Incomplete Information and Certain Answers in General Data Models

Leonid Libkin
School of Informatics, University of Edinburgh
libkin@inf.ed.ac.uk

ABSTRACT

While incomplete information is ubiquitous in all data models – especially in applications involving data translation or integration – our understanding of it is still not completely satisfactory. For example, even such a basic notion as certain answers for XML queries was only introduced recently, and in a way seemingly rather different from relational certain answers.

The goal of this paper is to introduce a general approach to handling incompleteness, and to test its applicability in known data models such as relations and documents. The approach is based on representing degrees of incompleteness via semantics-based orderings on database objects. We use it to both obtain new results on incompleteness and to explain some previously observed phenomena. Specifically we show that certain answers for relational and XML queries are two instances of the same general concept; we describe structural properties behind the naïve evaluation of queries; answer open questions on the existence of certain answers in the XML setting; and show that previously studied ordering-based approaches were only adequate for SQL's primitive view of nulls. We define a general setting that subsumes relations and documents to help us explain in a uniform way how to compute certain answers, and when good solutions can be found in data exchange. We also look at the complexity of common problems related to incompleteness, and generalize several results from relational and XML contexts.

Categories and Subject Descriptors. H.2.1 [**Database Management**]: Logical Design—*Data Models*; I.7.2 [**Document and Text Processing**]: Document Preparation—*XML*

General Terms. Theory, Languages, Algorithms

Keywords. Incompleteness, naive tables/evaluation, certain answers, XML, orderings, homomorphisms

PODS'11, June 13–15, 2011, Athens, Greece.

1. INTRODUCTION

In most database applications, one has to deal with incomplete data – this fact has been recognized almost immediately after the birth of the relational model. And yet for a long time the treatment of incomplete information did not receive the attention that it deserved. The design of SQL, for example, is notorious for its nulls-related features, which, as [14] nicely put it, are "in some ways fundamentally at odds with the way the world behaves" (a well-known example of such an oddity is that the statements $|X| > |Y|$ and $X - Y = \emptyset$ are logically consistent in SQL!). On the theoretical side, foundational research from the 1980s, first by Imielinski and Lipski [25] and then by Abiteboul, Kanellakis, and Grahne [3] provided models of incompleteness appropriate for handling queries in different relational languages, and established the computational costs of the key tasks associated with these models. However, until recently, the topic has seen only sporadic activity.

The situation changed rather dramatically over the past several years though, and handling of incompleteness in databases was brought to the fore of database research. This happened mainly due to a number of new applications such as those dealing with data on the web, integrated data, or data that was moved between different applications. In them, the issue of incompleteness is essential. For example, in data integration and data exchange, null values and associated concepts such as representation systems and certain answers play a central role [1, 6, 18, 28]. Representation and querying of uncertain data has also been an active field of study [5, 32]; another active direction that borders incompleteness is handling of probabilistic data [35].

Most of the classical theory of incompleteness has been developed in the relational context, where the key concepts (such as, e.g., certain answers) are well understood. A number of attempts were made in the 1990s to develop a general theory of incompleteness in databases, applicable to multiple data models [9, 10, 30, 31, 34]. This was done by using techniques from semantics of programming languages [21]. The connection is rather natural: in programming semantics, a program is assigned a meaning in an ordered set, where the order describes how partial, or incomplete, a function is. In

databases, we deal with domains of objects, rather than functions, but these are still ordered based on the degree of incompleteness.

Those general theories, while achieving some success with nested relations [29, 30], fell well short of handling incomplete information in more complex data models, notably XML. We shall elaborate on the reasons for it later in the paper. Incompleteness in XML was addressed recently [4, 7, 15], and one issue that required a complete reworking was the concept of certain answers for queries that return XML documents (i.e., the notion of certain information in a family of trees).

Such certain answers were defined in [15] using an approach that, on the surface looked very different from the relational approach. But does it mean that incompleteness in XML is really different from relational incompleteness? And if it is not – as we shall argue – then what is the general theory that encompasses them all and can be applied to other models?

Our main goal is to develop such a general theory of incompleteness in databases, that subsumes in a uniform way existing relational and XML results, is easily extendible to other data models, and is useful in the main applications of incompleteness.

Our theory will again be based on the notion of an information ordering, to represent the degree of an incompleteness of a database object. Unlike investigations from the 1990s, however, we shall use orderings that are closely tied with the semantics of instances that most commonly arise in applications (more precisely, we deal mainly with naïve tables and their analogs). In view of this, we start by examining two different approaches to handling incompleteness: in relational databases, and in XML documents. We reformulate (and in some cases, strengthen) known results to demonstrate the usefulness of information ordering for query answering.

We then present a very general and datamodel-independent framework for handling incompleteness, that allows us to make two conclusions. First, the notions of certainty in relational and XML contexts, while very different on the surface, are really identical, and correspond to computing greatest lower bounds in ordered sets. Second, we find an easily verified criterion for queries that ensures that certain answers can be computed by *naïve evaluation*, i.e., treating nulls as if they were usual attribute values.

After that, we define the most natural ordering for both relational and XML databases: namely, an object x is less informative than an object y if x denotes more complete objects (indeed, no information means that every instance is possible, so the more objects are possible as the denotation of x, the less informative x is). These orderings can be characterized in terms of the existence of homomorphisms between instances. This immediately allows us to adapt techniques from graph theory, where lattices of graphs and their cores with respect to homomorphism-based orderings have been studied [23] (note, however, that homomorphisms of database instances are not identical to graph homomorphisms, so some work is required in adapting graph-theoretic techniques). We show how to compute greatest lower bounds for finding certain answers, and use this new view of the problem to answer several open problems about the existence of certain answers for XML queries.

To demonstrate that this approach is not limited to relational or XML data, we consider a general setting, reminiscent of data models in [13, 22, 27]. It separates structural and data aspects of a model, which is, essentially, a colored relational structure, where the color of a node determines the length of a tuple of data values attached to it. In relational databases, the underlying structure is just a set; in XML, it is, as expected, a tree.

It turns out that it is often easier to reason about such general models, deriving particular results (say, for XML) as corollaries. For example, the model immediately makes it clear how to compute lower bounds for certain answers – it is the only solution that "typechecks". We also look at upper bounds and explain that they are naturally related to building universal solutions in data exchange.

We conclude by studying computational problems typical in the context of incomplete information. As the underlying structure in our general model conveys some schema information, it gives rise to the consistency problem for incomplete objects. We also look at the membership problem: whether a complete instance represents a possible world for an incomplete one. For this problem, we prove a strong generalization of polynomial-time algorithms for both relational [3] and XML [7] cases under the Codd-interpretation of nulls (when each null occurs at most once). Finally, we look at query answering, and provide broad classes of queries for which upper bounds coincide with those for XML, even for much more general data models.

Organization In Section 2 we review incompleteness in relational and XML models (stating some new results along the way). In Section 3 we present a general model based on orderings. In Section 4 we study homomorphism-based orderings for relations and XML. In Section 5 we present a general data-model that subsumes relations and XML and study incompleteness in it. Section 6 studies common computational problems associated with incompleteness in the general model.

2. INCOMPLETENESS IN RELATIONS AND XML

2.1 Incompleteness in relations

We start with a few standard definitions. We assume two disjoint sets of values: $\mathcal{C}$ of constants, and $\mathcal{N}$ of nulls (which will be denoted by $\perp$, with sub/superscripts). A

relational schema is a set of relation names with associated arities. An incomplete relational instance associates with each k-ary relation symbol S a k-ary relation over $\mathcal{C} \cup \mathcal{N}$, i.e., a finite subset of $(\mathcal{C} \cup \mathcal{N})^k$. When the instance is clear from the context we shall write S for the relation itself as well.

Such incomplete relational instances are referred to as *naïve* databases [2, 25]; note that a null $\perp \in \mathcal{N}$ can appear multiple times in such an instance. If each null $\perp \in \mathcal{N}$ appears at most once, we speak of *Codd* databases. If we talk about single relations, it is common to refer to them as naïve tables and Codd tables.

The semantics $[\![D]\!]$ of an incomplete database D is the set of complete databases it can represent. These are defined via *homomorphisms*. We write $\mathcal{C}(D)$ and $\mathcal{N}(D)$ for the sets of constants and nulls, resp., that occur in D. A homomorphism $h : D \to D'$ between two databases of the same schema is a map $h : \mathcal{N}(D) \to \mathcal{C}(D') \cup \mathcal{N}(D')$ such that, for every relation symbol S, if a tuple $\bar{u}$ is in relation S in D, then the tuple $h(\bar{u})$ is in the relation S in D'. As usual, we extend h to constants by letting $h(c) = c$ for every $c \in \mathcal{C}$. The semantics of an incomplete database D, denoted by $[\![D]\!]$ is then defined as the set of complete databases R such that there is a homomorphism $h : D \to R$.

