International Journal of Approximate Reasoning ••• (••••) •••–•••

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

First-order under-approximations of consistent query answers ☆

Floris Geerts [a], Fabian Pijcke [b], Jef Wijsen [b,*]

[a] University of Antwerp, Dept. of Mathematics and Computer Science, Middelheimlaan 1, B-2020 Antwerpen, Belgium
[b] Université de Mons, 20 Place du Parc, B-7000 Mons, Belgium

ARTICLE INFO

Article history:
Received 19 February 2016
Received in revised form 23 September 2016
Accepted 19 October 2016
Available online xxxx

Keywords:
Conjunctive queries
Consistent query answering
Primary key

ABSTRACT

Consistent Query Answering (CQA) is a principled approach for answering queries on inconsistent databases. The consistent answer to a query q on an inconsistent database **db** is the intersection of the answers to q on all repairs, where a repair is any consistent database that is maximally close to **db**. Unfortunately, computing consistent answers under primary key constraints has already exponential data complexity for very simple conjunctive queries, and is therefore completely impracticable.

In this paper, we propose a new framework for divulging an inconsistent database to end users, which adopts two postulates. The first postulate complies with CQA and states that inconsistencies should never be divulged to end users. Therefore, end users should only get consistent query answers. The second postulate states that only those queries can be answered whose consistent answers can be obtained with low data complexity (i.e., by a polynomial-time algorithm or even a first-order logic query). User queries that exhibit a higher data complexity will be rejected.

A significant problem in this framework is as follows: given a rejected query, find other queries, called under-approximations, that are accepted and whose consistent answers are contained in those of the rejected query. We provide solutions to this problem for the special case where the constraints are primary keys and the queries are self-join-free conjunctive queries.

1. Introduction

Inconsistent, incomplete and uncertain data is widespread in the internet and social media era. This has given rise to a new paradigm for query answering, called *Consistent Query Answering* (CQA) [2]. This paradigm starts with the notion of *repair*, which is a new consistent database that minimally differs from the original inconsistent database. In general, an inconsistent database can have many repairs. In this respect, database repairing is different from data cleaning which aims at a unique cleaned database.

In this paper, we assume that the only constraints are primary keys, one per relation. A repair of an inconsistent database **db** is a maximal subset of **db** that satisfies all primary key constraints. Primary keys will be underlined. For example, the database of Fig. 1 stores ages and cities of residence of male and female persons. For simplicity, assume that persons have

☆ A short version of this article was published in the conference proceedings of SUM 2015 [1].

* Corresponding author.
E-mail addresses: floris.geerts@ua.ac.be (F. Geerts), fabian.pijcke@umons.ac.be (F. Pijcke), jef.wijsen@umons.ac.be (J. Wijsen).

http://dx.doi.org/10.1016/j.ijar.2016.10.005
0888-613X/

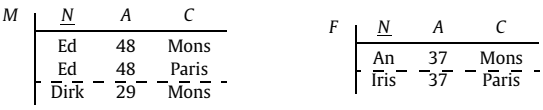

Fig. 1. Example database with primary key violations.

unique names (attribute N). Every person has exactly one age (attribute A) and city (attribute C). However, distinct tuples may agree on the primary key N, because there can be uncertainty about ages and cities. In the database of Fig. 1, there is uncertainty about the city of Ed (it can be Mons or Paris). The database can be repaired in two ways: delete either $M(\underline{\text{Ed}}, 48, \text{Mons})$ or $M(\underline{\text{Ed}}, 48, \text{Paris})$. A maximal set of tuples that agree on their primary key will be called a *block*; in Fig. 1, blocks are separated by dashed lines.

When database repairing results in multiple repairs, CQA shifts from standard semantics to certainty semantics. Given a query, the *consistent answer* (also called *certain answer*) is defined as the intersection of the answers on all repairs. That is, for a query q on an inconsistent database **db**, CQA replaces the standard query answer $q(\mathbf{db})$ with the consistent answer, defined by the following intersection:

$$\bigcap \Big\{ q(\mathbf{r}) \mid \mathbf{r} \text{ is a repair of } \mathbf{db} \Big\}. \tag{1}$$

Thus, the certainty semantics exclusively returns answers that hold true in every repair. Given a query q, we will denote by $\lfloor q \rfloor$ the query that maps a database to the consistent answer defined by (1).

