & \text{otherwise.} \end{cases}$$

We can see that ϕ is a bijection and that

$$\underset{\theta\in[\sigma]_q}{\alpha}\phi(\theta(y)) > \underset{\theta'\in[\sigma']_{q'}}{\alpha}\phi(\theta'(y)),$$

whence $q(\langle\emptyset,\phi\mathbf{I}\rangle) \neq q'(\langle\emptyset,\phi\mathbf{I}\rangle)$, which is absurd, since $\langle\emptyset,\phi\mathbf{I}\rangle$ is an incomplete database over $\mathbf{R}$ and q and q' are, by hypothesis, equivalent.

- Or $\#(\{\theta(y)|\theta \in [\sigma]_q\}) = \#(\{\theta(y)'|\theta' \in [\sigma']_{q'}\})$. Now, equations (7) and (8) together with proposition 4.3 imply the existence of a bijection ϕ: **Dom** $\rightarrow$ **Dom** (which is the identity over the non-numeric constants of **Dom**) such that

$$\underset{\theta\in[\sigma]_q}{\alpha}\phi(\theta(y)) \neq \underset{\theta'\in[\sigma']_{q'}}{\alpha}\phi(\theta'(y)).$$

As before, by genericity, it follows that $q(\langle\emptyset,\phi\mathbf{I}\rangle) \neq q'(\langle\emptyset,\phi\mathbf{I}\rangle)$. Contradiction. □

4.3 Reduction to Core Equivalence

From lemmas 4.1 - 4.4 and propositions 4.1 - 4.2 we immediately derive the following:

Theorem 4.1. (Reduction to core equivalence) *Equivalence of aggregate* **sum**, **count**, **prod** **min** *and* **max** *under set semantics is reducible to the equivalence (under, again, set semantics) of their cores.*

Furthermore, by considering the linearizations or linear expansions of conjunctive queries, as defined by Cohen *et al.* in [Cohen2007], we can immediately derive the complexity of equivalence for aggregate queries:

Corollary 4.1. *The equivalence problem fore relational aggregate* **sum**, **count**, **prod min** *and* **max** *queries under set semantics is* **NP-Complete** *and* $\Pi^2_{\mathbf{P}}$**-Complete** *for queries with comparisons.*

Remark 4.2. As the reader may recall, many different combinations of numerical values may add up to the same number and two different groups might be equipotent. Similar considerations hold for the maximum or the minimum of ordered sets. However, without constants or comparisons we cannot constrain values such that this may happen. It is only when comparisons and constants are both allowed to occur in the body of aggregate queries that these reductions do not hold, in general, anymore. Consider the queries:

$$q(\mathbf{sum}(x)) \leftarrow A(1) \wedge A(2) \wedge A(3) \wedge 1 \leq x \wedge x < 3, \text{ and}$$
$$q'(\mathbf{sum}(x)) \leftarrow A(1) \wedge A(2) \wedge A(3) \wedge x = 3.$$

Take $\mathbf{K} := \langle\emptyset, \{A(1), A(2), A(3)\}\rangle$ and suppose that x ranges over $\mathbb{Z}$. Then, clearly, $q \equiv_s q'$ but $\breve{q}(\mathbf{I}) \neq \breve{q}'(\mathbf{I})$, since $\breve{q}(\mathbf{I}) = \{1, 2\}$, whereas $\breve{q}'(\mathbf{I}) = \{3\}$. That is, aggregate queries can be equivalent and their cores not.

5 QA with Incomplete Information

In this section we will address the last remaining issue, namely, that of the complexity of the query answering (QA) for conjunctive queries, which is the decision problem stated as follows:

- **Input:** A query q, an incomplete database $\mathbf{K} = \langle \Sigma, \mathbf{I} \rangle$ and a tuple $\overline{\boldsymbol{t}}$ over **Dom**.
- **Question:** Does $\mathbf{K} \models q[\overline{\boldsymbol{t}}/\overline{\boldsymbol{x}}]$?