For example, given a naïve table D below, the relation shown next to it is in $[\![D]\!]$, as witnesses by homomorphism $h(\perp_1) = 4$, $h(\perp_2) = 3$, and $h(\perp_3) = 5$:

D:

1	2	$\perp_1$
$\perp_2$	$\perp_1$	3
$\perp_3$	5	1

R:

1	2	4
3	4	3
5	5	1
3	7	8

Certain answers and naïve evaluation Given an incomplete database D and a query Q, one normally tries to compute *certain answers*:

$$\mathsf{certain}(Q, D) \;=\; \bigcap \{Q(R) \mid R \in [\![D]\!]\},$$

that are true regardless of the interpretation of nulls.

Note that when Q is a Boolean (true/false) query, we normally associate true with the set containing the empty tuple, and false with the empty set. Then the above definition says that for a Boolean query, $\mathsf{certain}(Q, D)$ is true iff Q is true in every R from $[\![D]\!]$.

The problem of finding certain answers is undecidable for FO queries (as it becomes finite validity), but for positive relational algebra queries there is a simple polynomial-time algorithm, described below.

Define $Q_{\text{naïve}}(D)$ as follows: first, evaluate Q as if nulls were values (e.g., $\perp_1 = \perp_1$, $\perp_1 \neq \perp_2$, $\perp_1 \neq c$ for every $c \in \mathcal{C}$), and then eliminate tuples with nulls from the result. A classical result from [25] states that if Q is a union of conjunctive queries, then $\mathsf{certain}(Q, D) = Q_{\text{naïve}}(D)$. We can actually show that this is essentially optimal: one cannot find a larger subclass of FO for which naïve evaluation would compute certain answers.

Proposition 1. *If Q is a Boolean FO query and $\mathsf{certain}(Q, D) = Q_{naïve}(D)$ for all naïve databases D, then Q is equivalent to a union of conjunctive queries.*

Certain answers via preorders Note that each naïve database D is naturally viewed as a *Boolean conjunctive query (CQ)* Q_D. For example, the naïve database shown earlier is viewed as a CQ

$$\exists x_1, x_2, x_3 \; D(1, 2, x_1) \wedge D(x_2, x_1, 3) \wedge D(x_3, 5, 1),$$

that is, each null $\perp_i$ is replaced by an existentially quantified variable x_i. Likewise, each Boolean CQ Q can be viewed as a naïve database D_Q (its tableau).

By viewing incomplete databases as logical formulae, we can restate the definition of the semantics in terms of satisfaction of formulae: $R \in [\![D]\!]$ iff $R \models Q_D$.

We now relate certain answers to two standard *preorders* (recall that a preorder is a relation that is reflexive and transitive). The first is what we referred to as the *information ordering* in the introduction: a naïve database D_1 is *less informative* than D_2 if it represents more databases, i.e. $D_1 \preceq D_2$ iff $[\![D_2]\!] \subseteq [\![D_1]\!]$. Notice that this is a preorder rather than a partial order: if $[\![D]\!] = [\![D']\!]$, then both $D \preceq D'$ and $D' \preceq D$ hold.

For queries, we have a well-known preorder: containment (denoted, as usual, by $Q_1 \subseteq Q_2$). The following is essentially a restatement of known results:

Proposition 2. *For a Boolean conjunctive query Q and a naïve database D, the following are equivalent:*

1. $\mathsf{certain}(Q, D) = \mathit{true}$;
2. $D_Q \preceq D$;
3. $Q_D \subseteq Q$.

This also immediately implies that $\mathsf{certain}(Q, D)$ is the greatest lower bound of the set $\{Q(D') \mid D \preceq D'\}$, where by $Q(D')$ we mean running a query over an incomplete database as if it were a complete database.

These indicate that information ordering and lower bounds have a close connection with certain answers; this will be made much more precise later in the paper.

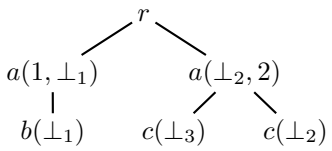
2.2 Incompleteness in XML

Although rather elaborate models of incompleteness for XML exist [4, 7], for our purposes we shall present only a fairly simple model, with nulls occurring as attribute values. This will suffice to explain the main concept used for defining certain answers for XML queries.

The data model is unranked trees, with nodes labeled by letters from a finite alphabet Σ. For each letter $a \in \Sigma$, we have an associated arity $ar(a)$ telling us the number of attributes such nodes carry. More precisely, a tree over Σ is defined as $T = \langle V, E, \lambda, \rho \rangle$, where

- $\langle V, E\rangle$ is a rooted directed unranked tree, with nodes V and the child relation E;
- $\lambda : V \to \Sigma$ is a labeling function; if $\lambda(v) = a$ then we refer to v as an a-labeled node;
- the function ρ assigns to each a-labeled node an $ar(a)$-tuple of data values from $\mathcal{C} \cup \mathcal{N}$.

An example is shown below. Here the alphabet is $\{a, b, c\}$, the arity of a is 2, both b and c have arity 1, and data values are shown next to the labels.



If all the data values in $\rho(v)$ come from $\mathcal{C}$ and the root is labeled by a designated root symbol $r \in \Sigma$, then we refer to a *complete* tree T; otherwise we speak of *incomplete trees*. The semantics of incomplete trees is again defined by means of homomorphisms which act on both tree nodes and data values.

Given a tree T, we use the notation $\mathcal{C}(T)$ for the set of constants in $\mathcal{C}$ that appear as data values in T, and $\mathcal{N}(T)$ for the set of nulls that appear in it. A *homomorphism* $h : T \to T'$ from a tree $T = \langle V, E, \lambda, \rho\rangle$ into a tree $T' = \langle V', E', \lambda', \rho'\rangle$ is a pair of mappings $h_1 : V \to V'$ and $h_2 : \mathcal{N}(T) \to \mathcal{C}(T') \cup \mathcal{N}(T')$ such that:

1. if $(x, y) \in E$ then $(h_1(x), h_1(y)) \in E'$;
2. if x is labeled a in T, then $h_1(x)$ is labeled a in T', i.e., $\lambda'(h_1(x)) = \lambda(x)$; and
3. the data values of $h_1(x)$ are the result of h_2 on the data values of x, i.e., $\rho'(h_1(x)) = h_2(\rho(x))$.

As usual, we extend h_2 to be the identity on constants.

Now the semantics $[\![T]\!]$ of an incomplete tree T is defined as the set of all complete trees T' such that there is a homomorphism $h : T \to T'$.

Certain answers via max-descriptions To define relational certain answers, we used intersection. But how do we define them for trees – what is an analog of intersection then? Most papers addressing incompleteness in XML only looked at queries returning relations, to find a way around this problem.

For proper XML-to-XML queries, the problem was addressed in [15]. If T is an incomplete XML tree, and Q is a query returning XML trees, then certain answers $\mathsf{certain}(Q, T)$ should be the certain information in the set $Q([\![T]\!]) = \{Q(T') \mid T' \in [\![T]\!]\}$. Thus, we need to know how to extract certain information from a collection of XML documents.

For this, it is convenient to use the analogy with naïve tables and CQs, and view trees as both objects and formulae: we say $T \models T'$ iff there is a homomorphism from T' to T (i.e., T satisfies what is described by an incomplete tree T'; this is the analog of $D \models Q_{D'}$ we used above to describe when D is more informative than D'). In XML, this corresponds to describing incompleteness via tree patterns, as done in, e.g., [4, 7, 8].

Our goal now is to extract the certain information from a set $\mathcal{T}$ of incomplete trees. The proposal of [15] was to view incomplete trees as *both* models and formulas, and reconstruct a classical model/formulas Galois connection (cf. [37]). More precisely, for $\mathcal{T}$ we define:

- its theory $\mathrm{Th}(\mathcal{T}) = \{T' \mid \forall T \in \mathcal{T} :\ T \models T'\}$;
- its models $\mathrm{Mod}(\mathcal{T}) = \{T' \mid \forall T \in \mathcal{T} : T' \models T\}$.

Under this view, $\mathrm{Th}(\mathcal{T})$ describes the certain knowledge conveyed by $\mathcal{T}$. If we want to capture it by an object, we look for a tree T whose models are the models of the certain knowledge, i.e., $\mathrm{Mod}(T) = \mathrm{Mod}(\mathrm{Th}(\mathcal{T}))$. Such trees T were called *max-description* in [15].

