Representing and Querying Incomplete Information: a Data Interoperability Perspective

Cristina Sirangelo

Mémoire d'Habilitation à Diriger des Recherches
École Normale Supérieure de Cachan

Representing and Querying Incomplete Information: a Data Interoperability Perspective

Cristina Sirangelo
École Normale Supérieure de Cachan

This thesis has been defended on December 5th, 2014 at École Normale Supérieure de Cachan, before the following jury:

- Nicole Bidoit - Université Paris Sud (reviewer) - *absent*
- Georg Gottlob - University of Oxford (reviewer)
- Maurizio Lenzerini - Università di Roma, La Sapienza (reviewer)
- Wim Martens - University of Bayreuth
- Nicole Schweikardt - University of Berlin, Humboldt
- Luc Segoufin - INRIA, ENS-Cachan

Abstract

This thesis is intended to be a succinct and rather informal presentation of some of my most recent work, which has been done in collaboration with several other people. In particular this thesis concentrates on our contributions to the study of incomplete information in the context of data interoperability. In this scenario data is heterogenous and decentralized, needs to be integrated from several sources and exchanged between different applications.

Incompleteness, i.e. the presence of "missing" or "unknown" portions of data, is naturally generated in data exchange and integration, due to data heterogeneity. The management of incomplete information poses new challenges in this context.

The focus of our study is the development of models of incomplete information suitable to data interoperability tasks, and the study of techniques for efficiently querying several forms of incompleteness.

The work presented in Chapter 4 is ongoing in the context of Nadime Francis's PhD, whom I am co-supervising together with Luc Segoufin.

Acknowledgements

I gratefully thank all the people with whom I co-authored the work presented in this thesis: Pablo Barceló, Nadime Francis, Amélie Gheerbrant, Leonid Libkin, Antonella Poggi and Luc Segoufin.

Contents

Chapter 1

Introduction

Recent years have witnessed the proliferation of communicating applications generating and manipulating huge volumes of data. The distributed Web environment is certainly at the basis of this phenomenon: data on the Web resides at different sites, in different formats and without centralization.

In this setting *data interoperability* refers to the ability of different independent applications to operate on the same data, e.g. share or exchange data or integrate data from different sources. What makes these tasks difficult is the intrinsic syntactic and semantic heterogeneity of data handled by different applications.

Data translation between heterogeneous formats (often referred to as *data exchange*), and *data integration* from different sources are the basic building blocks of all data interoperability tasks.

Foundational work on data exchange and integration [Len02, LLR02, Gra02, HLS11, FKPT11, APR13] has provided evidence that data interoperability techniques have to rely on appropriate models of *incomplete information* in data.

In fact reconciling, as well as restructuring data, naturally generate missing information. Intuitively this is due to the fact that interoperability tasks need to rely on complex *schema mappings* between concepts and data formats of different communicating databases. These, in most cases, cannot fully specify the data translation.

The theory of incomplete information, originally developed in the 80's for relational databases [IL84, AKG91, GZ88], provided a solid ground for understanding incompleteness. In particular it provided a clean semantics for correctly querying databases with incomplete information, in contraposition with the badly designed null-related features of SQL [DD96].

However such a well established theory is no longer sufficient to cope with data interoperability tasks nowadays, for several reasons. First of all other data models than the relational one have emerged. The 2000s have seen the spread of the XML data model on the Web, and today we also deal with more general graph-structured data, which can be naturally found in central applications such as social networks and the semantic Web. Incompleteness beyond the relational data model has received much less attention.

Moreover incompleteness arising from data heterogeneity is often of a particular form, and may thus raise specific representation needs.

Finally the huge scale of data applications nowadays obliges us to aim at

particularly efficient data processing solutions. The possibility of tractable solutions in the presence of incomplete information has not been fully investigated in the past.

The work presented in this thesis stems from the need to provide suitable models and query mechanisms for incomplete data in a data interoperability context – the main application context where incompleteness arises. Specifically this perspective motivates and guides our study in various respects:

- The models and semantics of incompleteness we consider are mainly motivated by their applicability in a data exchange and integration context.
- We take into account different data models, such as the relational model, XML and graph data. On the one hand this brings us to develop specific solutions for each data model. On the other hand we develop unified techniques which abstract away the data models, and can be instantiated on several of them.
- With the objective of finding efficient solutions for managing incomplete information, our analysis aims at pushing tractability bounds as much as possible.

In this chapter we introduce our main contributions in the setting outlined above; these are then presented in more detail in the following chapters. We start by establishing the main connection between models/semantics of incomplete information and data interoperability tasks in Section 1.1, and we show how this connection naturally identifies several problems to investigate. In Section 1.2 we introduce these problems and explain how our results contribute to them.

We emphasize that this thesis does not exhaustively cover all the topics we have studied. A full list of publications can be found at the end of this document.

1.1 Data interoperability and incomplete information

To cope with the fact that the same data may be structured differently in different databases, schema mappings are usually adopted to specify the relationship between concepts of different schemas [BM07, Kol05, Len02].

In very abstract terms a *schema mapping* can be viewed as a set of assertions of the form

$$q_\sigma \ \rightarrow \ q_\tau$$

where q_σ is a query over a *source* schema σ an and q_τ is a query over a *target* schema τ.

Such an assertion intuitively states that data selected by q_σ in the first database "corresponds" to data selected by q_τ in the other database. As we will see in the sequel, the precise adopted meaning of this "correspondence" may vary depending on several factors such as the particular formalism used, the data model and the type of application.

Target instances satisfying the schema mapping with a given source are in general further restricted by the possible presence of *constraints* at the target schema. Source instances may be subject to constraints as well.

We remark that, although the notion of schema mapping above can in principle account for source and target structured according to different data models (e.g. relational source and XML target), we will always deal with schema mappings where both source and target are schemas of the same data model.

Schema mappings play a central role in both data exchange and integration. Existing implementations [FHH+09, HHH+05] have also been incorporated into major database products.

In *data exchange* the objective is to restructure data from a source schema σ into a target schema τ, in order to allow the transfer of data from a source application to a target one. Schema mappings between σ and τ are used to represent the needed data "translation". The assertions $q_\sigma \rightarrow q_\tau$ in this case are often referred to as *souce-to-target dependencies*.

In a *data integration* scenario, the objective is to unify different heterogeneous source databases, into a single unified *global* database (which can be either *materialized* or just *virtual*). In such a setting a schema mapping represents the correspondence between concepts of the schema σ of the sources and the corresponding concepts represented in the global schema τ.

As an example of schema mapping consider two relational databases storing data about travel plans. The first one records flight reservations and night-stays of customers, as well as flight schedules. The second represents full trips, combining both flight and hotel reservations, and a possible discount for such combined reservations. A possible schema mapping is given by the following dependency:

Flight-res(cust, flight)∧		∃ hotel, discount(
Flight(flight, dep-city, arr-city, date)	→	Trip(cust, flight, hotel, date, discount)
∧ Stay(cust, date, arr-city)		∧ Hotel (hotel, arr-city))

The query on the left finds flight reservations with a corresponding night-stay reservation in the arrival city in the same date. The query on the right specifies that this data is represented in the target/global database as a single trip combining both a hotel and a flight reservation, and a possible discount; the hotel is also associated with the arrival city.

Incomplete information is central when dealing with schema mappings, as we discuss next.

1.1.1 Schema mappings and incompleteness

Schema mappings usually underspecify the relationship between the source and the target. In fact in both data integration and exchange scenarios it is common to have concepts represented in the target/global schema which are not (directly) modeled in the sources. For instance in the example above the source does not represent explicitly the hotels, and there is no notion of discount, since flight and hotel reservations are dealt with independently from one another.

As a consequence several target instances (or instances of the global schema) may satisfy the schema mapping dependencies with the same source instance (these target instances are usually called *solutions* in data exchange [FKMP05], and *legal* global databases in data integration [Len02]). In the example above, among others, any target instance associating a particular hotel and a discount value to each tuple resulting from the source query is a valid one.

In general the *semantics* of a schema mapping specifies what it means for a target instance to satisfy the schema mapping with a given source instance (i.e. it specifies the notion of solution in data exchange, and the notion of legal global database in data integration). This clearly depends on the particular chosen formalism for specifying mappings and constraints, but also on the way one choses to "interpret" the dependencies, as we will discuss later.

The situation where a database is only partially specified, and may thus have several possible instantiations, can often be modeled as a form of *incomplete information* in data. In fact the theory of incomplete information is centered around a notion of *possible worlds* [IL84, AKG91]: an incomplete database is essentially a set of possible databases, one for each possible way of interpreting the missing information.

Thus the correspondence specified by schema mappings can be viewed as generating *missing or unknown information* in the target.

To deal with such incompleteness, and therefore with the multiplicity of target instances compatible with the source data and the schema mapping, query answering over the target/global schema is usually based on the notion of *certain answers* [AD98, Kol05, Len02]. If Q is a query over the target schema, given a database S over the source schema, the *certain answers* to Q over S under schema mapping Σ are defined as the intersection of $Q(D)$'s for all target instances D satisfying the schema mapping with S (according to the adopted semantics of the schema mapping):

$$\mathsf{certain}_{\Sigma}(Q, S) = \bigcap\{Q(D) \mid D \text{ satisfies } \Sigma \text{ with } S\}.^{1}$$

I.e. certain answers are computed over all data exchange solutions/legal global databases.

Certain answers are the main query answering mechanism in data exchange and integration, as well as other applications of incompleteness such as ontology-based query answering [CGL$^+$07, CGP12, BtCLW13] and querying inconsistent data [ABC99, CLR03].

1.1.2 Incomplete data versus views

Data exchange and integration – although based on a similar specification formalism – have different objectives.

Problems arising in the two settings are technically close (and indeed techniques for dealing with them are often similar, see [DGLLR07]), however they offer different angles to look at the problem of dealing with schema mappings. As we will next explain, these different angles justify that we may concentrate on slightly different questions in the two settings, and we may deal with different forms of incompleteness.

In particular, we will see that in data exchange incomplete information about the target is provided by materialized incomplete target instances. To the contrary in common data integration settings we often deal with virtual global

[1]This definition assumes that Q returns a relation. When moving beyond the relational data model this is not the only notion of queries we want to deal with. An extension of the notion of certain answers for queries returning trees or more general objects was given in [DLM10, Lib11]; a generalized notion of certain answers was also proposed in [Lib14], but will not be considered in this thesis.

databases, and incomplete information about the global database is provided by a form of views.

Data exchange and incomplete instances In data exchange the goal is to transfer data from a source to a target. Since source and target are intended to be independent applications, queries posed to the target should be answered using local materialized data; source data is not considered available. This raises the question of *building solutions*, i.e. constructing suitable target instances which are general enough to compactly represent the whole space of solutions (such target instances are often called *universal solutions* [Kol05] or *universal representatives* [BPR13] in the data exchange literature). Incompleteness can be used to this end. In fact since incomplete instances represent a set of possible completions, they can be used to materialize the target and "capture" the whole space of solutions.

One can devise several ways of representing partial information about a database. The theory of relational incompleteness developed in [IL84] is based on the notion of *nulls* (i.e. variables, usually denoted by the symbol $\perp$ with sub/superscripts) to represent unknown data values.

In its simplest form an *incomplete relational instance* (also referred to as *naïve table* [IL84]) is just a database instance whose domain may contain both constants and nulls. The set of *complete* instances represented by an incomplete database I, usually denoted by $[\![I]\!]$, depends on the *semantics* of incompleteness, which intuitively asserts the way unknown/missing data should be interpreted. For example in the simplest case $[\![I]\!]$ contains all *valuations* of I, i.e. instances obtained from I by replacing nulls with constants (we will deal with several other semantics of incompleteness in this thesis).

This model of incompleteness has served as a working ground for the study of relational data exchange [HLS11, FKPT11, APR13, GO12]. Indeed nulls can model the form of incompleteness generated by schema mappings in the target.

As an illustration, consider the example given in Section 1.1.1. An incomplete instance containing tuples of the form $\mathsf{Trip}(c, f, \perp, d, \perp')$ and $\mathsf{Hotel}(\perp, a)$ for all customers c, flights f, dates d and arrival cities a satisfying the source query, can represent arbitrary solutions (under suitable semantics of both schema mappings and incompleteness).

Queries issued on the target can then be answered based on a single materialized incomplete target instance (the *canonical solution* and the *core* in data exchange are examples of such instances [Kol05, ABFL04]).

In fact certain answers are one of the main querying mechanisms over incomplete data as well. If I is an incomplete instance, certain answers to a query Q, usually denoted by $\mathsf{certain}(Q, I)$, are answers which are true over all instances belonging to $[\![I]\!]$.

Therefore if one can find for example an incomplete target instance I^* with the property that $[\![I^*]\!]$ coincides with the set of data exchange solutions for a source S, then for a query Q over the target:

$$\mathsf{certain}_\Sigma(Q, S) = \mathsf{certain}(Q, I^*)$$

Techniques for efficiently computing certain answers over incomplete instances are then of central importance in data exchange.

Of course the expressiveness of the model of incompleteness is an essential point here (in relationship with the constraint and query language). In particular even in the cases where incomplete instances of a given model cannot represent the space of data exchange solutions (especially in the presence of constraints), this may still be true as far as answering given classes of target queries is concerned [FKMP05, GO12]. Moreover the space of data exchange solutions depends on the semantics of schema mappings; different semantics of incompleteness are then needed, as we will discuss later.

Data integration and views Turning to the data integration scenario, the point of view is slightly different. The global schema is in general only a virtual reconciled view of the source data, and the global instance is usually not materialized (although materialized and hybrid data integration settings have been considered as well [Hul97]).

User queries are issued on the global schema in order to query several sources in a unified way, but data usually only resides into the sources. The sources are the only "incomplete" available information about the global database. This usually raises the problem of reformulating (or *rewriting*) user queries, originally formulated on the global schema, as queries over the sources.

This problem may take several forms depending on the type and semantics of schema mappings. The data integration literature usually distinguishes among LAV,GAV and GLAV mappings. In LAV relational mappings each dependency relates a single source atom to a target query, and there is exactly one such dependency for each source relation; thus the sources can be viewed as defined by views over the global schema. The situation is reversed with GAV mappings where relations of the global schema are associated to queries over the source schema. GLAV mappings are the most general, where dependencies relate source queries to target queries.