As we said, nothing warrants that the semantics of a query in an incomplete (or OWA) setting is a finite set, unless we impose some conditions on incomplete databases, namely on their set of dependencies. Given a CQ q and $\mathbf{K}$, it has been shown (cf. [Klug1982B, Cali2003]) that when we restrict Σ to IDs, FDs or to non-conflicting IDs and KDs, then $q(\langle \Sigma, \mathbf{I} \rangle)$ is a finite set and can be therefore computed (even if not efficiently). Furthermore, there exists a query q_Σ, called the *unfolding* of q, s.t. $q \equiv q_\Sigma$. This unfolding q_Σ is obtained by a procedure known as the *chase* that we will not describe, referring the reader instead to [Abiteboul1995, Klug1982B]. It consists, intuitively, in rewriting the query q w.r.t. the IDB relations of $\mathbf{K}$, i.e. the relations R occurring in Σ but not in $\mathbf{I}$ (or in $\mathbf{R}$). The size $size(q_\Sigma)$ of q_Σ might be exponential on $size(q)$, but this does not affect data complexity.

5.1 Relational Queries

Proposition 5.1. *Let $\mathbf{K} = \langle \Sigma, \mathbf{I} \rangle$ be an incomplete database over $\mathbf{R}$ with Σ a set of non-conflicting* IDs *and* KDs*,* IDs *or* FDs*, q a* UCQ *over $\mathbf{R}$ and $\overline{\boldsymbol{t}}$ a tuple over* **Dom**. *Then the following holds:*

$$\begin{aligned} \langle \Sigma, \mathbf{I} \rangle \models q[\overline{\boldsymbol{t}}/\overline{\boldsymbol{x}}] \ \textit{iff}\ \overline{\boldsymbol{t}} &\in q(\langle \Sigma, \mathbf{I} \rangle), \\ \textit{iff}\ \overline{\boldsymbol{t}} &\in \bigcap_{\substack{\mathbf{I} \subseteq \mathbf{J} \\ \mathbf{J} \models \Sigma}} q(\mathbf{J}), \\ \textit{iff}\ \overline{\boldsymbol{t}} &\in q_\Sigma(\mathbf{I}). \end{aligned}$$

Based on this proposition we can conceive a sound, complete and terminating algorithm $\textsc{Answer}_{\mathbf{K}}$ (cf. [Cali2003]) for conjunctive query QA:

Algorithm 1 $\textsc{Answer}_{\mathbf{K}}$

```
1: procedure ANSWER_K(q, t, ⟨Σ, I⟩)
2:     if t ∈ q_Σ(I) then
3:         return TRUE;
4:     else
5:         return FALSE;
6:     end if
7: end procedure
```

This algorithm runs in time polynomial on $\#(adom(\mathbf{K}))$ and in space polynomial both on $size(q)$ and on $size(q)$ and $\#(adom(\mathbf{K}))$. More precisely (cf. [Cali2003]):

Proposition 5.2. (Complexity of QA) QA *over incomplete databases $\mathbf{K} = \langle \Sigma, \mathbf{I} \rangle$ for relational* CQs *is:*

- *In* **P** *on data complexity and* **PSPACE-Complete** *on both query and combined complexity when is a set of* IDs*.*
- *In* **P** *on data complexity if Σ is a set of* FDs*.*
- *In* **P** *on data complexity and* **PSPACE-Complete** *on both query and combined complexity when is a set of non conflicting* KDs *and* IDs*.*

5.2 Queries with Comparisons

As we said before, aggregate functions are defined on the power set of *ordered numerical domains*. Database active domains contain now a finite number of numerical values (such as integers, rationals or reals). So too are their cores. But evaluating an aggregate query does not dramatically increase the computational data complexity of QA. Given a group $[\sigma]_q$ of size m for a CQ + agg q and computed over an instance I, computing the sum or the product will be of complexity $O(m)$ and computing either the maximum or the minimum, of complexity $O(m \cdot log_2 \cdot m)$ (the complexity of sorting a table or array). The question is then, what is the complexity of incomplete QA over an ordered domain, when q is a CQ *with comparisons* (and constants)? In the case of relational databases, where CWA holds, it will be in **LOGSPACE**, since q lies within FOL. Dependencies plus OWA turn the issue more delicate. In order to consider this, we must first fix what we understand by an ordering. That is, when a relational structure $\mathcal{A} = \langle \mathcal{A}, \mathcal{R}^{\mathcal{A}} \rangle$ over a FOL signature (with identity) containing the diadic predicate symbol R and where $R^{\mathcal{A}} \subseteq A \times A$ is called an *order*? When it satisfies any of the following axioms:

- A *strict partial order* or, simply, an *order* (O) is defined by the axioms:

$$\forall x \neg R(x,x) \text{ (\textbf{irreflexivity})}$$
$$\forall x \forall y \forall z (R(x,y) \wedge R(y,z) \rightarrow R(x,z)) \text{ (\textbf{transitivity})}$$

- A *total* (T) relation by the axiom:

$$\forall x \forall y (R(x,y) \vee R(x,y) \vee x = y) \text{ (\textbf{tricotomy})}$$

- A *dense* (Den) relation by:

$$\forall x \forall y (R(x,y) \rightarrow \exists z (R(x,z) \wedge R(z,y))$$

- A *Discrete partial order* (Dis) is defined by the axioms:

$$\forall x \exists y (R(x,y) \vee x = y)$$
$$\forall x \exists y (R(x,y) \vee x = y) \wedge \forall z (R(x,z) \rightarrow R(y,z) \vee z = y))$$
$$\forall x (\forall y (R(x,y) \vee x = y) \vee \exists z (R(x,z) \vee x = z) \wedge$$
$$\wedge \forall y (R(z,y) \vee z = y) \rightarrow R(x,y) \vee x = y \vee y = z)$$

- A relation has a *least element* (L) if it satisfies:

$$\exists x \forall y R(x,y)$$

- A relation has a *greatest element* (G) if it satisfies:

$$\exists y \forall y R(x,y)$$

- A *partial order* (PO) is defined by the axioms:

$$\forall x R(x,x) \text{ (\textbf{reflexivity})}$$
$$\forall x \forall y (R(x,y) \wedge R(y,x) \rightarrow x = y) \text{ (\textbf{antisymmetry})}$$
$$\forall x \forall y \forall z (R(x,y) \wedge R(y,z) \rightarrow R(x,z)) \text{ (\textbf{transitivity})}$$

Remark 5.1. Note that, actually, we need only one kind of basic ordering. That is, POs can be built or defined out of Os (and vice versa), provided our FOL signature contains the identity symbol. If R' denotes a PO and R an O, we just need to add the (definitional) axiom $\forall x \forall y(R'(x,y) \leftrightarrow (R(x,y) \vee x = y))$. Note, furthermore, that POs are trivially dense, i.e., PO $\models$ Den.

These axioms can be combined together into different theories, characterizing different kinds of orderings like *total orders* (TO), *total dense orders* (TODen), *orders with enpoints* (OLG), etc. The following theories lie within the $\forall^*$ fragment of FOL, and hence satisfy the *finite model property* (FMP) that is, when they have a model, they have a finite model: Dis, TDis, O, TO, PO, TPO, DenPO, TDenPO. Furthermore, since skolemezing L or G introduces only a constant (denoting the least or, resp., the greatest element of the ordering, which must be, moreover, unique), this means that DisL, DisG, DisLG, OL, OG, OLG, POL, POG, POLG, DenPOL, DenPOG and DenPOLG (together with their associated total orders) satisfy too, *modulo* skolemization, FMP. This classification of orderings is important in that it characterizes the kind of orderings we may impose on the (ordered) domain of an incomplete database $\mathbf{K}$. We may thus assume implicitly that, for $\mathbf{K} = \langle \mathbf{I}, \Sigma \rangle$ and T a theory of order, $T \subseteq \Sigma$. Hence, if we restrict T to the above theories, we can, when looking at incomplete database states or instances, focus on those that are finite (if always of an arbitrary finite size).

Theorem 5.1. *Incomplete* QA *for relational* CQs *with comparisons is* **Co-NP-Hard** *in data complexity.*

Proof. The proof is by reduction of the complement of the $2+2$-SAT problem for propositional logic, which is known to be **Co-NP-Complete** and a *a fortiori* **Co-NP-Hard**. Let Φ be a $2+2$ clause. In that case, $\Phi = C_1 \wedge ... \wedge C_n$, with $C_i = p_{i1} \vee p_{i2} \vee \neg p_{i1} \vee \neg p_{i2}$, for $i \in [1,n]$. We encode Φ into an incomplete database $\mathbf{K}_\Phi$ and a query q_Φ. Remember that we always assume **Dom** to be totally ordered by $\leq_{\mathbf{Dom}}$. Note that we are interested in unsatisfiable formulas, which means that by hypothesis, for some $i \in [1,n]$, the clause $p_{i1} \vee p_{i2} \vee \neg p_{i1} \vee \neg p_{i2}$ in Φ is unsatisifiable (i.e. has no model or truth value assignment under which it evaluates to the truth). The reduction proceeds as follows:

- For each clause C_i in Φ, we map positive litterals p_{ij} to a fact $P(c_i, p^+_{ij})$ and negative litterals to a fact $N(c_i, p^-_{ij})$, for $i \in [1,n], j \in \{0,1\}$.
- We map formula Φ to the incomplete database $\mathbf{K}_\Phi := \langle \Sigma_\Phi, \mathbf{I}_\Phi \rangle$, where Σ_Φ contains (at most) the axioms of total order (with or without endpoints) and $\mathbf{I}_\Phi$ is the incomplete instance

$$\{P(c_1, p^+_{11}), P(c_1, p^+_{12}), N(c_1, p^-_{11}), N(c_1, p^-_{12}),$$
$$\vdots$$
$$P(c_n, p^+_{n1}), P(c_n, p^+_{n2}), N(c_n, p^-_{n1}), N(c_n, p^-_{n2}), Z(c)\},$$

  and to the (constant) conjunctive query q_Φ with comparisons

$$q() \leftarrow \exists x,y,z,w,t,s(P(x,y) \wedge y \leq s \wedge P(x,z) \wedge z \leq s \wedge$$
$$\wedge N(w,w) \wedge w > s \wedge N(x,t) \wedge t > s \wedge Z(s)).$$

Computing $\mathbf{K}_\Phi$ and q_Φ is polynomial on the number of litterals of Φ, which is $k \leq 4n$. Furthermore, the only input of the entailment problem whose size depends on k is $\mathbf{I}_\Phi$. Therefore, we are reasoning on the *data complexity* of QA. We now claim that

$$\Phi \text{ is unsatisfiable iff } \mathbf{K}_\Phi \models q_\Phi.$$

Ordering $T \subseteq \Sigma$	Rel + $\leq$ + IDs
TO (TOLG,TOL,TOG)	**Co-NP-Complete**
TDis (TDisLG,TDisL,TDisG)	**Co-NP-Complete**
TPO (TPOLG,TPOL,TPOG)	**Co-NP-Complete**

Figure 3: data complexity of incomplete QA for CQs with comparisons

($\Rightarrow$) Suppose that $\mathbf{K}_\Phi \not\models q_\Phi$. We want to show that Φ is satisfiable. We know that $\mathbf{K}_\Phi \not\models q_\Phi$ iff there exists an instance $\mathbf{J} \in Mod(\mathbf{K}_\Phi)$ s.t. $q_\Phi(\mathbf{J}) = \emptyset$. Now, since $\mathbf{I}_\Phi \subseteq \mathbf{J}$, this means that in $\mathbf{J}$ we find all the atoms of $\mathbf{I}_\Phi$, like, e.g. $P(c_1, p^+_{11})$, $N(c_1, p^-_{11})$, etc. On the other hand, $q_\Phi(\mathbf{J}) = \emptyset$, i.e. $\mathbf{J} \not\models q_\Phi$, implies that, forall $\sigma\colon Var(q) \to \mathbf{Dom}$ with $\sigma(s) = 0$, either $\mathbf{J} \models \sigma(p_1) > \sigma(s)$ or $\mathbf{J} \models \sigma(p_2) > \sigma(s)$ or $\mathbf{J} \models \sigma(p_3) \leq \sigma(s)$ or $\mathbf{J} \models \sigma(p_4) \leq \sigma(s)$. On the basis of this, we define an assignment $v\colon \mathsf{LITT} \to \{0, 1\}$ by putting

$$v(p) := \begin{cases} 1 & \text{if } \{\sigma(p) >_{\mathbf{Dom}} c\} \subseteq \mathbf{J}, \\ 0 & \text{otherwise.} \end{cases}$$

Where LITT denotes the family $\{p_i\}_{i\in\mathbb{N}}$ of propositional litterals. Clearly, for all $i \in [1, n]$, it holds that $v(p_{i1} \vee p_{i2} \vee \neg p_{i1} \vee \neg p_{i2}) = 1$, i.e., $v(\Phi) = 1$.