Note that all max-descriptions T, T' are equivalent, i.e., $T \models T'$ and $T' \models T$. The definition of certain answers $\mathsf{certain}(Q, T)$ in [15] was simply to take a max-description of $Q([\![T]\!])$. It was shown that this agrees perfectly with the relational definition when XML documents model relations. It was also shown that every set of complete XML documents has a max-description, and every finite set of complete or incomplete XML documents has a max-description. In the finite case, these are computable in time polynomial in the total size of $\mathcal{T}$, and exponential in the number of trees in $\mathcal{T}$.

It was left open however whether arbitrary sets (in fact, even computable sets) of XML documents have max-descriptions. We shall answer this soon. To do so, we first build a general theory (which will completely demystify max-descriptions, among other things), and then apply it to relations, trees, and beyond.

3. ORDERED SETS AND INCOMPLETENESS

The notions of certain answers for relational and XML queries, on the surface, appear to be very different. But in reality, they are not; in fact, they are the same. To see this, we now describe a very general framework for handling incompleteness and certain answers. In this framework, all that we are going to assume is the presence of an information ordering on a set of database objects. Even with this, we can reason about certain answers, max-descriptions, query answering, and naïve evaluation.

First, some very basic definitions. A *preorder* $\preceq$ is a binary relation that is transitive and reflexive. Associated with the preorder we have an equivalence relation $x \sim y$ defined by $(x \preceq y) \wedge (y \preceq x)$, and the quotient of $\preceq$ by $\sim$ is a partial order. We let $\uparrow x = \{y \mid x \preceq y\}$

and $\downarrow x = \{y \mid y \preceq x\}$. Also $\uparrow X = \bigcup_{x \in X} \uparrow x$ and likewise for $\downarrow X$. A lower bound of a set X is an element y such that $y \preceq x$ for all $x \in X$. A *greatest lower bound (glb)* of X is a lower bound y such that $y' \preceq y$ whenever y' is another lower bound of X. It is denoted by $\bigwedge X$. Note that in a preorder, $\bigwedge X$ is an equivalence class wrt $\sim$, but whenever we do not distinguish between equivalent elements we shall write, slightly abusing notation, $y = \bigwedge X$. Glb's need not exist in general.

By a *database domain* we mean a set $\mathcal{D}$ with a preorder $\preceq$ on it. The interpretation is as follows: $\mathcal{D}$ is a set of database objects (say, incomplete relational databases or incomplete trees over the same schema), and $\preceq$ is the information ordering. That is, we assume that incomplete database objects come with some notion of their semantics $[\![\cdot]\!]$ (what exactly it is, is immaterial to us in this section; this will become important when we look at concrete models), and the interpretation of the preorder is, as before $x \preceq y$ iff $[\![y]\!] \subseteq [\![x]\!]$. The associated equivalence relation $x \sim y$ is simply $[\![x]\!] = [\![y]\!]$.

Next, we need to define *certain information* in a set $X \subseteq \mathcal{D}$ of objects. If this certain information is represented by an object c, then:

1. c is less informative than every object in X, since it defines information common to all objects in X; that is, c is a lower bound of X;
2. if there are two such objects c, c' and $c' \preceq c$, then we prefer c as it has more information.

Thus, we want the most informative object that defines information common to all objects in X, i.e., a maximal element among all lower bounds of X. That is, *the certain information in the set X is $\bigwedge X$*, the glb of X.

If we look at relational databases without nulls (which arise as elements of $Q([\![D]\!])$), then the semantic ordering for them is $\subseteq$. Thus, for any collection $\mathcal{X}$ of complete databases, we have $\bigwedge \mathcal{X} = \bigcap_{D \in \mathcal{X}} D$ (i.e., in the case of relations certain answers are obtained as intersections).

Certain information defined by max-descriptions does not at first seem to be related to greatest lower bounds; nonetheless, we show that this is precisely what it is.

Max-descriptions deconstructed As we did with naïve databases and XML trees, we can view objects as partial descriptions. Then $x \models y$ is just $y \preceq x$: an object x satisfies every description that is less informative than itself. Therefore, $\mathrm{Mod}(x) = \uparrow x$ and $\mathrm{Th}(x) = \downarrow x$; these are extended to sets, as usual, by $\mathrm{Mod}(X) = \bigcap_{x \in X} \mathrm{Mod}(x)$ and similarly for Th.

The definition of certain information in a set X used in [15] (in the XML context) was a max-description of X: an object x such that $\mathrm{Mod}(x) = \mathrm{Mod}(\mathrm{Th}(X))$.

Theorem 1. *Given a subset X of a database domain $\mathcal{D}$, an element x is a max-description of X iff $x = \bigwedge X$. In particular, a max-description of a set exists iff its greatest lower bound exists.*

So max-descriptions are precisely glb's. The XML case was not different after all: for both relations and XML, the certain information in a set of objects is its glb.

Certain answers for queries We can now use the notion of certain information in a set of objects to define certain answers to queries. An abstract view of a query is as follows. For each schema σ, we have a domain $\mathcal{D}_\sigma$ of database objects of that schema, as well as a preorder $\preceq_\sigma$ on it (if clear from the context which relation $\preceq_\sigma$ we refer to, we write just $\preceq$). A *query* is then a mapping $Q : \mathcal{D}_\sigma \to \mathcal{D}_\tau$ between two domains. If we have a set $X \subseteq \mathcal{D}_\sigma$ of objects, then $Q(X) = \{Q(x) \mid x \in X\}$, and certain answers to Q over X are defined as

$$\mathsf{certain}(Q, X) \;=\; \bigwedge Q(X).$$

There is a simple recipe for finding certain answers when a query Q is *monotone*, i.e., when $x \preceq_\sigma y$ implies $Q(x) \preceq_\tau Q(y)$.

A *basis* of $X \subseteq \mathcal{D}$ is a set $B \subseteq X$ such that $\uparrow B = \uparrow X$. The following simple observation was already made in some concrete cases (like incompleteness in XML [15]):

Lemma 1. *If Q is monotone and B is a basis of X, then* $\mathsf{certain}(Q, X) = \bigwedge Q(B)$.

Hence, if we can find a finite basis $B = \{b_1, \ldots, b_n\}$ of X, then we can compute certain answers $\mathsf{certain}(Q, X)$ as $Q(b_1) \wedge \ldots \wedge Q(b_n)$. Thus, in such a case it is essential to be able to compute the glb of two (and hence finitely many) elements. The meaning of monotonicity depends on the exact interpretation of the information ordering; this will be addressed in detail in Section 4.

While cases of finite but non-singleton bases do arise in applications [15], most often one computes certain answers over sets X of the form $[\![x]\!]$. In the very general ordered setting, one way of defining the semantics is by $[\![x]\!] = \uparrow x$, i.e., an object represents all objects which are more informative than itself. Then Lemma 1 implies:

Corollary 1. $\mathsf{certain}(Q, \uparrow x) = Q(x)$ *for a monotone Q.*

It says that certain answers can be computed by simply evaluating Q on x. However, in many models the semantics is more refined than just $\bigwedge Q(\uparrow x)$, and is based on *complete* objects that do not have null values. We now study how to incorporate those into our setting.

Complete objects

We extend our setting with the notion of complete objects, i.e., objects without nulls, and explain the properties that lead to naïve evaluation of queries. *Database domains with complete objects* are structures of the form $\langle \mathcal{D}, \preceq, \mathcal{C} \rangle$ where $\preceq$ is a preorder and $\mathcal{C} \subseteq \mathcal{D}$ is a set of objects we view as those having no nulls. To state some basic requirements for these sets, we look at their behavior in the standard models, i.e., in naïve tables.

1. For each object x, the set $\uparrow_{\mathrm{cpl}} x = \uparrow x \cap \mathcal{C}$ of more informative complete objects is not empty (guaranteeing well-defined semantics).
2. For each object x, there is a unique maximal complete object $\pi_{\mathrm{cpl}}(x)$ under x (think of a relation obtained by removing all the rows with nulls from a naïve table). The function π_{cpl} is monotone, and the identity on $\mathcal{C}$. In the standard terminology, this means that $\pi_{\mathrm{cpl}} : \mathcal{D} \to \mathcal{C}$ is a retraction.
3. There are complete objects in $\uparrow_{\mathrm{cpl}} y - \uparrow_{\mathrm{cpl}} x$, unless x is less informative than y. In essence, it says that there are sufficiently many complete objects.