Under the simplest form of GAV mappings rewriting is a relatively straightforward task, since each symbol of the global schema in the query can be replaced by its definition in terms of the source schema. Query rewriting is a more challenging problem under more general GAV mappings (with constraints) and under LAV mappings. The problem has also been considered in its full generality for GLAV mappings [CDLV12] by reducing it to a combination of LAV and GAV mappings.

In the LAV setting – which we concentrate on in this thesis – reformulating queries over the sources amounts to *answering queries using views*. Specifically, while queries are posed to the global schema, data only resides in some materialized views of the virtual global database (i.e. the sources). One then needs to find a rewriting (often in some desired query language) to be issued on the view instance, that answers the original query.

Consider for example the case of two data sources, one representing information about conference papers with schema Paper(title, authors, year, pages, conference), and another one representing journal articles with schema Article(title, authors, year, pages, vol#, journal). Assume one wants to integrate these sources in a virtual global database representing arbitrary scientific publications, Publication(id, title, authors, year, pages, book_id), together with information about conference proceedings, Proceedings(id, conference, year), and journal volumes, Volume(id, vol#, journal, year). Let the correspondence be specified by the fol-

lowing LAV schema mapping:

$$
\begin{aligned}
&\mathsf{Paper}(\mathsf{title}, \mathsf{authors}, \mathsf{year}, \\
&\quad \mathsf{pages}, \mathsf{conference}) \rightarrow \quad \exists\, \mathsf{id}, \mathsf{book\_id}\ (\\
&\qquad \mathsf{Publication}(\mathsf{id}, \mathsf{title}, \mathsf{authors}, \mathsf{year}, \mathsf{pages}, \mathsf{book\_id}) \\
&\qquad \wedge\ \mathsf{Proceedings}(\mathsf{book\_id}, \mathsf{conference}, \mathsf{year})\ ) \\
&\mathsf{Article}(\mathsf{title}, \mathsf{authors}, \mathsf{year}, \\
&\quad \mathsf{pages}, \mathsf{vol\#}, \mathsf{journal}) \rightarrow \quad \exists\, \mathsf{id}, \mathsf{book\_id}\ (\\
&\qquad \mathsf{Publication}(\mathsf{id}, \mathsf{title}, \mathsf{authors}, \mathsf{year}, \mathsf{pages}, \mathsf{book\_id}) \\
&\qquad \wedge\ \mathsf{Volume}(\mathsf{book\_id}, \mathsf{vol\#}, \mathsf{journal}, \mathsf{year})\ )
\end{aligned}
$$

Queries asking about publications in the global schema will be answered over the sources distinguishing papers and articles, both defined as conjunctive views over the global schema.

The problem of answering queries using views has been considered in several variants (see for instance [AD98, Hal00, CGLV07, NSV10]), and is also of interest in other contexts then data integration. In all these variants one of the main objectives is to study when efficiently computable rewritings can be found.

In all problems mentioned in this section, the core question is to be able to represent and (efficiently) query a form of incomplete information about data, provided by either incomplete instances or views. These issues are the main focus of this thesis, as we will explain next.

1.2 Contributions

We now introduce the particular problems we concentrate on, and briefly outline our main contributions, which will be presented in more detail in the rest of this thesis.

1.2.1 Representing and querying incomplete data

Models and semantics of incompleteness As discussed in the previous section, data interoperability tasks, and in particular data exchange applications, need models and semantics of incompleteness to compactly represent the multiplicity of target solutions under schema mappings. The needed expressiveness of the model depends on several aspects.

First of all we may deal with different *semantics* of schema mappings. As we will argue, this may influence both the needed model and the semantics of incompleteness for the target.

In relational data exchange schema mappings are usually specified by a set Σ of first-order *source-to-target tuple generating dependencies* and target constraints [FKMP05]. The original semantics of such mappings introduced in [FKMP05] was based on the usual notion of FO satisfaction. I.e. under this notion, D is a target solution for a source instance S if $(S, D) \models \Sigma$. This semantics corresponds to a form of *Open World Assumption (OWA)* since it allows arbitrary data in the target, besides the data explicitely "imported" from the source.

There is an alternative notion of data exchange solutions, originally proposed in [Lib06] and later developed in [HLS11]. It is based on the *Closed World Assumption (CWA)*, which intuitively prevents the addition of data in the target which is not "justified" by the source and the mapping rules.

A particular target instance, called the *canonical solution* was introduced in [FKMP05]. It is obtained by chasing the source instance w.r.t the dependencies Σ. For example, if Σ consists of the source-to-target dependency $\forall xy(S(x,y) \rightarrow \exists z D(x,z))$, then for $S = \{(a,c_1),(a,c_2),(b,c_3)\}$, the canonical solution is $D^* = \{(a,\perp_1),(a,\perp_2),(b,\perp_3)\}$.

The canonical solution is an incomplete instance and one can interpret incompleteness in several ways. In particular OWA and CWA are two common semantics of incompleteness as well [IL84] (they will be recalled in Chapter 2).

For schema mappings specified by a set of source-to-target tuple generating dependencies, it turns out that the canonical solution, interpreted with OWA (respectively CWA) semantics of incompleteness, represents the space of open world (respectively closed world) data exchange solutions [FKMP05, Lib06].

This is immediate to see in the example of canonical solution above, under the OWA: notice that in this example a complete instance D is such that $(S,D) \models \Sigma$ if and only if D contains a valuation of D^*; the set of such D is precisely $[\![D^*]\!]$ under the OWA semantics of incompleteness.

In Chapter 2 we describe how we pushed this correspondence further in [5], by defining more flexible forms of schema mappings. In such mappings the semantics of data exchange is explicitly specified via a form of *open/closed* annotation of attributes, allowing a mixed open-world/closed-world semantics.

We show that there exists a notion of canonical solution that can represent the space of all solutions under annotated mappings. In order to do this we adopt a model of incompleteness based on annotated nulls introduced in [GZ88]. In Chapter 2 we present our analysis of the complexity of querying such annotated instances in data exchange. The reader is referred to our full article [5] and our invited paper [10] for a more comprehensive study of annotated schema mappings, including their closure properties w.r.t. composition (a basic schema management task [APRR09]).

We remark that semantics between closed world and open world have been recognized of particular interest in data exchange. Other data exchange semantics following this principle have been proposed, based on earlier work in the area of logic programming [Min82], and with them the need for other semantics of incompleteness. In particular for the so called GCWA*-semantics of data exchange [Her11], one can easily find a suitable semantics of incompleteness for the usual canonical solution, so that it represents the set of GCWA*-solutions. It is based on the combination of different possible "minimal" valuations of nulls. We introduced and studied this semantics of incompleteness for arbitrary instances in [1]; our results about efficient query answering under this semantics are briefly presented in Chapter 3.

The semantics of schema mappings is not the only factor that influences the need of specific models and semantics of incompleteness. As important is the possible presence of target constraints, or incompleteness in the source. Although these aspects are not considered in this thesis, representation issues related to them have been studied in the literature [APR13, GO12]. These works show that one needs the expressiveness of a more complex form of incompleteness, namely *conditional tables* [IL84], in order to represent the space of

solutions under schema mappings including large classes of target constraints, and possibly incomplete source instances. Alternatively naïve instances can provide a "weak" representation, i.e. intuitively a representation which is equivalent to the space of solutions, when it comes to compute certain answers to given classes of queries [FKMP05, GO12].

Our contributions are not limited to the relational setting. However while for relational databases we could rely on existing models and semantics of incompleteness, the picture is quite different for other data models.

The literature has to some extent addressed the problem of incompleteness in XML, but often in specific scenarios. For example [ASV06] concentrated on handling incompleteness arising in a dynamic setting in which the structure of a tree is revealed by a sequence of queries; graph and tree data models expressed as description logic theories that could incorporate incompleteness were dealt with in [CGL98, CGL02]; incompleteness in query results but not inputs was studied in [KNS02]; and incorporating probabilities into XML was looked at in [SA07, CKS09].

In the setting we are concerned with, incompleteness arises from schema mappings. XML schema mappings are usually *pattern based*, i.e they specify a correspondence between patterns of the source tree and patterns in the target tree [ABLM14]. This naturally calls for pattern-based models of incompleteness.

In [6] we followed this idea and developed and analyzed a pattern-based model of incomplete XML. We provided a classification of incomplete descriptions of XML documents, and we separated features - or groups of features - that lead to hard computational problems from those that admit efficient algorithms. Our classification of incomplete information is based on the combination of null values with partial structural descriptions of documents.

Our model is suitable for data exchange applications, and extends usual tree patterns used in this context. Fragments of our patterns have been considered in XML data exchange [AL08, ADLM14] to represent solutions.

There are several key computational problems of interest over incomplete tree descriptions. In [6] we considered *consistency* of partial descriptions, *representability* of complete documents by incomplete ones, and *query answering*. In [4] we also surveyed results about pattern *containment*.

In all cases we showed how factors such as schema information, the presence of node ids, and missing structural information affect the complexity of these main computational problems.

In Chapter 2 we present our model of incomplete XML and concentrate on the query answering problem, where our main results find robust classes of incomplete XML descriptions that permit tractable query evaluation.

Efficient query evaluation over incomplete data The problem of querying incomplete instances has been extensively studied both in foundational work on incompleteness [Lip79, IL84, AKG91, Lib11] and in the data exchange and integration literature [Kol05, Len02, Bar09]. As expected the problem is often hard, since computing certain answers is a form of entailment problem: it consists in checking whether every model of the incomplete description is also a model of the query. Computationally the problem usually ranges from coNP-complete to even undecidable depending on the model and semantics of incompleteness, as well as the query language.

Most of the work presented in Chapter 2 aims at tracing a frontier of tractability of query answering, under the models of incompleteness we consider for both relational and XML data.

Results in Chapter 2 already show that tractable solutions can often rely on variants of classical techniques for querying incomplete information. These techniques originated in [IL84], which not only identified union of conjunctive queries as a tractable class for querying naïve tables, but also provided a very natural procedure for computing certain answers to these queries under OWA. This procedure referred to as *naïve evaluation* essentially consists in evaluating the query over the incomplete data as if it were complete.

What makes this procedure particularly attractive is that in the case that naïve evaluation computes certain answers, query answering over incomplete instances is not only tractable, but can rely on known algorithms and optimization techniques.

Most of the tractability results we provide for query answering on both relational and XML incomplete data in Chapters 2 and 3 rely on a form of *naïve evaluation*. This shows that naïve evaluation is a versatile technique that makes sense well beyond the context of naïve tables and open world assumption. In fact we were able to apply it to more complex data models (e.g. XML data), under different semantics of incompleteness (e.g. various forms of CWA), and for non-positive queries as well.

Motivated by this observation, in [1] we embarked on a systematic study of naïve evaluation, which we present in Chapter 3. We analyzed the intrinsic properties that make naïve evaluation work, in a very general framework abstracting away the model, the semantics of incompleteness, and the query language. This allowed us to show how naïve evaluation depends on these parameters, and to apply results to several different settings.

1.2.2 Querying views

When partial information about a database is provided by materialized views, one usually assumes only a view instance S available, but no information about the original database D. In this sense S is a partial description which represents all possible databases D yielding it as a view instance.

We saw that this situation arises under LAV schema mappings, when one assumes only source data available. Queries are usually issued on the original database D (i.e. the virtual global database, in a data integration setting) but can only be answered using S (i.e. the available sources).

As with arbitrary schema mappings, one can consider several semantics of view definitions. The *sound view assumption* is a form of OWA for the LAV schema mapping. If V denotes the view definition, under the sound view assumption a view instance S represents all databases D such that $V(D) \supseteq S$. Under the the *exact view assumption*, to the contrary, a database D is compatible with a view instance S if and only if $S = V(D)$. This is a particular form of CWA for the corresponding LAV schema mapping.

One common approach to querying views is to compute certain answers over all databases compatible with the view [AD98, CDGLV00a], e.g. under the sound view assumption one computes $\mathsf{certain}_V(Q, S) = \bigcap\{Q(D) \mid V(D) \supseteq S\}$. Note that this coincides with $\mathsf{certain}_\Sigma(Q, S)$ under the LAV schema mapping Σ associating each view symbol to its definition (interpreted under the suitable

semantics). However, unlike in the data exchange setting, $\mathsf{certain}_V(Q, S)$ needs to be computed by issuing a query on the available view instance S (such a query is sometimes called a *perfect rewriting* [CGLV07]).

This approach has been thoroughly investigated both under the sound and the exact view assumption [AD98, CDGLV00a].

We follow a different approach, considered in the literature both for relational and semi-structured data [NSV10, CDGLV02, CGLV07]. It consists in studying conditions on views and queries which guarantee that the query result only depends on the view extension, and not on the particular database yielding this view.

This property of a set of view definitions V and a query Q is called *determinacy* (or *losslessness* as in [CDGLV02]); it amounts to require that any two databases having the same view also yield the same query result, i.e.

$$V(D) = V(D') \quad \Rightarrow \quad Q(D) = Q(D') \quad \text{for all } D, D'$$

Deciding determinacy answers precisely the question of whether the information provided by the view is always sufficient to answer the query, and is a question of interest in many application scenarios.

Under determinacy, computing certain answers on a given view instance is no longer necessary; in fact certain answers coincide with the query evaluated on any single instance yielding the given view. However this instance may be hard to compute from the view, and therefore in most cases it is unfeasible to use directly this approach for query answering.

On the other hand determinacy implies that there exists a function mapping view instances (V(D)) to corresponding query results (Q(D)). Therefore the query result $Q(D)$ can in principle be computed directly by issuing a query on the view instance $V(D)$, i.e. by only using the partial available information. Such query over the view is simply called a *rewriting* (or sometimes *exact rewriting* [CGLV02]). In other words a rewriting of a query Q using views V is a query R over view instances such that $R(V(D)) = Q(D)$ for all databases D.

Determinacy only implies the existence of a rewriting, but does not imply that such rewriting is expressible in a particular language, nor that enjoys particular computational properties. The *query rewriting problem* in a language $\mathcal{L}$ asks for the existence of a rewriting expressible in $\mathcal{L}$. We are usually interested in rewritings with efficient (polynomial time) data complexity.

Deciding determinacy and establishing a precise relationship between determinacy and rewriting are usually hard problems, which remain open in many settings [NSV10, CDGLV02], especially under the exact view assumption. In [2] we studied these problems under the exact view assumption for queries and views over *graph databases*, expressed as *Regular Path Queries*, a very common query language in this setting. Our main result shows that under a strong notion of determinacy, requiring an extra monotonicity assumption, one can always effectively construct rewritings in Datalog, therefore with efficient data complexity. These results have been obtained during Nadime Francis's PhD, whom I am co-supevising, and are presented in Chapter 4.