($\Leftarrow$) Assume that Φ is satisfiable. We want to build an instance $\mathbf{J}$ such that $\mathbf{J} \models \mathbf{K}_\Phi$ and $\mathbf{J} \not\models q_\Phi$. If Φ is satisfiable this means that there exists a truth value assignment $v\colon \mathsf{LITT} \to \{0, 1\}$ such that, for all $i \in [1, n]$, $v(C_i) = 1$. Instance $\mathbf{J}$ is the instance over $\mathbf{R}$ where, for $i \in [1, n]$, $j \in \{0, 1\}$,:

$$\{P(c_i, p^+_{ij})\} \subseteq \mathbf{J}, \text{ when } v(p_{ij}) = 1,$$
$$\{N(c_i, p^-_{ij})\} \subseteq \mathbf{J}, \text{ when } v(\neg p_{ij}) = 1,$$
$$\{p_{ij} >_{\mathbf{Dom}} c\} \subseteq \mathbf{J}, \text{ when } v(p_{ij}) = 1 \text{ and}$$
$$\{Z(c)\} \subseteq \mathbf{J}.$$

Clearly, $\mathbf{J} \models \langle \Sigma_\Phi, \mathbf{I}_\Phi \rangle$, but $q_\Phi(\mathbf{J}) = \emptyset$. Otherwise, suppose that $\mathbf{J} \models q_\Phi$. If this were true, it would hold that, for some substitution $\sigma\colon Var(q) \to \mathbf{Dom}$ with $\{x \mapsto c_1, s \mapsto 0, y \mapsto p^+_{11}\} \subseteq \sigma$, $\mathbf{J} \models P(\sigma(x), \sigma(y))$ and $\mathbf{J} \models \sigma(y) \leq \sigma(s)$. But if this were the case, then, by definition of $\mathbf{J}$ it would follow that $v(p^+_{ij}) = 1$ and that therefore $\mathbf{J} \models \sigma(y) > \sigma(s)$. Contradiction. □

Remark 5.2. We assumed $\langle \mathbf{Dom} \leq_{\mathbf{Dom}} \rangle$ to be a total order, i.e. such that $\langle \mathbf{Dom} \leq_{\mathbf{Dom}} \rangle \models \mathsf{T}$ (tricotomy). But the same argument would hold for ordered sets of numbers with or without endpoints, such as $\langle \mathbb{N}, \leq_{\mathbb{N}} \rangle$, $\langle \mathbb{Z}, \leq_{\mathbb{Z}} \rangle$, $\langle \mathbb{Q}, \leq_{\mathbb{Q}} \rangle$, $\langle \mathbb{R}, \leq_{\mathbb{R}} \rangle$, or any totally ordered subset (whether finite or infinite) thereof. Indeed, the constant c above "behaves" like 0, by partitioning the ordering (which is linear and can be thus pictured as a line). That is, for all $c' \in \mathbf{Dom}$, either $c' \leq_{\mathbf{Dom}} c$ or $c' >_{\mathbf{Dom}} c$. This partitioning in its turn mimics the behaviour of negation, thus yielding an exponential blowup when evaluating the query over an incomplete database.

Lemma 5.1. *Incomplete* QA *for relational* CQs *with comparisons and* IDs *is in* **Co-NP** *in data complexity.*

Proof. (Sketch) Assume that q is constant, that Σ contains only IDs and that $T \subseteq \Sigma$ is a universal theory of order, that is: TO (TOLG,TOL,TOG), TDis (TDisLG,TDisL,TDisG) or TPO (TPOLG,TPOL,TPOG).