These conditions are satisfied in the standard domains, such as naïve databases or XML documents with nulls. In those domains we define the semantics of x as $\uparrow_{\mathrm{cpl}} x$, the set of complete objects above a given object. It follows immediately from the definition that this notion of the semantics agrees with the ordering.

Lemma 2. *If $\langle \mathcal{D}, \preceq, \mathcal{C} \rangle$ is a database domain with complete objects, then $x \preceq y \;\Leftrightarrow\; \uparrow_{\mathrm{cpl}} y \;\subseteq \uparrow_{\mathrm{cpl}} x$.*

We denote the glb in the restriction $\langle \mathcal{C}, \preceq \rangle$ (when it exists) by $\bigwedge_{\mathrm{cpl}}$. This gives us the notion of certain answers based on complete objects, for a query $Q : \mathcal{D} \to \mathcal{D}'$ that send complete objects to complete objects. It is the glb of the answers to Q over complete objects dominating the input:

$$\mathsf{certain}_{\mathrm{cpl}}(Q, x) \;=\; \bigwedge\nolimits_{\mathrm{cpl}} Q\big(\uparrow_{\mathrm{cpl}} x\big).$$

We say that *certain answers are computed by naïve evaluation* if $\mathsf{certain}_{\mathrm{cpl}}(Q, x) = \pi_{\mathrm{cpl}}(Q(x))$: in other words, they are obtained by running Q and then finding the complete approximation of the result.

For relations, these are of course the familiar concepts, as $\mathsf{certain}_{\mathrm{cpl}}(Q, x)$ is exactly $\bigcap\{Q(R) \mid R \in [\![D]\!]\}$, and $\pi_{\mathrm{cpl}}(Q(D))$ is $Q_{\mathrm{naïve}}(D)$.

To provide a criterion for checking whether certain answers can be computed by naïve evaluation, we say that a function $f : \mathcal{D} \to \mathcal{D}'$ between two database domains with complete objects $\mathcal{C}$ and $\mathcal{C}'$ has the *complete saturation property* if it maps complete objects to complete objects and the following conditions hold:

- if $f(x) \in \mathcal{C}'$ then $f(c) = f(x)$ for some $c \in \uparrow_{\mathrm{cpl}} x$;
- if $f(x) \notin \mathcal{C}'$ and $f(x) \not\succeq c' \in \mathcal{C}'$, then $f(c)$ and c' are incompatible for some $c \in \uparrow_{\mathrm{cpl}} x$.

By incompatibility of two elements we mean that neither of them is less than the other. Intuitively, the conditions say that $\uparrow_{\mathrm{cpl}} x$ has enough complete objects to witness different relationships that involve $f(x)$.

Theorem 2. *For every query that is monotone and has the complete saturation property, certain answers are computed by naïve evaluation.*

Over relational databases, complete saturation is very easy to check for unions of CQs, providing an alternative proof that the naïve evaluation works for them (see Section 4). Note, however, that Theorem 2 is not limited to any language, as it identifies a structural property behind naïve evaluation.

4. HOMOMORPHISM-BASED ORDERING

We now apply the general theory to relational databases with nulls and to incomplete XML trees. In both cases, the ordering was based on the semantics: $D_1 \preceq D_2$ iff $[\![D_2]\!] \subseteq [\![D_1]\!]$ and likewise for trees. There is a simple way to characterize this ordering:

Proposition 3.
- *If D, D' are naïve databases over the same schema, then $D \preceq D'$ iff there is a homomorphism from D to D'.*
- *If T, T' are XML documents over the same alphabet, then $T \preceq T'$ iff there is a homomorphism from T to T'.*

Hence, we deal with a well-established notion, and the ordering corresponds to a concept studied in fields such as graph theory [23] and constraint satisfaction [26]. This raises two questions: how does it relate to orderings previously studied in connection with incompleteness, and what can we get from known results, in particular in graph theory? We now address those.

Information ordering and other orderings As mentioned in the introduction, orderings have been used to provide semantics of incompleteness in the past [9, 10, 30, 31, 34]. A typical ordering (say, for relational model) would be defined as follows. For two tuples over constants and nulls we let $(a_1, \ldots, a_m) \sqsubseteq (b_1, \ldots, b_m)$ if, for each each i, either a_i is a null, or both a_i and b_i are the same constant (i.e., each null is less informative than each constant). This would be lifted to sets by $X \sqsubseteq^\flat Y \;\Leftrightarrow\; \forall x \in X \exists y \in Y :\; x \sqsubseteq y$.

In general, $\sqsubseteq^\flat$ cannot coincide with $\preceq$, as testing the existence of a homomorphism is NP-complete, and $\sqsubseteq^\flat$ is easily testable in quadratic time. But they do coincide for Codd databases (where nulls cannot be reused).

Proposition 4. *If D and D' are Codd databases, then $D \preceq D'$ iff $D \sqsubseteq^\flat D'$.*

Thus, the orderings used back in the 1990s were well suited for the Codd interpretation of nulls, but not the most commonly used naïve interpretation, which arises in applications such as integration and exchange of data.

The lattice of cores Orderings induced by homomorphisms are well known in graph theory. We look at graphs $G = \langle V, E \rangle$, where V is a set of nodes and E is a set of (directed) edges. We write $G \preceq G'$ if there is a homomorphism $h : G \to G'$, i.e., a map $h : V \to V'$ such that $(x, y) \in E$ implies $(h(x), h(y)) \in E'$. We use the

same notation $\preceq$ as before, but it will always be clear from the context which homomorphisms we talk about. Clearly $\preceq$ is a preorder, and the associated equivalence relation $G \sim G'$ is given by $\text{core}(G) = \text{core}(G')$. Recall that the core of a graph G is the smallest subgraph G_0 of G such that there is a homomorphism from G to G_0; the core is unique up to isomorphism [23].

When restricted to cores, $\preceq$ defines a lattice with the glb $\wedge$ and the least upper bound $\vee$ given by [23]

- $G \wedge G' = \text{core}(G \times G')$, and
- $G \vee G' = \text{core}(G \bigsqcup G')$ ($\bigsqcup$ is the disjoint union).

In general, $G \times G'$ is an element of the equivalence class defining $G \wedge G'$, and $G \bigsqcup G'$ is an element of the equivalence class defining $G \vee G'$.

There are many facts known about this lattice; many of them stem from the classical result of Erdős [17] that there are graphs of arbitrarily large chromatic number and odd girth (the minimum length of an odd cycle). Since the former is monotone with respect to $\preceq$ and the latter is antimonotone, this lets one construct arbitrary antichains and dense chains inside $\preceq$. In fact, a very deep result [24] says that every countable partial order can be embedded into $\preceq$ over directed graphs.

However, "database homomorphisms" and "graph homomorphisms" are not the same. In naïve databases, homomorphisms are only defined on nulls, but their range is unrestricted. In XML, they are pairs of mappings, one on tree nodes, and the other on nulls. Thus, we cannot use standard graph-theoretic results, but of course we can adapt them.

Lower bounds for naïve databases We have seen that lower bounds are essential for computing certain answers. We now show how to compute the glb of two naïve tables (the construction extends to databases, simply by doing it relation-by-relation). The construction, as suggested by graph-theoretic results, is essentially a product that accounts for the different roles of nulls and constants.

Let R, R' be two naïve tables over the same set of attributes. Take any 1-1 mapping that assigns every pair $(x, y) \in \mathcal{C} \cup \mathcal{N}$ a null $\bot_{xy}$ that does not belong to $\mathcal{N}(R) \cup \mathcal{N}(R')$. As instances R, R' will always be clear from the context, we simply write $\bot_{xy}$ instead of the more formal but cumbersome $\bot_{x,y,R,R'}$. Then, for two tuples $t = (a_1, \ldots, a_m)$ and $t' = (b_1, \ldots, b_m)$, we define

$$t \otimes t' \;=\; (c_1, \ldots, c_m) : \;\; c_i = \begin{cases} a_i & \text{if } a_i = b_i \in \mathcal{C} \\ \bot_{a_i b_i} & \text{otherwise} \end{cases} \quad (1)$$

Proposition 5. *The set $\{t \otimes t' \mid t \in R,\ t' \in R'\}$ is $R \wedge R'$, i.e., a glb of R and R' in the preorder $\preceq$.*

Thus, for every finite collection $\mathcal{X}$ of naïve tables, $\bigwedge \mathcal{X}$ exists. In fact, measuring $\|\mathcal{X}\|$ as the total number of tuples in all relations $R \in \mathcal{X}$, one can easily use the inequality between the arithmetic and geometric means to derive that for $\mathcal{X}$ with n tables, $|\bigwedge \mathcal{X}| \leq (\|\mathcal{X}\|/n)^n$, for $\bigwedge$ constructed in Proposition 5. Results of [15] can also be adapted to show that even $|\text{core}(\bigwedge(\mathcal{X}))|$ is necessarily exponential in the number of relations in $\mathcal{X}$.