Chapter 2

Representing and querying incomplete data

In a data interoperability context data often needs to be materialized at the target as an incomplete instance. This is a particularly strict requirement in data exchange, since data is assumed to be no longer available at the source, once it has been transferred to another application. This raises two main issues: how to represent target data, so that it captures the intended data exchange semantics, and how to query it efficiently. In this chapter we start presenting our main contributions on the two aspects.

After recalling in Section 2.1 the basic model of relational incompleteness, in Section 2.2 we show how flexible models of incomplete information mixing open and closed world assumption can be used in relational data exchange to overcome some of the limitations of usual semantics. We establish the connection between query answering in data exchange and querying such incomplete data, and study its complexity [5].

In Section 2.3 we move to the XML data model. Motivated by XML schema mappings based on tree patterns, we develop pattern-based models of incomplete XML [6]. We concentrate in particular on separating features that lead to intractability of query answering from restrictions which allow tractable solutions.

2.1 Incompleteness in relational databases

In this section we review a basic model and some common semantics of incompleteness for relational data.

Incomplete instances As already introduced in Chapter 1, in the relational setting we consider incomplete instances with nulls, that appear most commonly in integration and exchange scenarios, and that can very easily be supported by commercial RDBMSs.

More formally we assume two countably infinite set of possible data values: Const representing the set of *constants*, and Null representing *nulls*. Nulls will normally be denoted by $\perp$, sometimes with sub- or superscripts.

An *incomplete relational instance* I (also referred to as *incomplete database*, or *naïve table* [IL84]) over a given relational schema assigns to each k-ary relation symbol of the schema a k-ary relation over $\mathsf{Const} \cup \mathsf{Null}$, i.e., a finite subset of $(\mathsf{Const} \cup \mathsf{Null})^k$. Its active domain is denoted by $\mathrm{adom}(I)$. Other models of incompleteness exist, and most notably *conditional tables* [IL84, AKG91], but we will not consider them in this thesis.

A *complete* database D has no nulls, i.e. its active domain is a subset of Const.

An incomplete database I represents a set of complete databases, denoted by $[\![I]\!]$. The *semantics of incompleteness* specifies this set. Several semantics have been considered in the literature starting with [IL84, Rei77]; they are all homomorphism-based.

Homomorphisms and valuations Given two (possibly incomplete) relational instances I and I' over the same schema, a *homomorphism* $h : I \to I'$ is a map from the active domain of I to the active domain of I' so that for every relation symbol R, if a tuple $\bar{u}$ is in relation R in I, then the tuple $h(\bar{u})$ is in the relation R in I'.

Given a homomorphism h and a database I, by $h(I)$ we mean the image of I, i.e., the instance consisting of all tuples $R(h(\bar{u}))$ where $R(\bar{u})$ is in I. If $h : I \to I'$ is a homomorphism, then $h(I)$ is a subinstance of I'.

A homomorphism h from I to I' is *strong onto* if $I' = h(I)$; It is *onto* if $\mathrm{adom}(I') = h(\mathrm{adom}(I))$.

A *valuation* on an incomplete instance I is a mapping assigning a value of Const to each null in $\mathrm{adom}(I)$. It is usually considered defined on the whole $\mathrm{adom}(I)$ by extending it to be the identity on Const. This way a valuation v can be viewed as a homomorphism from I to $v(I)$.

Semantics of incompleteness We shall see many possible semantics for incomplete information, but first we review two common ones: open world and closed world semantics.

The semantics under the closed world assumption (or *CWA semantics*) is defined as

$$[\![I]\!]_{\mathrm{CWA}} \;=\; \{v(I) \mid v \text{ is a valuation on } I\}.$$

The semantics under the open world assumption (or *OWA semantics*) is defined as

$$[\![I]\!]_{\mathrm{OWA}} \;=\; \{D \;\mid\; D \text{ is complete and } D \supseteq v(I) \text{ for some valuation } v\}.$$

Intuitively the closed world semantics assumes that the incomplete specification is "complete" as for tuples present in the database, and only data values are possibly missing. To the contrary under the open world assumption, an incomplete database specifies only positive information about the content of the database, and entire tuples may be possibly missing.

Another somewhat less known semantics was introduced in [Rei77]. We refer to it as the *Weak Closed World Assumption* (WCWA) semantics, since it relaxes the closed world restriction, but not as much as in the OWA. It is defined as follows:

$$[\![I]\!]_{\text{WCWA}} = \left\{ D \;\middle|\; \begin{array}{l} D \text{ is complete and there is a valuation } v \text{ such that} \\ D \supseteq v(I) \text{ and } \text{adom}(D) = \text{adom}(v(I)) \end{array} \right\}.$$

Certain answers Relational query languages considered in this thesis are fragments of FO. We will always assume active domain semantics for FO queries. When FO queries are evaluated over incomplete relational instances, we adopt the usual semantics of FO, by viewing nulls just as additional (pairwise distinct) domain elements, other than the constant elements.

Given an incomplete database I, a semantics of incompleteness $[\![\,]\!]$, and a query Q, *certain answers under the semantics* $[\![\,]\!]$ are

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

i.e. answers that are true regardless of the interpretation of nulls under the given semantics.

The complexity of computing certain answers for FO queries over incomplete relational instances ranges from coNP-complete to even undecidable depending on the restrictions on the type of incomplete instance, the semantics of incompleteness and the fragment of FO [AKG91]. In one case of particular interest query answering is tractable: under both open world and closed world semantics, certain answers to *union of conjunctive queries* (UCQs) can be computed in polynomial time by simply treating nulls as additional usual domain elements, i.e. using *naïve evaluation* that we will study in depth in Chapter 3.

2.2 Incompleteness in relational data exchange: open/closed world semantics

Theoretical foundations of data exchange first developed in [FKMP05, FKP05] adopted an implicit OWA semantics of schema mappings. In such original setting schema mappings are specified by a set Σ of source-to-target *tuple generating dependencies (tgds)* , i.e. first-order dependencies of the form

$$\forall \bar{x}\ \forall \bar{y}\ \big(\varphi_\sigma(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\ \psi_\tau(\bar{x}, \bar{z})\big)$$

(sometimes abbreviated as $\varphi_\sigma(\bar{x}, \bar{y}) \rightarrow \psi_\tau(\bar{x}, \bar{z})$) where φ_σ is a first-order formula over a source vocabulary σ, and ψ_τ is a conjunction of atomic formulae over a target schema τ [FKMP05, Kol05]. Moreover Σ may possibly contain a set of FO constraints on the target schema. In [FKMP05] a data exchange solution for a source instance S is any τ-instance D such that $(S, D) \models \Sigma$ (according to the usual notion of FO satisfaction). We refer to such target instances as *OWA-solutions*.

Note that according to this notion, a solution is allowed to contain tuples not explicitly needed to satisfy the dependencies with the source. This is why this semantics of schema mappings is referred to as *OWA semantics*.

Recall that the goal of query answering in data exchange is to compute certain answers

$$\mathsf{certain}_\Sigma(Q, S) = \bigcap \{Q(D) \mid D \text{ is a solution for } S \text{ under } \Sigma\}$$

by posing a query against a materialized target instance.

The *canonical solution* $\text{CSol}_\Sigma(S)$, for a mapping Σ and a source S (as well as universal solutions in general) was shown to play this role [FKMP05].

As in [ABFL04, HLS11], it is computed essentially by applying a *chase* procedure to the source instance S using the dependencies Σ. In the absence of target constraints it consists in the following procedure. For each dependency $\varphi(\bar{x}, \bar{y}) \to \psi(\bar{x}, \bar{z})$ in Σ and for each pair of tuples $\bar{a}, \bar{b}$ such that $\varphi(\bar{a}, \bar{b})$ holds in S, create a fresh tuple of distinct nulls $\bar{\perp} = \bar{\perp}_{(\varphi,\psi,\bar{a},\bar{b})}$ (so that $|\bar{\perp}| = |\bar{z}|$) and put atoms of the conjunction $\psi(\bar{a}, \bar{\perp})$ as tuples in the target. If the mapping is understood from the context, we write just $\text{CSol}(S)$.

For example, if $\sigma = \{E\}, \tau = \{R\}$, where E and R are binary, and Σ consists of the tgd $E(x, y) \to R(x, z)$, then for $E = \{(a, c_1), (a, c_2), (b, c_3)\}$, the canonical solution has tuples $\{(a, \perp_1), (a, \perp_2), (b, \perp_3)\}$ in R.

Clearly $\text{CSol}(S)$ can be computed in time polynomial in the size of S.

For schema mappings Σ consisting of a set of source-to-target tgds, it turns out that the canonical solution, interpreted under the OWA semantics of incompleteness, represents precisely the set of OWA data exchange solutions, i.e.

$$[\![\text{CSol}_\Sigma(S)]\!]_{\text{OWA}} = \{D \mid D \text{ is a complete OWA-solution for S under } \Sigma\} \quad (2.1)$$

and thus target queries can be answered on the canonical solution. This was implicitly used already in [FKMP05] to find efficient query answering algorithms for UCQs in data exchange.

However [Lib06] was the first to observe that data exchange semantics other than OWA make sense. For instance in a *copying* data exchange setting one deals with dependencies of the form $R(\bar{x}) \to R(\bar{x})$. One may want such a dependency to specify that the target needs to be populated just with the data copied from the source (while observe that under usual FO semantics, any target instance containing the source is a solution). [HLS11] introduced the notion of *CWA-solutions*, and a notion of certain answers based on valuations of such solutions (which we call here *complete CWA-solutions*). Such solutions have "just as much as needed" to satisfy the conditions imposed by the schema mapping. For example, for the copying schema mapping above, the only CWA-solution for a source R would be a copy of R, since instances are no longer open to adding new tuples. We refer to [HLS11] for a formal definition of CWA data exchange solutions.

It turns out that the same canonical solution, interpreted this time under the CWA semantics of incompleteness, captures precisely the set of complete CWA-solutions. I.e. for schema mappings Σ consisting of a set of source-to-target tgds we have the analog of (2.1):

$$[\![\text{CSol}_\Sigma(S)]\!]_{\text{CWA}} = \{D \mid D \text{ is a complete CWA-solution for } S \text{ under } \Sigma\}.$$

Thus query answering in CWA data exchange amounts to answering queries over the canonical solution, under the CWA semantics of incompleteness.

Fully open or fully closed semantics of schema mappings, being two extreme cases, are bound to have their shortcomings.

The main reason for that is that the most suitable data exchange semantics depends on the real-world scenario that is modeled by the schema mapping. There are situations where it is natural to allow in the target other data than the one explicitly transferred from the source, and situations where this should be

avoided. Even within the same schema mapping the most suitable data exchange semantics may be different on different attributes of relations. For example, consider a mapping *Papers(paper#, title)* $\rightarrow$ *Submissions(paper#, author)*. Under the CWA data exchange semantics each paper in the source will be present in the target with exactly one author. This is a limitation; in fact even though we may want the target to represent only the papers present in the source, we do not want to loose the one-to-many relationship between papers and authors.

Motivated by this observations, in [5] we introduced mappings that are not rigidly controlled by the OWA, as in [FKMP05], or by the CWA, as in [HLS11]. To the contrary, the semantics of data exchange can be explicitly specified at the level of attributes, and and is part of the mapping formalism. Open attributes can be instantiated by many values, but for closed, only one value is permitted.

In the rest of Section 2.2 we show that data exchange solutions under the mixed open/closed world semantics can be represented using a suitable model of incompleteness for the target. The connection between query answering in data exchange and querying incomplete data can then be reestablished.

In particular we get the analog of (2.1) by adopting a model of incompleteness introduced in [GZ88] which permits nulls to be *open* or *closed*. This will allow us to analyze the complexity of query answering by studying the complexity of querying the incomplete canonical solution.

2.2.1 Representing solutions under annotated mappings

Annotated mappings An annotated mapping consists of a set of source-to-target tgds with extra annotations *op* or *cl* (for *open* and *closed*) of variables in the target atoms. We refer to our full article [5] for the formal syntax and semantics of such mappings. Here we illustrate them with an example.

Consider a source schema σ with binary relations *Papers(paper#, title)* and *Assignments(paper#, reviewer)*. Each instance of σ represents the list of papers submitted to a given conference and the assignments of papers to reviewers. The target schema τ consists of two binary relations *Reviews(paper#, review)* and *Submissions(paper#, author)*. The mapping between the source and the target is provided by a set of rules below:

$$\begin{array}{lcl} \mathit{Papers}(x,y) & \rightarrow & \mathit{Submissions}(x^{cl}, z^{op}) \\ \mathit{Assignments}(x,y) & \rightarrow & \mathit{Reviews}(x^{cl}, z^{cl}) \\ \mathit{Papers}(x,y) \wedge \neg\exists r\, \mathit{Assignments}(x,r) & \rightarrow & \mathit{Reviews}(x^{cl}, z^{op}) \end{array}$$

Intuitively, the first rule says that the target instance contains exactly the submitted papers from the source (enforced by the closed annotation of the attribute *paper#*). The open annotation of the *author* attribute in the first rule models the one-to-many relationship between papers and their authors. The second rule says that for each assigned paper and each of its reviewers, exactly one review is associated to the paper in the target. Completely closed annotation here prevents the target from having reviews of assigned papers without a corresponding reviewer in the source. The third rule deals with papers that have not been assigned, according to the source. In this case, the attribute *review* of *Reviews* is annotated as open, to allow several reviews to be generated for the same paper.

We remark that atoms of the same relation can be annotated differently in different rules. Indeed, the annotation of an atom of a given target relation R

in a rule describes the way *the particular rule* allows data to be moved from the source to relation R in the target, and this may vary from a rule to a rule.

Open/closed annotations could be an easy addition to systems that handle schema mappings [FHH$^+$09, HHH$^+$05] as they essentially state whether we have a one-to-one or a one-to-many relationship for a correspondence between attributes in the source and the target, and only require one-bit annotations for target attributes.

Solutions under annotated mappings In [5] we provide a notion of data exchange solutions under annotated mappings. These are a form of incomplete instances, and therefore represent a set of complete instances, which will be referred to in this thesis as *complete solutions* under the annotated schema mapping. Certain answers are defined over all possible complete solutions; these are therefore the space of solutions that we need to represent.