A practical obstacle to CQA is that the shift to certainty semantics involves a significant increase in complexity. When we refer to complexity in this paper, we mean data complexity, i.e., the complexity in terms of the size of the database (for a fixed query) [3, p. 422]. It is known for long [4] that there exist conjunctive queries q that join two relations such that the data complexity of $\lfloor q \rfloor$ is already **coNP**-hard. If this happens, CQA is completely impracticable.

This paper investigates ways to circumvent the high data complexity of CQA in a realistic setting, which is based on the following assumptions:

- If a query returns an answer to a user, then every tuple in that answer should belong to the consistent answer. In Libkin's terminology [5], query answers must not contain *false positives*, i.e., tuples that do not belong to the consistent answer.
- The only queries that can be executed in practice are those with data complexity in **FP** or, even better, in **FO**. Here, **FO** refers to the descriptive complexity class that captures all queries expressible in relational calculus [6]. **FP** is the class of function problems solvable in polynomial time.

Therefore, if the data complexity of a query $\lfloor q \rfloor$ is not in **FP**, then the best we can go for is an approximation without false positives (also called under-approximation), computable in polynomial time. The term *strategy* will be used for queries that compute such approximations. Intuitively, a strategy can be regarded as a two-step process in which one starts by issuing a number of well-behaved queries $\lfloor q_i \rfloor$, for $i \in \{1, \ldots, \ell\}$, which can then be subject to a post-processing step. In this paper, well-behaved queries are those that are accepted by a query interface, e.g., self-join-free conjunctive queries q_i such that $\lfloor q_i \rfloor$ is in **FO**, and post-processing is formalized as queries built-up from the $\lfloor q_i \rfloor$'s.

We next illustrate our setting by an example. Consider the following scenario with two persons, called *Bob* and *Alice*. The person called *Bob* owns a database that is publicly accessible only via a query interface which restricts the syntax of the queries that can be asked. Our main results concern the case where the interface is restricted to self-join-free conjunctive queries. The database schema including all primary key constraints is publicly available. However, *Bob* is aware that his database contains many mistakes which should not be divulged. Therefore, whenever some end user asks a query q, *Bob* will actually execute the query $\lfloor q \rfloor$. That is, end users will get exclusively consistent answers. But, for feasibility reasons, *Bob* will reject any query q for which the data complexity of $\lfloor q \rfloor$ is too high. In this paper, we assume that *Bob* considers that data complexity is too high when it is not in **FO**. The person called *Alice* interrogates *Bob*'s database, and she will be happy to get exclusively consistent answers. Unfortunately, her query q will be rejected by *Bob* if the data complexity of $\lfloor q \rfloor$ is too high (i.e., not in **FO**). If this happens, *Alice* has to change strategy. Instead of asking q, she can ask a finite number of queries $q_1, q_2, \ldots, q_\ell$ such that for every $i \in \{1, \ldots, \ell\}$, the data complexity of $\lfloor q_i \rfloor$ is in **FO**, and hence the query q_i will be accepted by *Bob*. No restriction is imposed on the number ℓ of queries that can be asked. The best *Alice* can hope for is that she can compute herself the answer to $\lfloor q \rfloor$ (or even to q) from *Bob*'s answers to $\lfloor q_1 \rfloor, \ldots, \lfloor q_\ell \rfloor$ by means of some post-processing. The question addressed in this paper is: Given that *Alice* wants to answer q, what queries should she ask to *Bob*?

Here is a concrete example. Assume *Bob* owns the database of Fig. 1. Interested in stable couples,[1] *Alice* submits the query q_1 which asks "Get pairs of ages of men and women living in the same city":

$$q_1 = \big\{ y, w \mid \exists x \exists u \exists z \left(M(\underline{x}, y, z) \wedge F(\underline{u}, w, z) \right) \big\}.$$

[1] According to [7], marital stability is higher when the wife is 5+ years younger than her husband.

Please cite this article in press as: F. Geerts et al., First-order under-approximations of consistent query answers, Int. J. Approx. Reason. (2016), http://dx.doi.org/10.1016/j.ijar.2016.10.005