What about infinite collections? Recall that in the case of XML, the existence of glb's (called at that time max-descriptions) for arbitrary collections was an open problem. We now show that in general, for arbitrary collections (even recursive ones), glb's need not exist.

Theorem 3. *There is an infinite family $\mathcal{X}$ of naïve tables such that $\bigwedge \mathcal{X}$ does not exist (in fact, there are uncountably many such families). Furthermore, there exist (countably many) recursive families $\mathcal{X}$ of relational databases such that $\bigwedge \mathcal{X}$ does not exist.*

Proof sketch. For the first statement, we use the fact that $\langle \mathbb{Q}, < \rangle$ can be embedded into the lattice of cores. If we look at naïve tables that only contain nulls, we have an embedding of $\langle \mathbb{Q}, < \rangle$ into the ordering $\preceq$. If each set had a glb, the Dedekind-MacNeille completion of $\langle \mathbb{Q}, < \rangle$ would be embeddable into the (countable) set of naïve tables, which is impossible by the cardinality argument.

For the second statement, we show that the family of directed cycles whose length is a power of two does not have a glb. □

Lower bounds for XML First, observe that the notion $T \models T'$ used in [15] to define certain answers is precisely $T \preceq T'$ (the existence of homomorphisms) and thus, by Theorem 1, max-descriptions for XML as defined in [15] are the glb's in the ordering $\preceq$. Now by coding naïve tables as XML documents (for each tuple there is a child of the root, whose attribute values are the values in the tuple), and using Theorem 3, we answer the open question from [15].

Corollary 2. *There are recursive collections $\mathcal{X}$ of XML documents (of depth 2) for which $\bigwedge \mathcal{X}$ does not exist.*

We shall have more to say about glb's for XML documents in the next section. For now, we offer one comment on the interaction between glb's and sibling-ordering in XML documents. Note that all the work on computing certain answers in XML was done for unordered documents. It turns out that if we add sibling-ordering, then glb's do not exist even for finite collections, which justifies restricting to unordered documents for handling certain answers to queries.

Proposition 6. *There are ordered XML trees T, T' such that $T \wedge T'$ does not exist.*

Proof sketch. Both T and T' have a root and two children, labeled a and b, but ordered differently. It is easy to show that $T \wedge T'$ does not exist. □

Naïve evaluation under $\preceq$ Theorem 2 stated two sufficient conditions for queries to admit naïve evaluation: monotonicity and complete saturation property.

We now see what they mean for relational databases wrt to the semantic ordering $D \preceq D' \Leftrightarrow [\![D']\!] \subseteq [\![D]\!]$. Note that monotonicity wrt this ordering is *not* the usual database notion of monotonicity which uses the partial order $\subseteq$: instead, it corresponds to preservation under homomorphisms. The complete saturation property, it turns out, is very easy to check for unions of CQs.

Proposition 7. *Every union of conjunctive queries is monotone and has the complete saturation property with respect to* $\preceq$.

Proof. Monotonicity wrt $\preceq$ (i.e., preservation under homomorphisms) is well known for unions of CQs. To show complete saturation, let Q be a union of UCQs. If $Q(D)$ does not have nulls, then, for h that maps all nulls in D into distinct constants different from those in D, by genericity we get $Q(h(D)) = h(Q(D)) = Q(D)$. For the second property, take a complete $R \not\preceq Q(D)$, assume that $Q(D)$ has nulls, and let h be a 1-1 map that sends nulls to constants not present in D and R. Take the complete database $h(D) \succeq D$. By genericity, $Q(h(D)) = h(Q(D)) \not\subseteq R$. If $R \subseteq Q(h(D))$ then again by genericity $R = h^{-1}(R) \subseteq h^{-1}(Q(h(D))) = Q(D)$, a contradiction which proves complete saturation. □

Together with Theorem 2 this gives an alternative proof of the classical result that certain answers for unions conjunctive queries can be obtained by naïve evaluation. Note that Theorem 2 is a general result that is not restricted to FO queries, as it states a structural condition behind naïve evaluation.

A remark about CWA We worked with the Open World Assumption (OWA) here: the ordering $\preceq$ is defined as the existence of an *into* homomorphism. The Closed World Assumption corresponds to the existence of an *onto* homomorphism $h : D \to D'$; in this case we write $D \preceq_{\text{CWA}} D'$. We now offer one comment on the relationship between this ordering, and orderings corresponding to CWA that were considered in the past. Typically (see [9, 30]), to model CWA, one used the Plotkin ordering [21] for sets, defined as $X \sqsubseteq^\sharp Y$ iff $\forall x \in X \exists y \in Y : x \sqsubseteq y$ and $\forall y \in Y \exists x \in X : x \sqsubseteq y$, to lift the ordering on tuples to the ordering on sets. We saw that $\sqsubseteq^\flat$ coincides with $\preceq$ over Codd databases. Will $\preceq_{\text{CWA}}$ coincide with $\sqsubseteq^\sharp$ under the same restriction?

It turns out that they do not coincide, but they are very close. Recall that a relation $S \subseteq A \times B$ satisfies Hall's condition iff $|S(U)| \geq |U|$ for every $U \subseteq A$ (this is the requirement for the existence of a perfect matching).

Proposition 8. *Over Codd databases,* $D \preceq_{\text{CWA}} D'$ *iff* $D \sqsubseteq^\sharp D'$ *and* $\sqsubseteq^{-1}$ *satisfies Hall's condition.*

5. INCOMPLETENESS IN GENERAL DATA MODELS

To further understand incompleteness in relational, XML, and potentially other models, we need to provide algorithms for computing glb's for certain answers, and to understand the basic computational properties associated with incompleteness (i.e., membership, query answering, consistency). The most general setting provided in Section 3 is too general for reasoning about such concrete tasks, and especially their complexity, and in Section 4 we handled relational and XML cases separately, as they had usually been handled in the literature. But we show now that this artificial separation is not necessary and we can derive many results simultaneously for a variety of data models.

Of course looking for general data models subsuming many others is nothing new in database theory, and several have been proposed (e.g., [13, 22, 27]). What we do here is similar in spirit (and in fact closest to [13]), and the model is well-suited to talk about relations in XML in one common framework.

The basic idea is simple: databases and documents have a structural part (e.g., trees for XML) and data part (tuples of attributes attached to tree nodes). The number of attributes attached to a node is determined by the label of that node. This model also subsumes relations, as the simplest case: the structural part is just a set, as we are about to see.

5.1 Generalized databases and information ordering

We now formalize this as follows. A *generalized schema* $\mathbb{S} = \langle \Sigma, \sigma, ar \rangle$ consists of a finite alphabet Σ, a relational vocabulary σ, and a function $ar : \Sigma \to \mathbb{N}$. To define databases, we assume sets $\mathcal{C}$ of constants and $\mathcal{N}$ of nulls, as before, as well as a set $\mathcal{V}$ of nodes to describe the structural part. A *generalized database* over $\mathbb{S}$ is then $\mathfrak{D} = \langle \mathbf{M}, \lambda, \rho \rangle$ where

- $\mathbf{M}$ is a finite σ-structure over $\mathcal{V}$;
- λ is a labeling of elements of $\mathbf{M}$ in Σ, and
- ρ assigns to each node ν in $\mathbf{M}$ a k-tuple over $\mathcal{C} \cup \mathcal{N}$ where k is the arity of the label of ν, i.e., $ar(\lambda(\nu))$.

Both relational and XML databases are special cases of generalized databases. To code a relational database, let $\sigma = \emptyset$ (i.e., $\mathbf{M}$ is just a set), and let Σ contain relation names, with the same arity as the relations themselves. For example, a relational instance $\{R(1, \bot_1), S(\bot_1, \bot_2, 2)\}$ is represented by a set $\{\nu_1, \nu_2\}$ with $\lambda(\nu_1) = R$ and $\lambda(\nu_2) = S$; the arities of R and S are 2 and 3, and $\rho(\nu_1) = (1, \bot_1)$ and $\rho(\nu_2) = (\bot_1, \bot_2, 2)$. For XML, σ is a vocabulary for unranked trees (there could be several; in the example in Sec. 2 we used only the edge relation, but one can use other axes such as next-sibling), and generalized databases themselves are trees with attribute tuples attached to nodes (again, as shown in the example in Sec. 2).