The precise definition of solutions under annotated mappings is not particularly relevant in this context, and we refer to our full article [5] for details. Here we concentrate on describing their properties and how they can be represented using a suitable notion of incompleteness.

In particular in [5] we show that solutions under annotated mappings are a natural generalization of OWA- and CWA-solutions, and have these as extremes. Informally this can be stated as follows:

- Complete solutions under annotated schema mappings with fully closed (respectively fully open) annotation coincide with CWA (respectively OWA) complete solutions.
- If the annotation is broadened (i.e. some *closed* annotations are turned to *open*) the set of complete solutions to a given source instance can only augment.

Annotated canonical solution The space of complete solutions under annotated mappings can be described using a form of *annotated incomplete instances*.

This model introduced in [GZ88] extends naïve tables by simply annotating each data value (both constants and nulls) by either *op* or *cl*.

The semantics $[\![I]\!]$ of an annotated incomplete instance I is given by a straightforward generalization of the CWA semantics of incompleteness: a complete instance D is $[\![I]\!]$ if D contains a valuation $v(I)$ of I, and moreover every tuple of D coincides with some tuple of $v(I)$ in all positions annotated by *cl*.

For example, $[\![\{(a^{cl}, \perp^{op})\}]\!]$ contains all binary relations whose projection on the first attribute is $\{a\}$, and $[\![\{(a^{cl}, \perp^{cl})\}]\!]$ contains all one-tuple relations $\{(a, b)\}$ with $b \in \mathsf{Const}$.

The *annotated canonical solution* for a source instance S under an annotated mapping is defined by the same procedure as before for the usual canonical solution, except that now it is populated with annotated tuples. In our previous example with $\sigma = \{E\}, \tau = \{R\}$, let Σ be the annotated tgd $E(x, y) \to R(x^{cl}, z^{op})$. Then, if the source instance S has $E = \{(a, c_1), (a, c_2), (b, c_3)\}$, the canonical solution $\textsc{CSol}_\Sigma(S)$ has annotated tuples $\{(a^{cl}, \perp_1^{op}), (a^{cl}, \perp_2^{op}), (b^{cl}, \perp_3^{op})\}$ in R.

The annotated canonical solution represents the space of data exchange solutions under annotated mappings, i.e.

Proposition 1 *for an annotated mapping Σ and a source instance S*

$$[\![\text{CSol}_\Sigma(S))]\!] = \{D \mid D \text{ is a complete solution for } S \text{ under } \Sigma\}.$$

Thus queries issued on the target can be answered over the polynomial time computable annotated canonical solution:

$$\mathsf{certain}_\Sigma(Q, S) = \mathsf{certain}(Q, \text{CSol}_\Sigma(S)).$$

2.2.2 Querying the canonical solution

We now analyze the complexity of querying the annotated canonical solution. We observe that, in the same way as in the usual (unannotated) setting, the annotated canonical solution is not an arbitrary incomplete instance in its particular model of incompleteness: the fact that it is obtained by chasing the source-to-target dependencies gives a particular structure to it. Thus getting hardness results is in general more difficult for the canonical solution.

We adopt two approaches. First we analyze the complexity of answering full first-order queries, depending on restrictions on the annotation. Then we assume arbitrary annotation and we look for restrictions on the query fragment lowering the complexity.

For the first approach our main result is a *trichotomy*, classifying the complexity of certain answers in terms of the maximum number k of open attributes per atom in a rule of the mapping:

Theorem 1 *The data complexity of computing certain answers to an* FO *query over the annotated canonical solution* $\text{CSol}(S)$ *for an input instance S is:*

- coNP*-complete over mappings with completely closed annotation;*
- coNExpTime*-complete over mappings where each target atom has at most* 1 *open attribute;*
- *undecidable over mappings where the maximum number of open attributes per atom is greater than* 1*.*

We emphasize that in the result above the schema mapping and the query are considered fixed, and lower-bounds are intended to mean that there exists mappings and queries that make the the problem hard.

Undecidability for more than 1 open attribute per rule is an easy consequence of Trakhtenbrot's theorem, as already noticed in [AD98, FKMP05], and coNP-completeness under CWA was shown in [HLS11] by an adaptation of results in [AKG91].

Most of the work goes into the coNExpTime result. Note that this is an unusual result since most commonly computing certain answers tends to fall in the polynomial hierarchy. In fact the coNExpTime upper bound requires new ideas and techniques which are uncommon for this kind of result. The key idea is proving that if there exists an instance witnessing that a tuple is not in the certain answers, then one can construct another witness instance from it, by keeping only an exponential number of values instantiating open attributes. We show that the two instances are equivalent for the query by an Ehrenfeucht-Fraïssé argument. Intuitively the exponential bound is justified by the fact that

we can bound the number of extra values instantiating an open attribute, based on the subset of tuples they occur with, and there are exponentially many such subsets.

The proof crucially relies on the fact that each tuple of the annotated instance contains a single open attribute. To the contrary it does not rely on the structure of the canonical solution, and holds for arbitrary annotated instances with at most one open attribute per tuple.

The lower bound is obtained by an encoding of the tiling problem. Open attributes here are used to encode subsets of an input set, thus representing the coordinates of an exponential size grid.

As a second approach, we show how lower complexity can be achieved by putting restrictions on queries (but leaving the annotation arbitrary).

One easy observation is that monotone queries do not distinguish between OWA, CWA, or any semantics in-between. Starting from this observation we could get a picture of the data complexity of answering monotone queries. Moreover by relying on techniques similar to the decidability of the Schönfinkel-Bernays class, we could also show that, even beyond monotone queries, the complexity of query answering can go down to coNP for arbitrary annotations. Our results can be summarized as follows:

Proposition 2 *For arbitrary annotated mappings, the data complexity of computing certain answers to a query over the canonical solution* $\text{CSol}(S)$ *for an input instance* S *is:*

- coNP *if the query is monotone and polynomial time computable over complete instances. And in particular it is*
  - PTIME *if the query is an existential positive* FO *query (i.e. a union of conjunctive queries); in this case certain answers are computed by naïve evaluation;*
  - coNP*-complete already for conjunctive queries with two inequalities;*
- coNP *if the query is in the* $\forall^*\exists^*$ *fragment of* FO.

Several extensions of our results can be obtained. First the trichotomy theorem is true for any query language of PTIME data complexity that contains FO. Second, if we allow 1-to-m relationships in place of 1-to-many relationships and define such limited open nulls (i.e. each such null can be replicated at most m times), then all the complexity results about CWA mappings apply to this case.

The complexity analysis we have presented can also be refined in several ways. In fact remark that we have looked at how the complexity of query answering can be lowered either by restricting the annotation, for all FO queries, or by restricting the query fragment, for all annotations. However the two restrictions could be combined, and tractable cases can in principle be found for different query fragments depending on restriction on the annotation.

This is the type of analysis we conduct in Chapter 3 for general models of incompleteness. Our analysis will provide general tools to study how tractable query fragments depend on the semantics of incompleteness. From results presented in Chapter 3 we can immediately derive for instance that the tractability

case of Proposition 2 can be extended to a larger fragment allowing universal quantification and a limited form of negation, if annotations are all *cl*.

Although we did not instantiate the general framework presented in Chapter 3 on the annotated model of incompleteness, this is in principle possible.

2.3 Representing and querying incomplete XML

XML data exchange and integration applications have brought the attention to *tree patterns* as a form of incomplete description of trees [AL08, ALM09].

A tree pattern presents a *partial* description of a tree, along with some variables that can be assigned values as a pattern is matched to a complete document. For instance, a pattern $a(x)[b(x), c(y)]$ describes a tree with the root labeled a and two children labeled b and c; these carry data values, so that those in the a-node and the b-node are the same. This pattern matches for example a tree with root a and children b and c with all of them having data value 1; not only that, such a match produces the tuple $(1, 1)$ of data values for (x, y) witnessing the match. A tree pattern can also specify more general relationships between tree nodes, for instance only requiring that the b node is a descendant of the a node in the tree.

In a typical XML schema mapping [AL08] we are given two automata $\mathcal{A}_s$ and $\mathcal{A}_t$, describing source and target schemas respectively. The correspondence between them is provided by a set of dependencies $\pi_s(\bar{x}, \bar{y}) \rightarrow \pi_t(\bar{x}, \bar{z})$, where π_s and π_t are tree patterns. Whenever the source pattern can be matched into the source tree with values $(\bar{a}, \bar{b})$, the target solution needs to match π_t with values $(\bar{a}, \bar{c})$ for some $\bar{c}$.

From this scenario is is easy to see that incompleteness in the target can occur at two levels: missing data values (corresponding to existentially quantified variables in the schema mapping) and missing information about the structure of the target tree. The latter is due to the fact that target patterns are incomplete descriptions themselves, and moreover schema mappings provide no information on how different tree portions satisfying the dependencies are connected among them in the target (remark that this is not a source of incompleteness relational data exchange).

Remark also that patterns we deal with in data exchange are naturally tree-shaped. This is in contrast with some of the patterns appearing in the literature [BMS08, BMS11] that can take the shape of arbitrary graphs (for instance, such a pattern can say that we have an a-node, that has b and c descendants, that in turn have the same d-descendant: this describes a directed acyclic graph rather than a tree). It is also natural to use such patterns for defining queries [Dav08, AYCLS02, MS04].

In the rest of this section we informally present the pattern-based model of incomplete XML we have proposed in [6]. It is based on a combination of nulls and missing structural information. First in Section 2.3.1 we review the main features that allow us to classify incomplete tree descriptions. Then in Section 2.3.2 we study how these features influence the complexity of query answering. As mentioned in Chapter 1, we have studied other computational problems of interest, such as consistency of incomplete tree descriptions and

membership in the semantics, which we do not review here. We refer to our full article [6] and to our survey [4] for a detailed comprehensive study of several computational problems over incomplete tree descriptions.

2.3.1 Incomplete tree descriptions

Complete XML trees are for us node-labelled ordered trees, where each node has associated: 1) a unique id and 2) a set of attributes, each with a data value associated.

To model incompleteness in XML, we start with complete trees and see how missing information can be incorporated into them. A first thing that can be missing is attribute values, i.e. we can have nulls (i.e. data variables) replacing constant attribute values. In addition to them, the following structural information can be missing too:

(a) node ids (they can be replaced by node variables);

(b) node labels (they can be replaced by wildcards _ which match any label);

(c) precise vertical relationship between nodes (we can use descendant edges $\downarrow^*$ instead of child edges $\downarrow$);

(d) precise horizontal relationship between nodes (using younger-sibling edges $\rightarrow^*$, or no edges at all, instead of next-sibling $\rightarrow$).

In both (c) and (d), we may allow partial information to be recovered: the incomplete description can in addition specify that a node is a leaf (by adding a *leaf* marking to the node), or the root (*root* marking) or that it is a first child (*fc* marking) or a last child (*lc* marking).

Incomplete tree descriptions will be graphically represented as trees using the type of edges mentioned above; we refer to our full article [6] for a precise syntax. Two examples of incomplete tree descriptions are given in Figure 2.1, where each leaf has a single attribute value. In Figure 2.1(a) node ids are all constants, shown in parentheses as (i_k). In Figure 2.1(b), node ids are all pairwise distinct variables and are omitted in the picture.

Each of the documents in the picture may represent many complete trees. For instance both documents may either represent the single "*Foundations of databases*" book with authors "*Abiteboul*","*Hull*" and "*Vianu*" or two distinct books: the first one being the "*Foundations of databases*" book, and the second one having only "'*Vianu*" as an author. Observe that we implicitly assume the OWA and allow addition of nodes; in particular the incomplete documents in the picture do not bring any information about the author "*Hull*".

Semantics More precisely the semantics of incomplete tree descriptions is based on homomorphisms on trees. A *homomorphism* from a (possibly incomplete) tree π_1 to another one π_2 is a mapping from nodes of π_1 to nodes of π_2 and from data values of π_1 to data values of π_2 which

- is the identity on constant ids and constant data values;
- preserves node labels, attribute values, markings and tree edges (where edges of π_2 are assumed augmented with the reflexive-transitive closure of $\downarrow \cup \downarrow^*$ and of $\rightarrow \cup \rightarrow^*$).

Given an incomplete tree description π, a complete tree T is in the semantics $[\![\pi]\!]$ iff there exists a homomorphism from π to T.

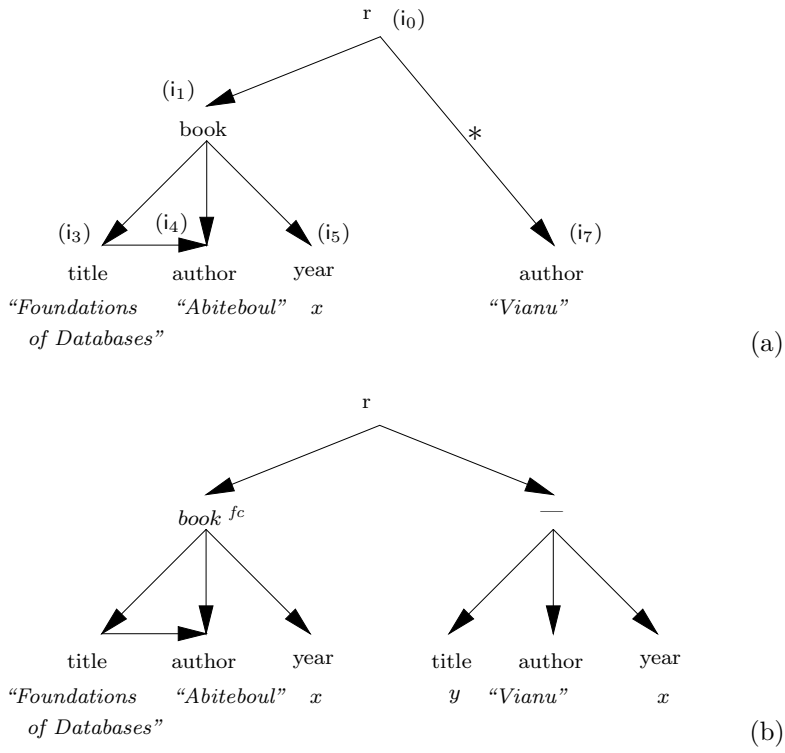


Figure 2.1: Two incomplete tree descriptions

Classification Incomplete descriptions where node ids are all constants (as in Figure 2.1(a)) correspond to the XML DOM interface [DOM04], where we can access each node in a document by its id; they will be referred to as *incomplete DOM trees*. On the other hand tree descriptions where node ids are all pairwise distinct variables (as in Figure 2.1(b)) will be referred to as *incomplete trees*.