	Rel	$\leq$
max(min)	**P**	**Co-NP-Hard**
count	**P**	**Co-NP-Hard**
sum	**P**	**Co-NP-Hard**
prod	**P**	**Co-NP-Hard**

Figure 4: data complexity of incomplete QA for aggregate queries

We know that:

$$\mathbf{K} \not\models q[\overline{\boldsymbol{t}}/\overline{\boldsymbol{x}}] \text{ iff } \overline{\boldsymbol{t}} \notin \bigcap_{\substack{\mathbf{I}\subseteq\mathbf{J} \\ \mathbf{J}\models\Sigma}} q(\mathbf{J}),$$

that is, iff $\overline{\boldsymbol{t}} \notin q(\mathbf{J})$ for some $\mathbf{J} \models \langle \Sigma, \mathbf{I} \rangle$. Now Σ contains only universal sentences. Hence it satisfies FMP. In particular, its class of finite models coincides with the set of its Herbrand models. Denote $\mathbf{J}_H$ the greatest Herbrand model of $\mathbf{K}$. Then the set $\{\mathbf{J} \subseteq \mathbf{J}_H | \mathbf{J} \models \langle \Sigma, \mathbf{I} \rangle\}$ is finite. Choose such an instance $\mathbf{J}$. Compute then $q(\mathbf{J})$, which can be done in time polynomial on $\#(adom(\mathbf{K}))$. Finally, check whether $\overline{\boldsymbol{t}} \notin q(\mathbf{J})$, which can be done, again, in time polynomial on $\#(adom(\mathbf{K}))$. □

From lemma 5.1 and theorem 5.2 we thus get:

Theorem 5.2. (Complexity of QA) *Incomplete* QA *for relational* CQs *with comparisons,* IDs *and universal theories of total order is* **Co-NP-Complete** *in data complexity.*

5.3 Aggregate Queries

As we already said aggregate functions have no biog impact on the data complexity of QA. Given a group $[\sigma]_q$ of size m for a CQ + agg q and computed over an instance I, computing the sum or the product will be of complexity $O(m)$ and computing either the maximum or the minimum, of, say, complexity $O(m \cdot log_2 \cdot m)$ (the complexity of sorting a table or array). This means that the complexity results holding for CQs can be easily generalized to cover this broader class of queries. Furthermore, since, by definition, aggregate functions are computed over the certain answers of their cores, we can almost immediately define an algorithm for QA, algorithm 2 ($\text{ANSWER}^{\alpha}_{\mathbf{K}}$) below, based on algorithm 1 ($\text{ANSWER}_{\mathbf{K}}$, see above).

Algorithm 2 $\text{ANSWER}^{\alpha}_{\mathbf{K}}$

1: **procedure** $\text{ANSWER}^{\alpha}_{\mathbf{K}}(q, (\sigma(\overline{\boldsymbol{x}}), \alpha_{\theta\in[\sigma]_q}\theta(\overline{\boldsymbol{y}})), \langle \Sigma, \mathbf{I} \rangle)$
2: $\quad \Gamma \leftarrow \{(\sigma(\overline{\boldsymbol{x}}), \theta(\overline{\boldsymbol{y}})) | \theta \in [\sigma]_q\}$;
3: $\quad Val \leftarrow \text{TRUE}$;
4: $\quad$ **for** $\overline{\boldsymbol{t}} \in \Gamma$ **do**
5: $\quad\quad Val \leftarrow Val$ **and** $\text{ANSWER}_{\mathbf{K}}(\breve{q}, \overline{\boldsymbol{t}}, \langle \Sigma, \mathbf{I} \rangle)$;
6: $\quad$ **end for**
7: $\quad$ **return** Val
8: **end procedure**

Proposition 5.3. *The algorithm* $\text{ANSWER}^{\alpha}_{\mathbf{K}}$ *is sound, complete and terminating. Furthermore, it runs in time polynomial on* $\#(adom(\mathbf{K}))$.

From theorem 5.2 and propositions 5.3 and 5.2 we immediately derive the following:

Theorem 5.3. (Complexity of QA) *Incomplete* QA *for* CQs + agg *is:*

- **Co-NP-Hard** *in data complexity for queries with comparisons.*
- *In* **P** *for relational queries.*

Conclusions

Conjunctive aggregate queries (CQs + aggs) are *grosso modo* SELECT - FROM - WHERE - SUM - COUNT - MIN - MAX - GROUP BY queries in SQL (to which we add the aggregate function **prod**, which behaves in a way similar to **sum**) and ranging this time not only over non-numerical domains but also over the ordered domain of the rational numbers $\mathbb{Q}$ and its subsets. We have considered the following cases for conjunctive aggregate queries, namely queries that are (i) only relational, (ii) contain numerical constants, (iii) contain numerical comparisons and (iv) contain both comparisons and numerical constants. Furthermore, we have extended the existing results for relational databases to an incomplete information setting. In doing so, we have achieved the following:

- We have defined a semantics for evaluating aggregate queries in an incomplete information setting. It is based on the certain answers semantics for CQs and relies on the well-known notion of the core of an aggregate query q (i.e. its underlying CQ $\breve{q}$). We have argued why this semantics makes sense.
- We have studied the computational complexity of the problems of query equivalence, QC and QA for aggregate queries under this incomplete information setting.
- In particular, we have reduced the equivalence problem of aggregate queries to that of their (conjunctive) cores, which holds for cases (i), (ii) and (iii) above.
- Finally, we have looked at the lower and upper computational complexity bounds of QA w.r.t. all the possible orderings that we can define or conceive over **Dom**.

As immediate further work we intend to look at ways of expressing such queries in natural language and, specifically, in controlled English, in a manner similar to [Thorne2007B] (i.e. extending Lite English). Aggregations are expressed by definite noun phrases (NPs) such as "the sum of all N", "the greatest N", "the product of all N", etc, where N stands for a plural or singular (common) noun. But is not clear how these would interplay with the other natural language components so as to compositionally translate questions into these formal queries.

References

[Abiteboul1995] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Welsey, 1995.

[Abiteboul1998] Serge Abiteboul and Oliver Dushka. Complexity of Answering Queries Using Materialized Views. In *PODS '98: Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1998.

[Cali2003] Andrea Cali, Domenico Lembo, and Ricardo Rosati. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Information. In *PODS '03: Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database systems*, 2003.

[Cali2004] Andrea Cali, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Data Integration under Integrity Constraints. *Information Systems*, (29), 2004.

[Calvanese2007D] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Domenico Lembo, Antonella Poggi, and Riccardo Rosati. MASTRO-I: Efficient Integration of Relational Data through DL Ontologies. In *Proceedings of the 20th International Workshop on Description Logics (DL 07)*, 2007.

[ChangKeisler] Chen Chung Chang and Howard Jerome Keisler. *Model Theory*. Elsevier, 1990.

[Clifford1988] James Clifford. Natural Language Querying of Historical Databases. *Computational Linguistics*, 14:10–35, 1988.

[Cohen2007] Sara Cohen, Wernet Nutt, and Yeshoshua Sagiv. Deciding Equivalences among Aggregate Queries. *Journal of the ACM*, (5), 2007.

[Elmasri2004] Ramez Elmasri and Shamkant B. Navanthe. *Fundamentals of Database Systems*. Addison Welsey, 2004.

[Fuchs] Norbert E. Fuchs, Kaarel Kaljurand, and Gerold Schneider. Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces. http://www.ifi.unizh.ch/attempto/publications, 2005.

[Klug1982A] Anthony Klug. Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. *Journal of the ACM*, (3), 1982.

[Klug1982B] Anthony Klug and David Johnson. Testing Containment of Conjunctive Queries Under Functional and Inclusion Dependencies. In *PODS '82: Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, 1982.

[LibkinB] Leonid Libkin. Expressive Power of SQL. *Theoretical computer Science*, (3), 2003.

[LibkinC] Michael Benedikt, Guozhu Dong, Leonid Libkin, and Limson Wong. Relational Expressive Power of Constraint Query Languages. *Journal of the ACM*, (1), 1998.

[Rosati2007] Riccardo Rosati. The Limits of Querying Ontologies. In *Proceedings of the Eleventh International Conference on Database Theory (ICDT 2007)*, 2007.

[Thorne2007B] Raffaella Bernardi, Diego Calvanese, and Camilo Thorne. Lite Natural Language. In *Proceedings of the 7th International Workshop on Computational Semantics (IWCS-7)*, 2007.

[Thorne2007D] Raffaella Bernardi, Diego Calvanese, Camilo Thorne, Francesca Bonin, and Domenico Carbotta. English Querying over Ontologies: E-QuOnto. 2007.

[Thorne2007E] Camilo Thorne. Managing Structured Data with Controlled English - An Approach Based on Description Logics. 2007.

[Vardi1982] Moshe Vardi. The Complexity of Relational Query Languages. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, 1982.

[Vardi1993] Surajit Chaudhuri and Moshe Vardi. Optimization of Real Conjunctive Queries. In *PODS '93: Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART symposium on Principles of Database Systems*, 1993.