We shall write $\mathbf{M}_\lambda$ for the colored structure $\langle \mathbf{M}, \lambda \rangle$; technically, it is a structure in the vocabulary σ expanded with unary relations P_a for each $a \in \Sigma$, whose

interpretation is the set of nodes labeled a. We also write $\langle \mathbf{M}_\lambda, \rho \rangle$ for a generalized database.

To define the semantics in terms of complete generalized databases (i.e., those not using nulls), we use, as before, homomorphisms between generalized databases. We let $\mathcal{V}(\mathfrak{D})$, $\mathcal{C}(\mathfrak{D})$, and $\mathcal{N}(\mathfrak{D})$ stand for the sets of vertices, constants, and nulls in $\mathfrak{D}$. A *homomorphism* $h : \mathfrak{D} \to \mathfrak{D}'$ between $\mathfrak{D} = \langle \mathbf{M}_\lambda, \rho \rangle$ and $\mathfrak{D}' = \langle \mathbf{M}'_{\lambda'}, \rho' \rangle$ is a pair $h = (h_1, h_2)$ of mappings $h_1 : \mathcal{V}(\mathfrak{D}) \to \mathcal{V}(\mathfrak{D}')$ and $h_2 : \mathcal{N}(\mathfrak{D}) \to \mathcal{C}(\mathfrak{D}') \cup \mathcal{N}(\mathfrak{D}')$ such that:

1. h_1 is a usual homomorphism $\mathbf{M}_\lambda \to \mathbf{M}'_{\lambda'}$;
2. if $\nu \in \mathcal{V}(\mathfrak{D})$ and $\nu' = h_1(\nu)$ then $\rho'(\nu') = h_2(\rho(\nu))$.

As always, we extend h_2 to be the identity on constants. Observe also that in condition 2, ν and ν' have the same label (since h_1 is a homomorphism) and hence tuples $\rho(\nu)$ and $\rho(\nu')$ have the same length. This notion of homomorphism becomes one of the standard notions when we consider relational databases or XML documents.

We now define $[\![\mathfrak{D}]\!]$, the semantics of $\mathfrak{D}$, as the set of all complete generalized databases $\mathfrak{D}'$ such that there is a homomorphism $h : \mathfrak{D} \to \mathfrak{D}'$. As usual, the information ordering is

$$\mathfrak{D} \preceq \mathfrak{D}' \;\Leftrightarrow\; [\![\mathfrak{D}']\!] \subseteq [\![\mathfrak{D}]\!].$$

We have a standard characterization of it:

Proposition 9. $\mathfrak{D} \preceq \mathfrak{D}'$ *iff there is a homomorphism* $h : \mathfrak{D} \to \mathfrak{D}'$.

We now look at constructions of upper and lower bounds wrt the information ordering. The greatest lower bound can be used for computing certain answers, while upper bounds naturally correspond to constructing target instances in data exchange.

5.2 Greatest lower bounds for certain answers

We have already argued that we need greatest lower bounds, or glb's, to compute certain answers to queries. As glb's of arbitrary sets need not exist in general, we want to compute finite bases of databases and calculate glb's over the results of queries on such finite bases. Lemma 1 tells us that this will give us certain answers for queries which are monotone wrt the information ordering.

However, simply calculating glb's as products is not always going to work. For example, the product of two trees is not a tree, so this will not work in the case of XML. In such a case we need glb's in the subclass of generalized databases for which the structural part is a tree. Generally, if we have a class $\mathcal{K}$ of structures $\mathbf{M}_\lambda$, we need a glb $\wedge_{\mathcal{K}}$ of *$\mathcal{K}$-generalized databases*, i.e., those generalized databases in which the structural part is restricted to be in $\mathcal{K}$ (think, for example, of XML documents: the underlying structures $\mathbf{M}_\lambda$ in this case are not arbitrary but labeled trees).

We now show how to construct such a glb using the minimal possible set of assumptions. In fact, in a way it is easier to do the construction in this general setting, without knowing what the concrete structures in $\mathcal{K}$ are, as the construction is fully guided by the setup: it is the only one that "typechecks". It consists of two steps: first, compute the lower bound for the structural part, and then add data as we did for relations.

Our minimal assumption is that we have a glb $\mathbf{M}_\lambda \sqcap_{\mathcal{K}} \mathbf{M}'_{\lambda'}$ of structures in class $\mathcal{K}$ (that is, a glb that itself is a member of $\mathcal{K}$). Without knowing anything at all about its structure, we can only conclude that $\mathbf{M}_\lambda \sqcap_{\mathcal{K}} \mathbf{M}'_{\lambda'} \preceq \mathbf{M}_\lambda \times \mathbf{M}'_{\lambda'}$, since the latter is a glb without the restriction to $\mathcal{K}$. That is, there is a homomorphism $h : \mathbf{M}_\lambda \sqcap_{\mathcal{K}} \mathbf{M}'_{\lambda'} \to \mathbf{M}_\lambda \times \mathbf{M}'_{\lambda'}$. Composing h with first and second projections from $\mathbf{M}_\lambda \times \mathbf{M}'_{\lambda'}$ to $\mathbf{M}_\lambda$ and $\mathbf{M}'_{\lambda'}$ gives us homomorphisms

$$\iota : \mathbf{M}_\lambda \sqcap_{\mathcal{K}} \mathbf{M}'_{\lambda'} \to \mathbf{M}_\lambda \quad \text{and} \quad \iota' : \mathbf{M}_\lambda \sqcap_{\mathcal{K}} \mathbf{M}'_{\lambda'} \to \mathbf{M}'_{\lambda'}.$$

This provides the construction of $\mathfrak{D} \wedge_{\mathcal{K}} \mathfrak{D}'$ of a glb of two $\mathcal{K}$-generalized databases. We let

$$\langle \mathbf{M}_\lambda, \rho \rangle \;\wedge_{\mathcal{K}}\; \langle \mathbf{M}'_{\lambda'},\ \rho' \rangle \stackrel{\text{def}}{=} \langle \mathbf{M}_\lambda \sqcap_{\mathcal{K}} \mathbf{M}'_{\lambda'}, \rho \otimes \rho' \rangle \quad (2)$$

where $\rho \otimes \rho'(\nu) = \rho(\iota(\nu)) \otimes \rho'(\iota'(\nu))$.

Here $\otimes$ is the merge operation on tuples (1) we used to define relational glb in Section 4 (technically speaking, it depends on $\mathfrak{D}, \mathfrak{D}'$ as the nulls it generates must be outside $\mathcal{N}(\mathfrak{D}) \cup \mathcal{N}(\mathfrak{D}')$, but the instances are always clear from the context). Since ι and ι' are homomorphisms, the colors of $\iota(\nu)$ and $\iota'(\nu)$ are the same, and hence $\rho(\iota(\nu)) \otimes \rho'(\iota'(\nu))$ is well-defined.

Theorem 4. *For every class $\mathcal{K}$ of labeled σ-structures that admits glbs, $\mathfrak{D} \wedge_{\mathcal{K}} \mathfrak{D}'$ given by (2) is a glb in the class of $\mathcal{K}$-generalized databases ordered by $\preceq$.*

When $\sigma = \emptyset$, i.e., when we deal with relational databases, this yields precisely the construction for relations in Proposition 5.

When $\mathcal{K}$ is the class of unranked trees (in the vocabulary of the child relations and labels), $\wedge_{\mathcal{K}}$ is precisely the construction shown for computing certain answers in [15]. The construction of the glb for trees themselves is standard, and is done inductively, level by level, by pairing nodes with the same labels.

5.3 Least upper bounds for data exchange

We now cast the well-studied problem of finding solutions in data exchange [6, 18] in our framework, proving a partial explanation why in some cases finding solutions is easy, and in some it is not. We shall present both relational and XML data exchange uniformly, using our notion of generalized databases.

In data exchange, we have a source schema $\mathbb{S}$, a target schema $\mathbb{S}'$, and a schema mapping $\mathbb{M}$ which consists of

the rules of the form $\mathfrak{I} \to \mathfrak{I}'$, where $\mathfrak{I}$ and $\mathfrak{I}'$ are instances over $\mathbb{S}$ and $\mathbb{S}'$, respectively. Given a complete instance $\mathfrak{D}$ over $\mathbb{S}$, an instance $\mathfrak{D}'$ over $\mathbb{S}'$ is called a *solution* for $\mathfrak{D}$ if for every rule $\mathfrak{I} \to \mathfrak{I}'$ in $\mathbb{M}$, and every homomorphism $(h_1, h_2) : \mathfrak{I} \to \mathfrak{D}$, there exists a homomorphism $(g_1, g_2) : \mathfrak{I}' \to \mathfrak{D}'$ such that g_2 coincides with h_2 on nulls common to $\mathfrak{I}$ and $\mathfrak{I}'$. A solution $\mathfrak{D}'$ is called *universal* (or $\mathcal{K}$-universal) if there is exists a homomorphism from $\mathfrak{D}'$ to every other solution (or every other solution whose structural part belongs to the class $\mathcal{K}$).