The assumption on constant vs. variable node ids has a big impact on the semantics of incomplete tree descriptions. For example in the document of Figure 2.1(b), if node ids were all distinct constants it would no longer be possible that the document represents a single book, as before. Indeed, we know that the two children of the root are different, since their ids are distinct.

There are other parameters that influence the semantics and computational properties of incomplete descriptions. In our analysis we consider:

- *node ids* (i.e. the distinction incomplete trees/incomplete DOM trees);
- whether or not *attribute values* are allowed;
- *structural parameters*, that is: the set of axes used in tree descriptions, possible markings on nodes, and possible missing information about the sibling order (i.e. whether the absence of horizontal edges is allowed);
- the possible presence of *schema information* such as DTDs, restricting the allowed structure of trees.

2.3.2 Querying incomplete tree descriptions

We look for classes of queries and incomplete representations that admit tractable query evaluation for computing certain answers. To this end we deal with conjunctive queries over trees and their unions, motivated by the fact that at least in the relational setting they can be efficiently evaluated using naïve evaluation. We consider a version of them that outputs tuples of values (this, of course, includes Boolean queries). Queries outputting trees have been later considered in [DLM10].

Over trees conjunctive queries are essentially standard (see, e.g., [BMS11, GKS06]). In the same way as in the relational setting – where there is a duality between incomplete relations and conjunctive queries, via their *tableau* representation – we can express conjunctive queries in our syntax for incomplete trees: a *conjunctive query* over trees is just an incomplete tree with some distinguished (data) variables (the other variables for data values as well as the node ids are all implicitly considered existentially quantified). Over a complete tree T such a conjunctive query Q produces the tuples of values of its free variables in the homomorphisms from Q to T. This query result is denoted by $Q(T)$. We consider unions of such conjunctive queries.

A fragment of the language, namely union of conjunctive queries using only downward axes, was considered in the study of query answering in XML data exchange [AL08].

Certain answers for conjunctive queries and their unions over an incomplete tree description π are defined as usual:

$$\mathsf{certain}(Q, \pi) = \bigcap \{Q(T) \mid T \in [\![\pi]\!]\}$$

Although both conjunctive queries and incomplete tree descriptions have a natural relational representation, computing certain answers is not a special case

of querying naïve tables, since we have the additional constraint that structures need to be trees. We can prove the following upper bound:

Theorem 2 *The data complexity of computing certain answers of unions of conjunctive queries over incomplete tree descriptions is* coNP*. This also holds under the presence of DTDs constraining the semantics of incomplete descriptions.*

This result is proved by using a "cutting technique" which eliminates portions of a witnessing tree which are not needed to violate the query, to satisfy the incomplete representation, and possibly the DTD. (Note that this upper bound was extended to boolean combinations of conjunctive queries in [GLT12], which also provided a Π_2^p upper bound for combined complexity.)

However in contrast with the relational case, answering union of conjunctive queries over incomplete tree descriptions is not always tractable.

We could easily rule out some features which immediately lead to intractability. These are essentially of two natures. First we proved that *the presence of schema information (DTDs) and markings lead easily to coNP-hardness of query answering, already for syntactically trivial conjunctive queries, and for both incomplete trees ad incomplete DOM-trees.*

This is intuitively due to the fact that both DTDs and markings allow us to express constraints on the set of possible worlds which force different nodes of the tree description to collapse.

When we rule out these features, query answering may remain hard, if one allows any form of *missing structural information* (note that below we say that data complexity of a class of queries is coNP-hard if there exists a query from that class whose data complexity is coNP-hard):

Theorem 3 *The data complexity of computing certain answers of unions of conjunctive queries over incomplete trees is* coNP*-complete if, besides child and next-sibling axes, one allows any of the following structural features in incomplete trees:*

1. *descendant axis* $\downarrow^*$
2. *younger sibling axis* $\rightarrow^*$
3. *possibly missing sibling order*

The first two hardness results also hold for incomplete DOM-trees.

When excluding all these features we get a robust tractable class of incomplete trees with respect to query answering. That is, in order to have tractability we restrict ourselves to incomplete trees where the order of siblings is completely specified, with no transitive closures of axes nor markings. We call them *rigid incomplete trees.*

We showed that unions of conjunctive queries can be evaluated naïvely over rigid incomplete trees. *Naïve evaluation* is defined as follows: each disjunct Q of the query is evaluated directly on the incomplete tree π, as if π were complete (i.e. one takes the value of the free variables of Q in all homomorphisms from Q to π, where as usual π is suitably extended with the reflexive-transitive closures of its downward and forward edges). Then only null-free tuples are kept in the result.

Theorem 4 *Let π be a rigid incomplete tree, and Q a union of conjunctive queries over trees that does not use markings. Then the naïve evaluation of Q over π computes* $\mathsf{certain}(Q, \pi)$. *In particular, evaluating no-marking queries over rigid incomplete trees has* DLogSpace *data complexity.*

We remark that it was later proved in [DLM10] that this result still holds over incomplete trees which are just downward rigid, provided that the query does not mention horizontal axes.

For boolean combinations of conjunctive queries naïve evaluation no longer works. Nonetheless, a more complex tractable algorithm over rigid incomplete trees was devised in [GLT12].

Rigid incomplete trees are then a robust class for answering union of conjunctive queries (and boolean combinations as well). The case of incomplete DOM trees is quite different.

While Theorem 4 also holds for rigid incomplete DOM trees, remark that in Theorem 3 missing sibling order information does not immediately bring to intractability for incomplete DOM trees.

Indeed we were able to push tractability boundaries further, and show that conjunctive queries given by a form of word patterns can be answered in polynomial time over incomplete DOM trees which are not completely rigid (i.e. they extend rigid trees by admitting missing sibling information). However this is done at the expense of algorithms which are significantly more complicated than naïve evaluation.

The term *word patterns* below refers to conjunctive queries given by depth-1 attribute-free incomplete trees whose structure only allows child and next-sibling axes, and possibly missing sibling order.

Theorem 5 *Certain answers to word patterns can be computed in polynomial time data complexity over incomplete DOM trees whose structure allows only child, next-sibling axes, and possibly missing sibling order.*

Observe that to the contrary word patterns are already coNP-hard to evaluate over the same class of incomplete trees (i.e. when dropping the DOM assumption).

This result suggests that incomplete DOM trees can be be queried more efficiently than incomplete trees, however the tractability frontier for incomplete DOM trees is not completely clear yet.

Theorem 5 is also of independent interest in the theory of *string pattern matching*. In [7] we deal with the problem of deciding whether a given set of string patterns implies the presence of a fixed pattern. While checking whether a set of patterns occurs in a string is solvable in polynomial time, this implication problem is well known to be intractable. In [7] we consider a version of the problem where patterns in the set are required to be disjoint (the analog of the DOM assumption). We show that for such a version of the problem the situation is reversed: checking whether a set of patterns occurs in a string is NP-complete, but a special case of Theorem 5 shows that the implication problem is solvable in polynomial time.

Chapter 3

Naïve evaluation: a general framework

Query answering solutions over incomplete data depend on several aspects: the adopted model of incompleteness (such as naïve tables, annotated instances, different fragments of incomplete tree descriptions, etc.) as well as the semantics of incompleteness, i.e. the way missing information is interpreted, and the query language. In the previous chapter we have found ad-hoc tractable solutions for different combinations of those. However one can find many commonalities to all these scenarios, both in the objects one deals with and in the underlying techniques. In particular tractable solutions are often based on a form of naïve evaluation, which consists in evaluating the query directly on the incomplete data, by "ignoring" in a way its incompleteness. In this chapter we embark on the development of a general framework for studying query answering solutions based on naïve evaluation. Our framework, developed in [1], encompasses many data models and semantics, and can be instantiated to a variety of them.

3.1 Naïve evaluation

Tractable solutions for querying incomplete data have been studied in the literature particularly in connection with *representation systems* ([IL84, AKG91]). In this setting, if Q is a query and I an incomplete instance, the problem is finding a compact representation of all possible answers $\{Q(D) \mid D \in [\![I]\!]\}$. One possible approach is to find another incomplete instance U which represents exactly this set, i.e. $[\![U]\!] = \{Q(D) \mid D \in [\![I]\!]\}$. When this is always possible the model of incompleteness and the query language are said to form a *strong representation system*. This is a very strong requirement and often not needed. In data exchange for instance one usually does not need to represent the set of all query answers but often only compute the *certain information* in this set, i.e. $\bigcap_{D \in [\![I]\!]} Q(D)$ which coincides with $\mathsf{certain}(Q, I)$. The requirement can then be relaxed : one needs to find an incomplete instance U such that $[\![U]\!] \approx \{Q(D) \mid D \in [\![I]\!]\}$, where equivalence is intended to mean that the two sets have the same certain information. If this is the case the certain information in $[\![U]\!]$ provides precisely $\mathsf{certain}(Q, I)$. (Note that, in order for guaranteeing compositionality of the query language, this requirement is

strengthened in [IL84] by requiring that $[\![U]\!]$ and $\{Q(D) \mid D \in [\![I]\!]\}$ not only contain the same certain information, but are equivalent for computing certain answers to all queries of the language. When such a U can be found for all I and all Q, the model of incompleteness and the query language are said to form a *weak representation system*.)

It turns out that for simple models of incomplete information such as naïve tables and union of conjunctive queries, under the OWA, such U can always be found and most importantly:

1) U can be computed with the usual query answering algorithms, since it coincides with $Q(I)$, i.e. the result of evaluating the query directly over the incomplete instance I, by considering nulls just as extra domain elements;

2) the certain information in $[\![U]\!]$ can be efficiently computed : it just consists of the tuples of U containing no nulls.

The two-step procedure consisting in computing $Q(I)$ first and then removing tuples with nulls has been referred to as *naïve evaluation* of the query Q on the incomplete instance I.

To give an example, consider the following two incomplete relational instances:

R:

A	B
1	$\perp_1$
$\perp_2$	$\perp_3$

E:

B	C
$\perp_1$	4
$\perp_3$	5

Suppose we have a conjunctive query $\pi_{AC}(R \bowtie E)$ or, equivalently, $\varphi(x, y) = \exists z \ \big(R(x, z) \wedge E(z, y)\big)$. Naïve evaluation says: evaluate the query directly on R and E, proceed as if nulls were usual values; they are equal only if they are syntactically the same (for instance $\perp_1 = \perp_1$ but $\perp_1 \neq \perp_2$, and $\perp_1 \neq c$ for every $c \in \mathsf{Const}$). Thus evaluating the above query results in two tuples: $(1, 4)$, and $(\perp_2, 5)$. Then tuples with nulls are eliminated from the result, so we only keep the tuple $(1, 4)$. Note that if Q is a Boolean query, the second step is unnecessary.

We say that *naïve evaluation works for* Q (under semantics $[\![\,]\!]$) if its result coincides with $\mathsf{certain}(Q, I)$ under $[\![\,]\!]$, for every incomplete instance I.

Naïve evaluation is in general efficient (polynomial time in data complexity for all FO queries over relational instances). It follows that, whenever naïve evaluation works, certain answers can be efficiently computed. Moreover it could be in principle directly implemented and benefit from the whole set of classical optimization techniques already in use in database systems.

3.2 Naïve evaluation and syntactic fragments of queries

In general naïve evaluation need not compute certain answers. For the query above, the tuple $(1, 4)$ is however the certain answer, under the common open world semantics. This is true because, as mentioned in Section 3.1, [IL84] showed that if Q is a union of conjunctive queries, then naïve evaluation works for it, under the OWA. It is natural to ask how much the language of UCQs can be extended by retaining this property. It turns out that in some cases this is not possible: [Lib11] showed that under the OWA, if naïve evaluation

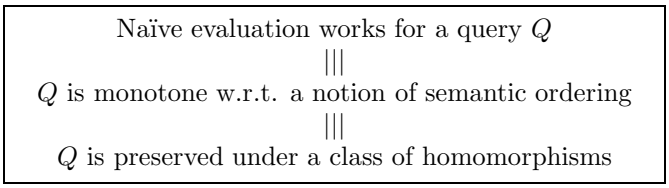
works for a Boolean first-order query Q, then Q must be equivalent to a union of conjunctive queries. That result crucially relied on a preservation theorem from mathematical logic [CK90], and in particular on its version over finite structures [Ros08].

Fact 1 ([IL84, Lib11]) *Let Q be a union of conjunctive queries. Then naïve evaluation works for Q under both OWA and CWA. Moreover, if Q is a Boolean* FO *query and naïve evaluation works for Q under OWA, then Q is equivalent to a union of conjunctive queries.*

The last equivalence result only works under the OWA semantics. Consider the instance $R = \{(\bot, \bot'), (\bot', \bot)\}$ and a query $\exists x, y\ (R(x, y) \wedge R(y, x))$. The certain answer to this query is true under both OWA and CWA, and indeed it evaluates to true naïvely over R. On the other hand, a query Q given by $\forall x \exists y\ R(x, y)$ (not equivalent to a union of conjunctive queries) evaluated naïvely, returns true on R, but under OWA its certain answer is false. However, under CWA, its certain answer is true. This is not an isolated phenomenon: we will later see that Q belongs to a class, extending unions of conjunctive queries, for which naïve evaluation works under CWA on all databases.

In [1] we formally investigated the applicability of naïve evaluation to querying incomplete data in a very general framework. This allowed us to reveal the most general properties underlying this procedure.

Roughly, our results can be seen as establishing the following equivalences



together with finding syntactic classes of queries guaranteeing preservation under homomorphisms (and therefore naïve evaluation). In fact preservation theorems characterize syntactically preservation properties of queries. The scheme above can therefore be used to find syntactic query fragments where naïve evaluation is possible.

We applied our framework to several concrete semantics of incompleteness on relational databases, coming from different application scenarios such as deductive databases, data exchange and integration, programming semantics etc. The framework is general enough to be applied to other models of incomplete data such as incomplete trees or graphs, which we leave as future work.

We now explain our main results and the key ideas behind them. We present them for Boolean queries; in [1] we show that these still hold for arbitrary k-ary queries, and queries with constants as well.

