Artificial Intelligence 232 (2016) 1–19

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Certain answers as objects and knowledge ☆

Leonid Libkin

School of Informatics, University of Edinburgh, United Kingdom

ARTICLE INFO

Article history:
Received 5 May 2015
Received in revised form 17 November 2015
Accepted 27 November 2015
Available online 2 December 2015

Keywords:
Incomplete information
Database queries
Certain answers
Data models
Certain knowledge
Open and closed world
Efficient computation

ABSTRACT

The standard way of answering queries over incomplete databases is to compute *certain answers*, defined as the intersection of query answers on all complete databases that the incomplete database represents. But is this universally accepted definition correct? We argue that this "one-size-fits-all" definition can often lead to counterintuitive or just plain wrong results, and propose an alternative framework for defining certain answers.

The idea of the framework is to move away from the standard, in the database literature, assumption that query results be given in the form of a database object, and to allow instead two alternative representations of answers: as objects defining all other answers, or as knowledge we can deduce with certainty about all such answers. We show that the latter is often easier to achieve than the former, that in general certain answers need not be defined as intersection, and may well contain missing values in them. We also show that with a proper choice of semantics, we can often reduce computing certain answers – as either objects or knowledge – to standard query evaluation. We describe the framework in the most general way, applicable to a variety of data models, and test it on three concrete relational semantics of incompleteness: open, closed, and weak closed world.

1. Introduction

Handling incomplete information is one of the oldest topics in data management research. It has been tackled both from the database perspective, resulting in classical notions of the semantics and complexity of query evaluation [1,2], and from the AI perspective, providing an alternative view of the problem, see, e.g., [3,4]. With the shifting focus in database applications, owing to the ever increasing amounts of data as well as data heterogeneity, the problem of incomplete information is becoming much more pronounced. It appears in many important application areas, particularly those where techniques from both the data management community and the knowledge representation community have been heavily used. These include data integration [5], data exchange [6], ontology-based data access [7,8], inconsistent databases [9], probabilistic data [10], and data quality [11]. Here we treat the notion of "incompleteness" rather broadly: it means that data at our disposal does not provide a complete description, but rather suggests a number – perhaps infinite – of possible worlds. Most of the development here will need just this intuition, but in concrete examples we shall use a common scenario of relational databases with missing (null) values [2,3].

The central problem in all applications of incomplete databases is query answering. In the presence of incompleteness, one normally looks for *certain answers*: those that do not depend on the interpretation of unknown data. The concept

☆ This paper is an invited revision of a paper which first appeared at the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR 2014).

E-mail address: libkin@inf.ed.ac.uk.

http://dx.doi.org/10.1016/j.artint.2015.11.004
0004-3702/

was first mentioned in [12] and formally defined 35 years ago in [13] as follows. Assume that the semantics $[\![D]\!]$ of an incomplete database D is given as a set of complete databases, or possible worlds, D'. These are databases which the incomplete D can represent. Then the certain answer to Q on D is defined as

$$\mathsf{cert}_\cap(Q, D) = \bigcap\{Q(D') \mid D' \in [\![D]\!]\}.$$

Since database queries produce collections (sets, multisets, etc.), it makes sense to talk about their intersection. This definition has been universally applied to all the semantics of incompleteness, and in all the scenarios such as those listed above. The intuition is that this gives us the set of tuples independent of the interpretation of the data that was not completely specified.

If D is a relational, or an XML, or a graph database, $[\![D]\!]$ is usually obtained by replacing nulls (missing values) with real values, and perhaps adding extra information, such as tuples that were not in the database. In other applications, the definition of $[\![D]\!]$ varies, but the notion of certain answers does not. We now give a few examples.

Data integration. Here D is a source database (in fact it may contain multiple sources) and we need to answer queries over an integrated global schema database. The integration process is guided by a schema mapping M relating the schema of the source with that of the global schema over which queries must be answered. The process is virtual: the integrated database is not built, as M rarely makes it unique. Instead, D and M together provide an incomplete description of the integrated database, i.e.,

$$[\![D]\!]_M = \{D' \text{ of the global schema} \mid D \text{ and } D' \text{ satisfy } M\}.$$

If a query Q is posed against the global schema database, and we have access to the source D, query answers are typically defined as certain answers, i.e., $\bigcap\{Q(D') \mid D' \in [\![D]\!]_M\}$, see [14,5].