It is very easy to check that in the cases of relations and XML, this abstract view coincides precisely with the standard definitions of [6, 18] (in the case of XML, we deal of course with $\mathcal{K}$-universal solutions when $\mathcal{K}$ is the class of trees). To give a concrete example, consider a relational rule (these are known as st-tgds)

$$S(x, y, u) \to T(x, z), T(z, y).$$

For a source relational database $\mathfrak{D}_S$ with a ternary relation S over $\mathcal{C}$, a target database $\mathfrak{D}_T$ with a binary relation T over $\mathcal{C}$ and $\mathcal{N}$ is a solution if $\mathfrak{D}_S$ and $\mathfrak{D}_T$ satisfy the sentence $\forall x, y, u\ (S(x, y, u) \to \exists z\ (T(x, z) \wedge T(z, y)))$. We can view the left and right-hand sides of the rule as databases $\mathfrak{I}_S = \{S(x, y, u)\}$ and $\mathfrak{I}_T = \{T(x, z), T(z, y)\}$ over nulls. Then the rule is satisfied iff for each homomorphism $h : \mathfrak{I}_S \to \mathfrak{D}_S$ (i.e., a fact $S(a, b, c)$ in $\mathfrak{D}_S$, where $h(x) = a, h(y) = b, h(u) = c$) we have tuples $T(a, v), T(v, b)$ for some value v: in other words, a homomorphism $g : \mathfrak{I}_T \to \mathfrak{D}_T$ such that $g(z) = v$, and g coincides with h on x and y. Thus, we indeed deal with the standard concepts from data exchange.

We now show that universal solutions are *least upper bounds* (lub's) in the preorder $\preceq$; these will be denoted by $\bigvee$ or $\bigvee_{\mathcal{K}}$ when restricted to $\mathcal{K}$-generalized databases.

We let $\mathrm{Hom}(\cdot, \cdot)$ denote the set of homomorphisms between two instances. For a source instance $\mathfrak{D}$ and a mapping $\mathbb{M}$, define

$$\mathbb{M}(\mathfrak{D}) = \{h_2(\mathfrak{I}') \mid \mathfrak{I} \to \mathfrak{I}' \in \mathbb{M},\ (h_1, h_2) \in \mathrm{Hom}(\mathfrak{I}, \mathfrak{D})\}.$$

Intuitively, $(h_1, h_2) \in \mathrm{Hom}(\mathfrak{I}, \mathfrak{D})$ provides an instantiation of a rule in the mapping, and then $\mathbb{M}(\mathfrak{D})$ is the set of single-rule applications of rules in $\mathbb{M}$ to the source $\mathfrak{D}$. Computing those is often the first – and easy – step of building a solution in data exchange. Now we can relate $\mathbb{M}(\mathfrak{D})$ to universal solutions.

Theorem 5. *For a mapping $\mathbb{M}$ and a source $\mathfrak{D}$, the $\mathcal{K}$-universal solutions are precisely the elements of the $\sim$-equivalence class $\bigvee_{\mathcal{K}} \mathbb{M}(\mathfrak{D})$.*

This result partly explains why in some cases computing solutions in data exchange is easy, and in some it is not. Consider, for example, relational databases without putting any restrictions on them. Then lub's exist – as we saw before, they are essentially disjoint unions (technically, disjoint unions after renaming of nulls). Indeed, then $\bigsqcup \mathbb{M}(\mathfrak{D})$ is what is called canonical universal solution in data exchange (without constraints on the target). Furthermore, the canonical representative of the equivalence class $\bigvee \mathbb{M}(\mathfrak{D})$ in this case is the core solution in data exchange, that is, precisely $\mathrm{core}(\bigsqcup \mathbb{M}(\mathfrak{D}))$.

However, if we move to XML, and let $\mathcal{K}$ be the class of all unranked trees, we immediately encounter the following problem:

Proposition 10. *Least upper bounds do not always exist in the restriction of $\preceq$ to labeled unranked trees.*

Hence, in the case of XML (especially with additional schema information) and relations with extra target constraints, one needs to find not an lub (as it may not exist) but rather *some* upper bound. This loss of canonicity in the choice of a solution leads to the necessarily ad-hoc – and varying – choices for solutions, which we see in data exchange literature outside of restricted cases of nicely behaved mappings.

6. COMPUTATIONAL PROBLEMS

We now address the complexity of the key computational problems associated with incompleteness. We do it in our general model from Section 5, and show that this form of reasoning allows us to get some results for both relational and XML cases in a uniform way.

The standard problems to consider are [2, 3, 4, 7, 25]:

- Consistency: does an incomplete database have a completion satisfying some conditions? This problem is commonly considered in the XML context, where schemas are usually more complex. This problem tends to be NP-complete, and in PTIME with suitable restrictions [7].
- Membership: does an incomplete database represent a complete one? It is NP-complete for naïve databases and XML documents, and in PTIME in both cases under the Codd interpretation (although the proofs of these facts in [3, 7] use very different techniques).
- Query answering. Typically one asks whether a given tuple of values is a certain answer. Complexity tends to range, depending on the language, from undecidable (for sufficiently expressive languages to encode the validity problem) to coNP-complete to PTIME [3, 7].

We now look at these problems in our general setting.

Consistency

Let φ be a condition, given by a logical formula, over the structures $\mathbf{M}_\lambda$. The consistency problem is:

PROBLEM:	CONS(φ)
INPUT:	a generalized database $\langle \mathbf{M}_\lambda, \rho \rangle$
QUESTION:	is there $\langle \mathbf{M}'_{\lambda'}, \rho' \rangle \in [\![\langle \mathbf{M}_\lambda, \rho \rangle]\!]$ such that $\mathbf{M}'_{\lambda'} \models \varphi$?

In general we should avoid undecidable classes of formulae defining structural conditions; indeed, since every generalized database belongs to $[\![\emptyset]\!]$, the consistency problem checks, in particular, satisfiability of sentences. Even though we only deal with data complexity (φ is fixed), we still prefer to avoid problems whose combined complexity is undecidable.

As a sample result on the complexity of the consistency problem, we consider the well-known decidable case of $\exists^*\forall^*$-formulas (i.e., the Bernays-Schönfinkel class, consisting of formulae of the form $\exists x_1, \ldots, x_k \forall y_1, \ldots, y_m \alpha(\bar{x}, \bar{y})$, where α is quantifier-free), and classify the complexity of CONS(φ) based on the exact shape of the quantifier prefix.

Proposition 11. • *If φ is an $\exists^*\forall^*$ sentence, then* CONS*(φ) is in* NP.

- *There is a $\exists^*\forall$ sentence φ_0 such that* CONS*(φ_0) is* NP*-complete.*
- *If φ is an $\exists^*$ sentence, then* CONS*(φ) is in* PTIME.

Notice that we consider *data* complexity, i.e., the sentence φ is fixed, so there is no contradiction with the known higher complexity for the satisfiability problem for the Bernays-Schönfinkel class.

Membership

The membership question is whether $\mathfrak{D}' \in [\![\mathfrak{D}]\!]$ for a complete database $\mathfrak{D}'$ and an incomplete database $\mathfrak{D}$. More generally, we can ask whether $\mathfrak{D} \preceq \mathfrak{D}'$ (which is the membership problem when $\mathfrak{D}'$ has no nulls).

Since checking whether $\mathfrak{D} \preceq \mathfrak{D}'$ amounts to checking the existence of a homomorphism from $\mathfrak{D}$ to $\mathfrak{D}'$, we deal with the general constraint-satisfaction problem [26]. Such a problem is in NP, and often NP-complete, even for a fixed $\mathfrak{D}'$. One case that is solvable in PTIME in both relational and XML contexts is the case of the Codd interpretation of nulls. The proofs of these results are very different however: for relational Codd tables, [3] reduced the problem to finding matchings in bipartite graphs, and for XML, [7] used an analog of CTL-model-checking algorithms on finite Kripke structures.

Now we provide a uniform explanation. We say that in a generalized database $\langle \mathbf{M}_\lambda, \rho \rangle$, the function ρ has *Codd interpretation* if each null occurs as its value at most once.