3.2.1 Naïve evaluation and monotonicity

We show that naïve evaluation is often related to some "monotonicity" properties of queries. We deal with a very abstract setting, where databases are just objects of a domain equipped with a notion of semantics. This is formalized as follows:

Definition 1 (Database domain) *A database domain is a structure* $\mathbb{D} = \langle \mathcal{D}, \mathcal{C}, [\![\,]\!], \approx \rangle$*, where* $\mathcal{D}$ *is a set,* $\mathcal{C}$ *is a subset of* $\mathcal{D}$*, the function* $[\![\,]\!]$ *is from* $\mathcal{D}$ *to nonempty subsets of* $\mathcal{C}$*, and* $\approx$ *is an equivalence relation on* $\mathcal{D}$*.*

The interpretation is as follows:

- $\mathcal{D}$ is a set of database objects;
- $\mathcal{C}$ is the set of complete objects in $\mathcal{D}$;
- $[\![x]\!] \subseteq \mathcal{C}$ is the semantics of an incomplete database object x, i.e., the set of all complete objects that x can represent;
- $\approx$ is the structural equivalence relation, that we need to describe the notion of generic queries.

For instance in the relational setting, $\mathcal{D}$ represents the set of all incomplete relational instances, $\mathcal{C}$ the ones without nulls, the semantics $[\![\,]\!]$ is a relational incompleteness semantics (for instance $[\![\,]\!]_{\text{OWA}}$ or $[\![\,]\!]_{\text{CWA}}$), and $\approx$ is the isomorphism relation of relational instances. These will be referred to as *relational database domains* in the sequel.

Of course there could be many non-relational database domains of interest, for instance, all XML documents of a given schema or all graph databases over a fixed labeling alphabet.

A *query* over $\mathbb{D}$ is a mapping $Q : \mathcal{D} \to \{0, 1\}$. We use 0 to represent *false* and 1 to represent *true*, as usual. A query is *generic* if $Q(x) = Q(y)$ whenever $x \approx y$.

For each $x \in \mathcal{D}$, the certain answer is

$$\mathsf{certain}(Q, x) = \bigwedge \{Q(c) \mid c \in [\![x]\!]\}$$

We can now reformulate the notion of naïve evaluation in this abstract setting:

Definition 2 *Over a database domain* $\mathbb{D} = \langle \mathcal{D}, \mathcal{C}, [\![\,]\!], \approx \rangle$ naïve evaluation works *for a query* Q *if* $Q(x) = \mathsf{certain}(Q, x)$ *for every* $x \in \mathcal{D}$*.*

We remark that for a relational database domain and a Boolean relational query the above definition specifying when naïve evaluation works goes back to the corresponding notion introduced in Section 3.1 for relational instances.

We will need to impose an additional property on database domains saying, essentially, that there are enough complete objects. A database domain $\mathbb{D} = \langle \mathcal{D}, \mathcal{C}, [\![\,]\!], \approx \rangle$ is *saturated* if every object has a complete object in its semantics which is isomorphic to it: that is, for each $x \in \mathcal{D}$ there is $y \in [\![x]\!]$ such that $x \approx y$.

For most common data models and semantics (such as OWA and CWA over naïve tables) this condition holds.

We say that a query Q over $\mathbb{D}$ is *monotone* w.r.t $[\![\,]\!]$ if

$$y \in [\![x]\!] \quad \Rightarrow \quad Q(x) \leq Q(y).$$

Remark that, even over relational database domains, this property defines different monotonicity notions depending on the semantics, and should not be confused with usual monotonicity of queries w.r.t the sub-instance relation.

We are now ready to state the promised connection, which easily derives from the saturation property and the genericity of queries:

Theorem 6 *Let $\mathbb{D}$ be a saturated database domain, and Q a generic Boolean query over $\mathbb{D}$. Then naïve evaluation works for Q iff Q is monotone w.r.t $[\![\ ]\!]$.*

3.2.2 Monotonicity and preservation

We next connect monotonicity with preservation. To do this we restrict to the relational setting and use a notion of relational homomorphisms.

By inspecting several relational semantics (see Chapter 2, Section 2.1) it is easy to realize that they are all instances of the following scheme. If $\mathcal{R}$ is a reflexive binary relation between complete relational instances, the semantics $[\![\ ]\!]_{\mathcal{R}}$ is defined as follows:

for all databases I, D

$$D \in [\![I]\!]_{\mathcal{R}} \quad \Leftrightarrow \quad \text{there is a valuation } v \text{ such that } (v(I), D) \in \mathcal{R}.$$

It is in fact straightforward to verify that $[\![\ ]\!]_{\mathcal{R}}$ coincides with $[\![\ ]\!]_{\text{OWA}}$ if $\mathcal{R}$ is $\subseteq$, with $[\![\ ]\!]_{\text{CWA}}$ if $\mathcal{R}$ is $=$, and with $[\![\ ]\!]_{\text{WCWA}}$ if $\mathcal{R} = \{(D, D') \mid D \subseteq D' \text{ and } \text{adom}(D) = \text{adom}(D')\}$.

We now show that monotonicity of a query Q corresponds to preservation under homomorphisms that respect relation $\mathcal{R}$:

Definition 3 ($\mathcal{R}$-homomorphism and preservation) *For complete databases D and D', a mapping h defined on the active domain of D is an* $\mathcal{R}$-homomorphism *from D to D' if $(h(D), D') \in \mathcal{R}$.*

Remark that directly by definition, $\mathcal{R}$-homomorphisms are: usual homomorphisms if $\mathcal{R}$ is $\subseteq$, *strong onto* homomorphisms if $\mathcal{R}$ is $=$, and *onto* homomorphisms if $\mathcal{R}$ is the relation $\{(D, D') \mid D \subseteq D' \text{ and } \text{adom}(D) = \text{adom}(D')\}$.

Clearly $\mathcal{R}$-homomorphisms "mimic" the semantic mapping between instances I and $D \in [\![I]\!]_{\mathcal{R}}$. However valuations used in this mapping are not as general as homomorphisms, since they are bound to be the identity on some domain elements (the constants). On the other hand our queries are generic; this implies that they cannot distinguish constants from nulls.

Based on this idea we can prove the desired correspondence between monotonicity of queries and preservation properties:

Proposition 3 *If Q is a generic Boolean relational query, then Q is monotone w.r.t $[\![\ ]\!]_{\mathcal{R}}$ iff it is preserved under $\mathcal{R}$-homomorphisms.*

This immediately gives a corollary relating naïve evaluation and preservation for general and specific relational semantics.

Corollary 1 *If Q is a generic Boolean relational query, naïve evaluation works for Q under $[\![\ ]\!]_{\mathcal{R}}$ if and only if Q is preserved under $\mathcal{R}$-homomorphisms.*

In particular naïve evaluation works for Q under:

- *OWA iff Q is preserved under homomorphisms.*
- *CWA iff Q is preserved under strong onto homomorphisms.*
- *WCWA iff Q is preserved under onto homomorphisms.*

3.2.3 Preservation properties and and syntactic classes of queries

We have so far established that naïve evaluation is captured by preservation under a class of homomorphisms. Such preservation results are classical in mathematical logic [CK90], and thus we would like to use them, at least as sufficient conditions, to find syntactic classes of queries for which naïve evaluation works.

Positive and existential positive formulae Recall that *positive* formulae use all the FO connectives except negation (i.e., $\wedge, \vee, \forall, \exists$). Formally, the class Pos of positive formulae is defined inductively as follows:

- *true* and *false* and every positive atomic formula (i.e., $R(\bar{x})$ or $x = y$) are in Pos;
- if $\varphi, \psi \in \mathsf{Pos}$, then $\varphi \vee \psi$, $\varphi \wedge \psi$, $\exists x\varphi$ and $\forall x\varphi$ are in Pos.

If $\forall x\varphi$ formulae are further excluded from the class, we obtain the class $\exists\mathsf{Pos}$ of *existential positive formulae*, i.e. unions of conjunctive queries.

Rossman's theorem [Ros08] says that an FO sentence φ is preserved under homomorphisms over finite structures iff φ is equivalent to a sentence from $\exists\mathsf{Pos}$. Lyndon's theorem [CK90] says that an FO sentence φ is preserved under onto homomorphisms (over arbitrary structures) iff φ is equivalent to a sentence from Pos. Lyndon's theorem fails in the finite [AG87, Sto95], but the implication from being positive to preservation is still valid.

A characterization of preservation under strong onto homomorphisms was stated in [Kei65a, Kei65b], but the syntactic class had a rather complex definition and was limited to a single binary relation. Even worse, we discovered a gap in one of the key lemmas in [Kei65b]. So instead we propose a simple extension of positive formulae that gives preservation under strong onto homomorphisms.

Extensions with universal guards The fragment $\mathsf{Pos}{+}\forall\mathsf{G}$, whose definition is inspired by [Com83], extends Pos with the following formation rule based on universal guards:

- if $\varphi(\bar{x}, \bar{y})$ is in $\mathsf{Pos} + \forall\mathsf{G}$, and R is an n-ary relation symbol, then the formula $\forall x_1, \ldots, x_n\big(R(x_1, \ldots, x_n) \rightarrow \varphi(x_1, \ldots, x_n, \bar{y})\big)$ is in $\mathsf{Pos} + \forall\mathsf{G}$ if $x_1, \ldots, x_n$ are pairwise distinct variables; (R can be the equality predicate if $n = 2$).

Note also that the first two rules are the same as for Pos, so we have $\exists\mathsf{Pos} \subsetneq \mathsf{Pos} \subsetneq \mathsf{Pos} + \forall\mathsf{G}$.

The difference between Pos and $\mathsf{Pos} + \forall\mathsf{G}$ is emphasized in the following example, which also witnesses the strict inclusion $\mathsf{Pos} \subsetneq \mathsf{Pos} + \forall\mathsf{G}$.

Consider a sentence $\varphi = \forall x, y\ (R(x, y) \rightarrow E(x))$. Clearly φ is in $\mathsf{Pos} + \forall\mathsf{G}$. However φ is not in Pos since it is not preserved under onto homomorphisms (while all formulae of Pos are).

In fact consider databases D and D' so that R is interpreted as $\{(1, 2)\}$ in D, as $\{(1, 2), (2, 1)\}$ in D', and E is interpreted as $\{(1)\}$ in both. Clearly D has an onto homomorphism h to D' (which is the identity) and $D \models \varphi$. However $D' \models \neg\varphi$ because $E(2)$ does not hold in D'. Intuitively this is due to the fact

that an onto homomorphism from D to D' "preserves" the domain of D but not its facts. New facts (such as $R(2,1)$) can be present in D'. Thus if the guard is satisfied in D', it need not be satisfied in D, and this is why satisfaction of φ may fail in D'. Indeed observe that if the fact $R(2,1)$ were in D, then $E(2)$ would hold in D as well (by satisfaction of φ), and therefore in D' (because the formula $E(x)$ is clearly preserved under (onto) homomorphisms).

In view of this example, the fact that strong onto homomorphisms disallow new facts in the target instance intuitively explains the following proposition, which is proved by structural induction.

Proposition 4 *Sentences in* $\mathsf{Pos} + \forall\mathsf{G}$ *are preserved under strong onto homomorphisms.*

We now combine all the previous implications (preservation $\rightarrow$ monotonicity $\rightarrow$ naïve evaluation) to show that naïve evaluation can work beyond unions of conjunctive queries under realistic semantic assumptions.

Theorem 7 *Let Q be a Boolean FO query. Then:*

- *If Q is in $\exists\mathsf{Pos}$, then naïve evaluation works for Q under OWA.*
- *If Q is in Pos, then naïve evaluation works for Q under WCWA.*
- *If Q is in $\mathsf{Pos} + \forall\mathsf{G}$, then naïve evaluation works for Q under CWA.*

3.3 Moving beyond the standard semantics

In the previous section we have developed our approach to naïve evaluation for standard relational semantics. We show now that there are other possible semantics for which the approach works. These, in general, are obtained by using (separately, or together) two ideas.

The first idea is giving up saturation, i.e., the condition that every object x must have an isomorphic object y in its semantics. The second idea is giving up uniqueness of valuation of nulls.

3.3.1 Giving up saturation: minimal semantics

Recall that Theorem 6 relates naïve evaluation and monotonicity by relying on the saturation property. Over arbitrary non-saturated database domain we can recover a connection between naïve evaluation and monotonicity by only requiring the existence of a saturated sub-domain of a particular type.

If $\mathbb{D} = \langle \mathcal{D}, \mathcal{C}, [\![\,]\!], \approx \rangle$ is a database domain, and $\mathcal{C} \subseteq \mathcal{S} \subseteq \mathcal{D}$ then $\langle \mathcal{S}, \mathcal{C}, [\![\,]\!], \approx \rangle$ is a *subdomain* of $\mathbb{D}$. As usual it is saturated if every object in $\mathcal{S}$ has a complete object in its semantics that is isomorphic to it. Moreover the subdomain is *representative* if for each $x \in \mathcal{D}$ there exists an object in $y \in \mathcal{S}$ (called its *representative*) having $[\![y]\!] = [\![x]\!]$.

In all the examples encountered so far we had $\mathcal{S} = \mathcal{D}$, but this need not always be the case. However, we have the following generalization of Theorem 6.

Theorem 8 *Let $\mathbb{D}$ be a database domain with a representative saturated subdomain, and Q a generic Boolean query over $\mathbb{D}$. Then naïve evaluation works for Q iff Q is monotone w.r.t the semantics and does not distinguish an objet of $\mathcal{D}$ from its representative.*

We shall next see a concrete non-saturated relational semantics where representative instances are a well known object, namely *cores* of structures (cf. [HN92]).

On an incomplete relational instance I this semantics only allows valuations v that are *minimal*, i.e. there exists no valuation v' such that $v'(I) \subsetneq v(I)$.

Consider, for instance, an incomplete database $I = \{(\bot, \bot), (\bot, \bot')\}$ and a valuation $v(\bot) = 1,\ v(\bot') = 2$. The result $v(I)$ is not the smallest possible: take for instance $v'(\bot) = v'(\bot') = 1$ and we have $v'(I) \subsetneq v(I)$. The set $v'(I)$ is minimal, i.e., not a proper subset of any other valuation.