Data exchange. This scenario is similar to the previous one: we have a schema mapping M between two schemas (usually called source and target schemas in this context), and still want to compute certain answers defined just as above. The difference is that this time the target schema database is materialized, and certain answers must be found based on a specific instance containing data exchanged between the source and the target, see [6,15].

Consistent query answering. Assume we have a database D and a set Σ of integrity constraints over it, such as keys, foreign keys, etc, that D is supposed to satisfy. An inconsistent database fails to satisfy Σ, so one looks for its repairs D'; those are smallest changes that restore consistency (these can be defined in a variety of ways [9]). Then $[\![D]\!]_\Sigma$ consists of all such repairs, and for a given query Q one looks for consistent query answers defined as $\bigcap\{Q(D') \mid D' \in [\![D]\!]_\Sigma\}$. These are query answers that are true in all the repairs that restore consistency.

Ontology based query answering. An ontology Θ provides additional information about an incomplete database. Together, a database D and an ontology Θ define a set $[\![D]\!]_\Theta$ of possible worlds that are completions of the database D that make it satisfy all the ontology constraints. Ontology-based query answering – a very active research theme as of late – boils down to taking a query Q and finding certain answers to it over $[\![D]\!]_\Theta$, see [7,8,16].

Thus, in all these applications, spanning a large spectrum of data management and knowledge representation tasks, certain answers are the standard way of defining answers to queries. While the exact semantics of possible worlds changes, the definition of certain answers stays intact.

The question that we address here is the following: *Is this standard "one-size-fits-all" definition really the right one to use for all the semantics, and all the applications?* The answer, as we shall argue, is negative: the standard intersection semantics leads to many problems, and crucially to producing meaningless query answers.

To argue that this is the case, and to explain basic ideas behind the approach we present, note that in the database field, one tends to operate with *objects* (i.e., relations, XML documents, graph databases, etc.). In particular, queries take objects and return objects. Thus, the idea behind certain answers is to find an *object* A representing the *set of objects* $Q([\![D]\!]) = \{Q(D') \mid D' \in [\![D]\!]\}$. Such an object A must contain information common to all the objects in $Q([\![D]\!])$: that is, it must be no more informative than any of the objects in $Q([\![D]\!])$. The definition of $\mathsf{cert}_\cap$ uses the intersection operator as a means of extracting such common information.

Now take a simple example: we have a relation $R = \{(1,2),(3,\bot)\}$ in a database D, where $\bot$ represents a null, or a missing value. The query Q returns R itself. Then $\mathsf{cert}_\cap(Q, D) = \{(1,2)\}$ under every reasonable semantics of incompleteness. But is it less informative than all of $Q(D')$ for $D' \in [\![D]\!]$? The answer depends on the semantics. Under the common *open-world* semantics, the answer is yes: in fact $(1,2)$ is precisely the greatest lower bound of $Q([\![D]\!])$ under the ordering whose meaning is "being less informative". But under the equally common *closed-world* semantics, the answer is no. Even worse, $(1,2)$ is not less informative than any of the answers $Q(D')$ for $D' \in [\![D]\!]$ which are of the form $\{(1,2),(3,n)\}$ for different values n. Indeed, under the closed world semantics, the answer $\{(1,2)\}$ contains *additional* information that no tuple except $(1,2)$ is present. Thus, returning just $(1,2)$ in this case makes no sense at all.

The problem with returning the single tuple $(1,2)$ as the certain answer becomes even more pronounced if we follow the approach, pioneered by [3], that views databases as logical theories and query answering as logical implication. The fact $R(1,2)$ is certainly implied by the database. But is it the only fact that is implied? Of course not: under the open-world semantics, we can deduce $\exists x\, R(1,2) \wedge R(3,x)$ with certainty, adding the fact that there is a tuple whose first component is 3.