Theorem 6. *For each fixed $k > 0$, checking whether $\langle \mathbf{M}_\lambda, \rho \rangle \preceq \langle \mathbf{M}'_{\lambda'}, \rho' \rangle$ can be done in polynomial time if ρ has Codd interpretation, and the treewidth of $\mathbf{M}_\lambda$ is at most k.*

Both relational and XML polynomial-time algorithms for the Codd interpretation of nulls are special cases of Theorem 6 when $k = 1$. This result is not a corollary of the standard results on the tractability of constraint satisfaction problems with bounded treewidth (cf. [19, 26]) due to the presence of data values and special conditions on homomorphisms.

Query answering

To answer questions about the complexity of query answering, we need, of course, a query language for generalized databases. Here we look at a natural analog of FO. For relational databases, we know that query answering is in PTIME for unions of conjunctive queries (i.e., existential positive FO sentences) but undecidable for all of FO [3]. For XML, even for analogs of conjunctive queries on trees [20, 8] the complexity of finding certain answers can be coNP-complete, but this is typically caused by missing structural information or the presence of a schema [7].

Since generalized databases are two-sorted structures, it appears to be natural to consider a two-sorted version of FO. We can however rather easily avoid the cumbersome multi-sorted presentation by considering, for a generalized schema $\mathbb{S} = \langle \Sigma, \sigma, ar \rangle$, the logic FO($\mathbb{S}, \sim$), which is first-order over σ, the labeling predicates, and predicates $=_{ij}$ (x, y) meaning that the ith attribute of x equals the jth attribute of y.

Note that this covers both relational and XML Boolean conjunctive queries (for XML, σ defines the set of axes). So the natural problem we have is the following, where φ is a sentence of FO($\mathbb{S}, \sim$):

PROBLEM:	QA(φ)
INPUT:	a generalized database $\mathfrak{D}$
QUESTION:	is $\mathsf{certain}(\varphi, \mathfrak{D}) = \text{true}$?

We now show that all three cases witnessed for relations and XML – tractability, intractability, and undecidability – are possible.

Theorem 7. • *If φ is an existential positive sentence of* FO($\mathbb{S}, \sim$)*, then* QA*(φ) is in* DLOGSPACE.

- *If φ is an existential sentence of* FO($\mathbb{S}, \sim$)*, then* QA*(φ) is in* coNP*. Moreover, there exists a generalized schema $\mathbb{S}$ and an existential sentence φ_0 of* FO($\mathbb{S}, \sim$) *so that* QA*(φ_0) is* coNP*-complete.*
- *There exists a generalized schema $\mathbb{S}$ and a sentence φ_1 of* FO($\mathbb{S}, \sim$) *so that* QA*(φ_1) is undecidable.*

7. FUTURE WORK

We briefly outline some directions for future work. Many results in this paper are of generic nature, and it would be nice to understand how they work when

they are applied for particular classes of structures. For example, Theorem 7 is stated over the class of all generalized databases, and one would like to see how similar results change when we impose restrictions on the structural part (e.g., require it to be a tree).

We have not looked at constraints, but they are known to cause problems in the presence of incompleteness; in particular, they affect complexity and decidability results [12]. We also would like to see if the structural study of constraints imposed on target instances in data exchange will help determine classes for which least upper bounds, and thus universal solutions, exist; to start with, one can attempt to extract such structural conditions from cases when the chase procedure is known to work (e.g. [18, 16]), or from restricted classes of tgds studied in ontological reasoning [11].

The general model we used in Section 5, while inspired by several other graph-based models, is rather close to the model of [13]. The focus of [13] was rather different from ours, but a more detailed study of the connections with that paper may be warranted.

Finally, all the results here (except Proposition 8) are based on the open world assumption. An ordering can naturally be extracted from the closed world semantics as well, and we plan to study it in the future. Going from OWA to CWA is often known to increase the complexity of the main computational tasks though [36].

Acknowledgments I thank Pablo Barceló, Claire David, Egor Kostylev, and Filip Murlak for helpful discussions on incompleteness and certain answers. I am especially grateful to Filip for suggesting a shorter proof of the second part of Theorem 3. Supported by EPSRC grant G049165 and FET-Open Project FoX, grant agreement 233599.

8. References

[1] S. Abiteboul, O. Duschka. Complexity of answering queries using materialized views. In *PODS 1998*, pages 254–263.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] S. Abiteboul, P Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *TCS*, 78(1):158–187, 1991.

[4] S. Abiteboul, L. Segoufin, and V. Vianu. Representing and querying XML with incomplete information. *ACM TODS*, 31(1):208–254, 2006.

[5] L. Antova, C. Koch, D. Olteanu. 10^{10^6} worlds and beyond: efficient representation and processing of incomplete information. *VLDB J.* 18(5): 1021-1040 (2009).

[6] M. Arenas, P. Barceló, L. Libkin, F. Murlak. *Relational and XML Data Exchange*. Morgan & Claypool, 2010.

[7] P. Barceló, L. Libkin, A. Poggi, and C. Sirangelo. XML with incomplete information. *J. ACM* 58(1): 1–62 (2010).

[8] H. Björklund, W. Martens, and T. Schwentick. Conjunctive query containment over trees. In *DBPL'07*, pages 66–80.

[9] P. Buneman, A. Jung, A. Ohori. Using powerdomains to generalize relational databases. *TCS* 91 (1991), 23–55.

[10] P. Buneman, S. Davidson, A. Watters. A semantics for complex objects and approximate answers. *JCSS* 43(1991), 170–218.

[11] A. Calì, G. Gottlob, T. Lukasiewicz. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In *ICDT'10*, pages 14–30.

[12] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS'03*, pages 260–271.

[13] S. Cohen and Y. Sagiv. An abstract framework for generating maximal answers to queries. In *ICDT 2005*, pages 129–143.

[14] C. Date and H. Darwin. *A Guide to the SQL Standard*. Addison-Wesley, 1996.

[15] C. David, L. Libkin, F. Murlak. Certain answers for XML queries. In *PODS 2010*, pages 191-202.

[16] A. Deutsch, A. Nash, J. Remmel. The chase revisited. In *PODS'08*, pages 149–158.

[17] P. Erdős. Graph theory and probability. *Canad. J. Math.* 11 (1959), 34–38.

[18] R. Fagin, Ph. Kolaitis, R. Miller, and L. Popa. Data exchange: semantics and query answering. *TCS*, 336(1):89–124, 2005.

[19] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[20] G. Gottlob, C. Koch, and K. Schulz. Conjunctive queries over trees. *J. ACM* 53(2):238-272, 2006.

[21] C. Gunter. *Semantics of Programming Languages*. The MIT Press, 1992.

[22] M. Gyssens, J. Paredaens, J. Van den Bussche, D. Van Gucht. A graph-oriented object database model *IEEE TKDE* 6(4):572–586, 1994.

[23] P. Hell and J. Nešetřil. *Graphs and Homomorphisms*. Oxford University Press, 2004.

[24] J. Hubička and J. Nešetřil. Finite paths are universal. *Order* 22(1):21–40, 2005.

[25] T. Imieliński and W. Lipski. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[26] P. Kolaitis and M. Vardi. A logical approach to constraint satisfaction. In *Finite Model Theory and Its Applications*, Springer 2007, pages 339–370.

[27] G. Kuper and M. Vardi. The logical data model. *ACM TODS* 18(3):379–413 (1993).

[28] M. Lenzerini. Data integration: a theoretical perspective. In *PODS'02*, pages 233–246.

[29] M. Levene and G. Loizou. Semantics of null extended nested relations. *ACM TODS* 18 (1992), 414-459.

[30] L. Libkin. A semantics-based approach to design of query languages for partial information. In *Semantics in Databases*, LNCS 1358, 1998, pages 170–208.

[31] A. Ohori. Semantics of types for database objects. *Theoretical Computer Science* 76 (1990), 53–91.

[32] D. Olteanu, C. Koch, L. Antova. World-set decompositions: expressiveness and efficient algorithms. *TCS* 403 (2008), 265–284.

[33] B. Rossman. Homomorphism preservation theorems. *J. ACM* 55(3): (2008).

[34] B. Rounds. Situation-theoretic aspects of databases. In *Situation Theory and Appl.*, CSLI vol. 26, 1991, pages 229-256.

[35] D. Suciu. Probabilistic databases. *Encyclopedia of Database Systems*, 2009, pages 2150-2155.

[36] M. Vardi. On the integrity of databases with incomplete information. In *PODS'86*, pages 252–266.

[37] W. Wechler. *Universal Algebra for Computer Scientists*. Springer, 1992.