Thus the semantics we deal with now is

$$[\![I]\!]^{\min}_{\text{CWA}} = \{v(I) \mid v \text{ is a minimal valuation}\}.$$

(We can actually also introduce an arbitrary semantic relation $\mathcal{R}$ following the valuation, as with usual relational semantics, and all our results continue to hold.)

This can be viewed as a very strong form of the closed world assumption. Alternatively, it can be viewed as a building block of a more relaxed notion of closed world, originating in the field of logic programming [Min82] and used in the data exchange scenario [Her11].

The following lemma implies that Theorem 8 can be applied to minimal semantics.

Lemma 1 *For the semantics $[\![\,]\!]^{\min}_{\text{CWA}}$, the set of cores is a representative saturated subdomain, where the representative of each instance is its core.*

To prove Lemma 1 we investigated the non-obvious relationship between minimal valuations and cores. It turns out that images of minimal valuations cannot be directly described in terms of cores. In fact, by strengthening results of [Her11], we can show that not all valuations of a core are minimal. This shows that the exist instances I such that $[\![I]\!]^{\min}_{\text{CWA}} \neq [\![\textsc{Core}(I)]\!]_{\text{CWA}}$. However we proved that valuations of cores which are isomorphisms must be minimal, and an instance and its core have the same images of minimal valuations. This easily implies Lemma 1.

Thanks to this lemma, Theorem 8 implies that for minimal semantics, naïve evaluation corresponds to monotonicity, provided that the query does not distinguish instances from their cores.

Moreover monotonicity can be shown to correspond to preservation under a suitable notion of *minimal* homomorphisms, which turn out to be a special case of strong onto homomorphisms. Putting results together we then have:

Corollary 2 *Let Q be a Boolean $\mathsf{Pos}{+}\forall\mathsf{G}$ query such that $Q(D) = Q(\textsc{Core}(D))$ for all D. Then naïve evaluation works for Q under the $[\![\,]\!]^{\min}_{\text{CWA}}$ semantics.*

We do not know how to check for this condition in relevant FO fragments; however note that if we restrict to cores the condition is not necessary. Then naïve evaluation works for Boolean $\mathsf{Pos} + \forall\mathsf{G}$ queries over cores under $[\![\,]\!]^{\min}_{\text{CWA}}$.

3.3.2 Dealing with multiple valuations

Semantics based on multiple valuations were used for instance in the data exchange scenario [Her11], based on earlier work in the area of logic programming [Min82]. In data exchange, they appeared in the context of searching for the right balance between open and closed world assumptions, so as to avoid anomalies that the OWA may lead to, without restricting the setting too much.

For example, the GCWA*-semantics from [Her11] combines several minimal valuations as follows:

$$(\!|I|\!)^{\min}_{\text{CWA}} \;=\; \{v_1(I) \cup \ldots \cup v_n(I) \;\mid\; v_1, \ldots, v_n \text{ are minimal valuations}, n \geq 1\}.$$

That is, not one, but multiple valuations can be applied to a database, and then the results are combined, in this case by the union operation.

If we do not restrict valuations to be minimal, we obtain the following semantics:

$$(\!|I|\!)_{\text{CWA}} \;=\; \{v_1(I) \cup \ldots \cup v_n(I) \;\mid\; v_1, \ldots, v_n \text{ are valuations}, n \geq 1\}.$$

Our framework can be easily transferred to these semantics, which we refer to as *powerset semantics* (and even to generalizations of those where the union is replaced by a more general operator).

In particular we identified a fragment of $\mathsf{Pos} + \forall\mathsf{G}$ guaranteeing the suitable preservation properties. The class is denoted by $\exists\mathsf{Pos} + \forall\mathsf{G}^{\text{bool}}$ and is defined as the class of existential positive queries extended with *Boolean universal guards*, i.e., universally guarded formulae of $\mathsf{Pos} + \forall\mathsf{G}$ which are sentences.

Using this class we obtain the analog of Theorem 7 and Corollary 2 for powerset semantics as well :

Corollary 3 *Let Q be a Boolean query in $\exists\mathsf{Pos} + \forall\mathsf{G}^{\text{bool}}$.*

- *Naïve evaluation works for Q under the $(\!|\,|\!)_{\text{CWA}}$ semantics.*
- *If $Q(D) = Q(\textsc{Core}(D))$ for all D, then naïve evaluation works for Q under the $(\!|\,|\!)^{\min}_{\text{CWA}}$ semantics.*
- *Naïve evaluation works for Q over cores under the $(\!|\,|\!)^{\min}_{\text{CWA}}$ semantics.*

Chapter 4

Query rewriting over graph views

In the previous chapters we mostly concentrated on the problems of compactly representing partially specified data, and querying such representations, motivated in particular by data exchange applications. In this chapter we move to a virtual data integration perspective where partial information about the database is usually available in the form of views. In this context queries posed to the virtual global schema need to be rewritten over the data sources, which can be specified as views over the global database.

We concentrate on the problem of query answering using views which is central in this context. It also finds other interesting applications beyond data integration (such as cache and bandwidth optimization in query processing, data-centric security and privacy), and has indeed received considerable attention (see [LMSS95, AD98, CDGLV00a, NSV10, Afr11] among others).

As in the series of works initiated by [CDGLV00a], we concentrate on a graph-based data model, originally at the basis of semi-structured data [AG08]. Its flexibility makes it particularly suitable for data integration applications, where data comes from heterogenous sources.

Graph databases are relational databases where all relation symbols are binary. In other words a graph database can be viewed as an edge-labeled directed graph. This model is witnessing a renewed interest nowadays since many large scale applications, such as *social networks* [RS09], the *semantic Web* and *RDF* [GHM04], regularly deal with graph-structured data.

Graph data differs conceptually from relational databases in that the topology of the underlying graph is as important as the data it contains. Queries will usually extract data based on connectivity properties of graph nodes [Bae13].

In the literature typical graph queries have at least the expressive power of *Regular Path Queries* (RPQ), defined in [CMW87] (see also the survey [Bae13]). An RPQ selects pairs of nodes connected by a path whose sequence of edge labels satisfies a given regular expression.

A view for us, denoted by V, is then specified using a finite set of RPQs. When evaluated over a graph database D, the view V yields a new graph database $V(D)$ where each $V_i \in V$ is a new edge relation symbol.

Recall from Chapter 1 that determinacy of a query Q by a set of views

V states that there exists a rewriting of Q using V, i.e. a query R such that $R(V(D)) = Q(D)$ for all databases D. On the other hand the rewriting problem asks for the existence of a rewriting in a particular language. It is then natural to ask which rewriting language $\mathcal{L}$ is sufficiently powerful so that determinacy is equivalent to the existence of a rewriting definable in $\mathcal{L}$. This clearly depends on the language used for defining the query and the view. For instance in the case of relational views and queries defined by conjunctive queries (CQs) it is still an open problem to know whether first-order logic is a sufficiently powerful rewriting language. Indeed it is not even known whether there always exists a rewriting that can be evaluated in time polynomial in the size of the view instance [NSV10]. A similar situation arises over graph-databases and RPQ views and queries [CDGLV02].

We investigate the relationship between determinacy and rewriting for RPQ queries and views, under the exact view assumption. Our main contribution in [2], which we briefly describe next, establishes this relationship under an additional monotonicity restriction on the notion of determinacy.

The decidability of the determinacy and rewriting problems is in general strictly related to the question we are interested in. Indeed when proving decidability of determinacy one may produce an explicit rewriting to show that determinacy holds; in these cases one shows at the same time that determinacy is equivalent to the existence of a rewriting in a particular language.

A corollary of our results in [2] also shows that the existence of a Datalog rewriting for RPQ views and queries is decidable. We remark that the existence of an RPQ rewriting for RPQ views and queries was proved decidable in [CGLV02], while the decidability status of determinacy is open [CDGLV02] (it is also open for conjunctive views and queries over relational databases [NSV10]). Determinacy has been shown to be decidable in a scenario where views and queries can only test whether there is a path of distance k between the two nodes, for some given k [Afr11]. This scenario lies at the intersection of CQ and RPQ.

4.1 Monotone determinacy and rewritings

If views and queries are defined in a language $\mathcal{L}$, the first natural question is to ask whether $\mathcal{L}$ itself is enough to express all the rewritings of queries of $\mathcal{L}$ using views of $\mathcal{L}$. If $\mathcal{L}$ is the language of RPQs this is not the case. One can immediately infer it from the following example from [Afr11].

The view Path_3 and Path_4, giving respectively the pairs of nodes connected by a path of length 3 and 4, determines the query Path_5 asking for the pairs of nodes connected by a path of length 5. In fact it can be easily checked that the following FO query over the view schema is a rewriting [Afr11]:

$$R(x, y) = \exists u \ (\text{Path}_4(x, u) \wedge \forall v \ (\text{Path}_3(v, u) \rightarrow \text{Path}_4(v, y)))$$

However one can show that there can exist no RPQ rewriting. In fact the mapping from view instances to query results defined by R is non-monotone. As a consequence this mapping cannot be expressed in any monotone language, such as RPQs.

Monotone query languages such as CQ, Datalog, RPQ and their extensions are of crucial importance in many database applications. It is then natural to

ask whether the non-monotonicity of the rewriting mapping is the only obstacle for the existence of a rewriting in such languages.

This is why we consider a stronger notion of determinacy, referred to as *monotone determinacy*. It further requires that the mapping from view instances to query results is *monotone*.

Definition 4 (Monotone determinacy) *We say that a view V determines a query Q in a monotone way if*

$$\forall D, D', \quad V(D) \subseteq V(D') \;\Rightarrow\; Q(D) \subseteq Q(D')$$

This turns out to coincide with the notion of *losslessness under the sound view assumption* defined in [CDGLV02], that was shown to be decidable, actually ExpSpace-complete, for RPQs.

In the case of CQ views and queries, monotone determinacy can be shown to be equivalent to the existence of a CQ query rewriting [NSV10]. As the latter is decidable [LMSS95], monotone determinacy for CQs is decidable.

In the case of RPQ views and queries the situation is quite different. It is decidable whether a rewriting definable in RPQ exists [CGLV02], and we know that there exist cases of monotone rewritings that are not expressible in RPQ [CDGLV02] (in [2] we also give a concrete example). We thus need a more powerful language in order to express all monotone rewritings.

In [2] we show that, besides RPQs, even their conjunctions (known as CRPQs) are not expressive enough as a rewriting language under monotone determinacy for RPQ queries and views. However our main result states that a rewriting in Datalog, and therefore with PTIME data complexity, can always be found:

Theorem 9 *If V and Q are RPQs and V determines Q in a monotone way then there exists a Datalog rewriting of Q using V.*

This implies that the monotone determinacy problem for RPQs coincides with the problem of the existence of a Datalog rewriting. The latter is therefore decidable by results of [CDGLV02]:

Corollary 4 *Let V and Q be RPQs. It is decidable,* ExpSpace*-complete, whether there exists a Datalog rewriting of Q using V.*

Our proof being constructive, the Datalog rewriting can be computed from V and Q.

4.2 Datalog rewritings and CSP

The starting point for proving Theorem 9 is the relationship between rewriting and certain answers under monotone determinacy. One can easily show that if the view determines the query in a monotone way then the certain answer query, mapping view instances S to $\mathsf{certain}_V(Q, S) = \bigcap\{Q(D) \mid V(D) \supseteq S\}$, is a rewriting.

However certain answers for RPQ views and queries have coNP-hard data complexity [CDGLV00a]. Here we show that there exists another rewriting that is expressible in Datalog. This rewriting can be thought of an "approximation"

of certain answers, which of course coincides with certain answers on view images (i.e. on instances S of the form $V(D)$ for some D), but may in general differ from them on arbitrary S.

This approximation is obtained by exploiting the relationship between certain answers under the sound view assumption and *Constraint Satisfaction Problems* (CSP) [FV98]. Each CSP is defined by a relational structure T, called *the template*; its solutions, denoted by $\mathrm{CSP}(T)$, are all the structures mapping homomorphically into the template. Its complement is usually denoted by $\neg\mathrm{CSP}(T)$.

[CDGLV00b] showed that, for RPQs V and Q, certain answers under the sound view assumption can be expressed as (the complement of) a CSP whose template $T_{Q,V}$ depends only on Q and V. It is known from [FV98] that for every integer l and k with $l \leq k$, and every template T, there exists a $\mathrm{Datalog}_{l,k}$ query approximating $\neg\mathrm{CSP}(T)$ (where for Boolean queries, $\mathrm{Datalog}_{l,k}$ is the fragment of Datalog allowing at most k variables in each rule and l variables in each head).

The $\mathrm{Datalog}_{l,k}$ approximation of $\neg\mathrm{CSP}(T_{Q,V})$ (i.e. of the certain answers) has been considered in [CDGLV00b], where building on results of [FV98] it is was shown to be in a sense maximally contained in certain answers to Q given V, for each l and k.

The crucial point we make in our proof is that even if its $\mathrm{Datalog}_{l,k}$ "approximation" does not compute precisely $\neg\mathrm{CSP}(T_{Q,V})$, if it is exact *on view images*, then it is a rewriting, because it coincides with certain answers.

To prove Theorem 9 we show that if the view determines the query in a monotone way then there is an l and a k, depending only on $V$and Q, such that the $\mathrm{Datalog}_{l,k}$ approximation is exact *on view images*. This proves the existence of a Datalog rewriting.

Chapter 5

Conclusions and future research directions

The work presented in this thesis provides several contributions to the problems of representing and efficiently querying incomplete information, motivated by data interoperability applications.

We concentrated on incompleteness arising from schema mappings relating similar concepts of different database schemas. This made us focus on two forms of incompleteness: incomplete instances incorporating missing/unknown information and views. The former is mostly used for materializing restructured data, while the latter is most common in virtual data integration.

We have seen that the particular adopted semantics of schema mappings influences the needed model and semantics of incompleteness. This has brought us to adopt or develop (when not available) models of incompleteness suited to the data interoperability scenario (such as annotated relational instances or incomplete XML tree descriptions), and to analyze the cost of querying incomplete information under different semantics.

Both in the relational setting and for XML trees this analysis has provided several ad-hoc query answering solutions.

In order to understand the principles inherent in these techniques, we have investigated, in a very abstract setting, the relationship between tractable query answering and the semantics of incompleteness. We have shown that efficient solutions based on naïve evaluation can work beyond previously known cases, under reasonable semantic assumptions.

We have then investigated the tractability of query answering over views, where this is often related to the possibility of finding efficiently computable query rewritings. Our analysis has related the existence of Datalog rewritings to a well known (decidable) determinacy property of RPQ views and queries. The semantic assumption has a big impact in this case as well, being many relevant problems still open under the exact view assumption.

The work presented in this thesis raises many other questions and opens new research directions. In what follows we discuss the main directions we intend to investigate in our forthcoming research. Although we can identify several different objectives, they share a main common thread: moving beyond the relational (and XML) models of incompleteness.

We have already started embarking on this with Nadime Francis's PhD, where we began looking at the *graph database model* (see Chapter 4), which is raising nowadays much interest in research [AG08, CME11, Bae13].

5.1 Incomplete graph data

A well established theory of incomplete information is missing for data models such as graph databases and RDF, but some initial work in understanding incompleteness has been done [CGL98, KNS02, NK13, BLR14]. Models of incompleteness are particularly needed in graph data exchange and integration applications which are witnessing a renewed interest [BPR13, CDLV12, CDLV13].

Over graph data, incompleteness is much more complex than its relational counterpart; essentially because of possibly missing information about the graph structure. This aspect is in common with models of incomplete XML (such as [ASV06] and [6]). However structural incompleteness in XML is constraint by the hierarchical structure of data.

In [KNS02] incompleteness is not modeled at the level of the graph data, but it is incorporated in the query answering semantics: queries are issued on the graph database by allowing a possible partial match between query variables and data nodes, i.e. possibly missing nodes are allowed to complete the query matching.

In [CGL98] incompleteness is introduced at the level of graph edges: edge labels are possibly replaced by unary formulae of a given vocabulary, including a given domain of constants (moreover constraints may be present in the form of formulae associated to nodes). The knowledge about domain elements is represented by a theory in the given vocabulary which is assumed incomplete. A graph, whose edges are labelled with domain elements, and a model of the theory conform to the incomplete representation if there is a suitable "simulation" between the graph and the incomplete representation. This is a sort of mapping, where data elements carried by the graph edges must satisfy, in the given model, the unary formulae they are mapped into.

In [BLR14] a model of incomplete graphs is presented, based on three basic incompleteness features: *node variables* to represent missing nodes, *label variables* to represent missing information about the relationship between nodes, and *regular expressions* which represent missing paths conforming to the given expression. Simple forms of the latter model have also been introduced in graph data exchange [BPR13].

Several aspects still remain to be investigated, especially in relationship with the use of incomplete information to model data heterogeneity in data exchange and integration applications. We next discuss some of them.

Graph constraints As discussed in Chapter 1, suitable models of incomplete information for data exchange applications need to take constraints into account. (However so far solutions in graph data exchange are built using a straightforward adaptation of the relational *chase* procedure, and do not consider constraints [BPR13].)

Constraints considered so far on graph databases essentially enforce the existence of reachability patterns and/or properties of graph nodes [AV99, GT03, BFW00, CGL98, ACD$^+$07].

Few examples of models integrating constraints and incompleteness exist for graph data (a form of node constraints on top of graph schemas has been considered in [CGL98], and a limited form of conditions were modeled over incomplete XML in [ASV06]), while several attempts have been done in the relational case [AM84, LL98].

In a data exchange scenario, in analogy with the relational case [APR13, GO12], we can expect that enforcing constraints would need a model of incompleteness based on a mechanism constraining the presence of data items to conditions on data and structure (along the lines of conditional tables or *world-set decompositions* [OKA08]).

It is expected that in the most general form, computational problems on incomplete data (such that query answering and consistency analysis) will be undecidable in the presence of constraints (cf. [AFL08, CLR03]) but one should expect to find reasonable restrictions for decidability and tractability.

Incompleteness in the source It has been recently advocated that data interoperability tasks cannot ignore the presence of incompleteness in the source data [ALP09, FKPT11, GO12, APR13]. In classical approaches to data exchange and integration, including the ones dealing with graph data [BPR13, CDLV13], source data is considered complete, and incompleteness is only generated by data transformation and integration. However in many scenarios, restructured data has, in turn, to undergo other restructuring and integration steps. This is typical in most *metadata management* tasks where several basic data transformation steps are composed. Here transformed (incomplete) data undergoes subsequent transformations (see [BM07, APRR09]).

When source data is already incomplete, incomplete information is also "translated" into the target database. However mapping rules can transform the original incomplete features into new ones that could be in principle of a different nature than the original ones. This raises the need to develop a model of incomplete information which is *closed* under typical schema mappings. I.e. if incompleteness in the source can be represented in a given class of incomplete instances, it is desirable that the restructured target data can be represented in the same class. It has been shown in [APR13, GO12] that conditional tables enjoy these closure property in the context of relational data exchange, even in the presence of a certain form of target constraints. This confirms the intuition that the presence of conditional data is essential to be able to capture the incompleteness introduced by the the schema mappings. Such closure properties are unexplored for incomplete graph data.

Supporting multiple semantics of incompleteness Existing models of incomplete graph data mostly consider the OWA, and do not seem to be easily adaptable to the CWA. Indeed, in the same way as with XML incomplete information, the main difficulty comes from structural incompleteness, i.e. missing information about the precise graph topology.

Semantics between CWA and OWA, which are interesting in the data exchange scenario (see Chapter 2), have not been considered either.

OWA semantics for incomplete graph data is often homomorphism (or simulation) based [BLR14, CGL98]: an incomplete graph represents, roughly speaking, all complete graph databases where it "embeds" into. Any notion of CWA,

or semantics between CWA and OWA, needs to avoid arbitrary embeddings. This corresponds to avoid an arbitrary amount of missing information.

The closed world assumption could be in principle captured by a suitable notion of *onto homomorphism*. However the question is how this requirement should be interpreted in the presence of graph structural incompleteness. Along the lines of the notion of CWA for incomplete trees introduced in [GLT12], we can interpret the requirement that the homomorphism be "onto" at two levels: either at the level of nodes or at the level of paths, yielding a stronger or weaker notion of CWA.

Both notions are reasonable and deserve investigation. The additional difficulty w.r.t the case of incomplete trees is the more general form of structural incompleteness. Structurally incomplete portions of the graph can be for instance represented by arbitrary regular expression edges as in [BLR14], and are not constraint by the hierarchical structure of trees.

For defining semantics which are between the open world and the closed world assumption, we could in principle adapt two approaches considered for relational data.

One possibility is to exploit an explicit annotation on portions of incomplete data, along the lines of [GZ88] and [5] (see Chapter 2). These annotations specify whether several completions are allowed or not for that portion. In this case a suitable notion of annotation is needed that can be associated to both the data and the structure of a graph.

Another approach would be to adapt the valuation/extension step presented in Chapter 3 for relational data. In the first *valuation* step placeholders for missing information are instantiated with actual data (in one or several possible ways) and in the *extension* step the resulting instance is possibly augmented. The second step defines how much the semantics is "open", by allowing different degrees of extension.

Of course suitable notions of "valuation" and "extension" need to be found for graph data.

5.2 Tractable query evaluation beyond the relational model of incompleteness

Our general analysis of tractable solutions for querying incomplete data in Chapter 3 has been developed either in a very general setting, which can subsume practically every data model, or in connection with the simplest model of incompleteness in relational databases. Moreover we only concentrated on FO queries. However the generality of our framework suggests that it can be instantiated to other database domains as well, and possibly other query languages.

In particular in Chapter 2 we have developed some naïve evaluation techniques for the XML data model. We showed that naïve evaluation works for unions of conjunctive tree queries over rigid incomplete trees [6]. This result was shown to hold also in the case that incomplete trees are "rigid" only w.r.t the axes used in queries [DLM10].

These results can also be viewed as instances of the general setting presented in Chapter 3. It suffices to consider a database domain $\mathbb{D} = \langle \mathcal{D}, \mathcal{C}, [\![\,]\!], \approx \rangle$, where $\mathcal{D}$ is the set of (possibly) incomplete trees, $\mathcal{C}$ is the set of complete trees, $[\![\,]\!]$ is

the usual homomorphism-based semantics of incomplete tree descriptions, and $\approx$ is isomorphism of (relational representations of) trees.

Using the machinery developed in Chapter 3 we can explain why rigid incomplete trees are well behaved w.r.t to naïve evaluation: they enjoy the saturation property. In fact since rigidity completely specifies the tree structure (in the axes concerned by the query) every rigid incomplete tree has an isomorphic complete tree in its semantics. Union of conjunctive tree queries are easily seen to be preserved under homomorphisms of tree descriptions, and therefore under the semantic mapping. Naïve evaluation is then guaranteed for such queries.

The connection between naïve evaluation and certain answers over incomplete tree descriptions has not been considered beyond positive queries. This would make sense especially for semantics other than OWA, which usually allow naïve evaluation over larger classes of queries. Our approach could then give new insights in this setting.

Our approach can in principle be instantiated to other data models, such as graph databases and RDF [BLR14, NK13], where much less is known about tractability restrictions.

However, to make it suitable to XML or graph incomplete data, our approach needs to be extended to take into account a form of structural incompleteness, and query languages possibly beyond FO. For this we need to look at different questions, as we discuss next.

Relaxing the notion of naïve evaluation In the presence of structural incompleteness the usual notion of naïve evaluation is likely not to be applicable in most of the cases. The saturation property immediately fails: in most cases no complete instance can be isomorphic to a structurally incomplete one. As a consequence very strong restrictions are usually needed. Rigidity of incomplete trees is an example of such restrictions. A similar situation arises over graph patterns considered in [BLR14]: naïve evaluation is used only for very simple patterns without structural incompleteness, in cases when it is possible to reduce query answering to the relational case.

On the other hand there are several examples of tractable query answering solutions over incomplete data which are not based on naïve evaluation. To mention a few, in [GLT12] it is shown that a form of Boolean combinations of conjunctive queries can be evaluated in polynomial time over rigid incomplete trees, although naïve evaluation is known not to work in general. Similarly the polynomial time algorithm we provide for word patterns over a class of incomplete DOM trees, as well as tractable cases of query answering over structurally incomplete graph patterns in [BLR14], are not based on naïve evaluation.

It would be interesting to find a sufficiently general notion of tractable solution that could subsume these ad-hoc approaches. While this is an ambitious objective, one can start with generalizing naïve evaluation-based solutions.

In fact there are several possible extensions of naïve evaluation that can be considered, which may still retain its nice properties, and in particular computational efficiency.

The usual notion of naïve evaluation requires that certain answers over an incomplete instance I can be computed by evaluating the query directly over I. However the same desirable properties of naïve evaluation would hold if certain answers over I could be computed by evaluating the query over *another instance*

I', easily computable from I.

We know at least one instance of such an approach. As discussed in Chapter 3, in [1] we have shown that when the saturation property does not hold on the whole database domain, certain answers to queries monotone w.r.t. the semantics are indeed computed by naïve evaluation on another instance: in particular any semantically equivalent instance which belongs to a saturated subdomain. The problem is that in most cases this new instance is hard to compute from the available one. For example in the relational case, under semantics based on minimal valuations, such new instance is the core of the original instance, which is in general hard to compute [CM77, HN92]. However in some cases, such as large classes of data exchange settings, the core of the canonical solution can be computed in polynomial time [GN08]. This makes naïve evaluation over the core a feasible solution to query answering in data exchange for $\mathsf{Pos} + \forall\mathsf{G}$ queries, under minimal semantics. One could in principle also combine this approach with query rewriting, by requiring that certain answers to a query Q can be computed by naïvely evaluating *another query* Q'. We can push the extension even further and require that certain answers can be computed by efficiently combining the query results over a finite, efficiently computable, set of other instances.

These ideas follow a very popular approach in the field of ontology-based data access, where query rewriting is the main mechanism to query ontology-based intensional data [GOP11]. In particular solutions along the lines of the combined approach [KLT$^+$10] and the chase of Datalog-based constraints [CGL12] are based on a modification of the original database, sometimes combined with query rewriting.

We expect that all this approaches would have to be necessarily combined with structural restrictions on incomplete instances.

Tractability in specific applications We saw that over complex models of incompleteness finding tractable query answering solutions becomes more difficult. However incomplete data generated in specific applications, such as data interoperability tasks, is often of a very particular form. This is already true in the relational setting. For instance the canonical solution in data exchange can be partitioned into blocks having an independent and bounded set of nulls. This kind of property has been effectively used to devise polynomial time algorithms for tasks which are computationally hard over arbitrary incomplete relational instances [FKP05, GN08, Her11]. This suggests that query answering in data interoperability applications can be in principle computationally easier than the general problem of query answering over incomplete data.

It would be interesting to analyze the peculiarities of incomplete trees or incomplete graphs arising in these applications, and see whether they can be used to obtain that tractable solutions – such as naïve evaluation or its generalizations – work over larger classes of queries. To the best of our knowledge, there has been no study of naïve evaluation over restricted classes of instances.

Preservation theorems Moving beyond the relational model and query languages raises two main points about the use of preservation theorems in connection with naïve evaluation techniques. First of all we may deal with query languages beyond FO, and second, we usually deal with restricted classes of

structures.

Query languages beyond FO. Preservation theorems have not received much attention beyond FO, even over arbitrary relational structures, with a few exceptions. For instance preservation theorems in the finite for existential fragments of several non-first-order logics have been proved in [FV03]; some infinitary logics have also been considered [Kei71].

Of course Datalog is preserved under homomorphisms, and in fact our approach for naïve evaluation under OWA presented in Chapter 3 applies to Datalog queries as well. However although Datalog is the existential positive fragment of *Least Fixed-Point Logic* (LFP), it does not coincide with the LFP fragment preserved under homomorphism, over both arbitrary and finite structures [DK08], and no analog of Rossman's theorem is known for LFP.

So for LFP queries naïve evaluation could work outside Datalog under the OWA, but natural fragments guaranteeing this property beyond Datalog are not known. Precise characterization of preservation properties of queries are also missing for second-order logics; and restricted forms of homomorphisms have not been considered at all for these logics, to the best of our knowledge.

While proving preservation theorems is usually a very difficult task, especially in the finite, finding classes of queries preserved under various notions of homomorphism is already interesting to our purposes. One possible approach is to start with syntactic fragments of FO which we introduced in [1] (see Chapter 3), and see which higher order features can be introduced without altering the preservation property.

Restricted classes of structures. Although some forms of graph and tree patterns have a relational representation, they are not arbitrary instances of their schema. Preservation theorems for such instances need to be relativized to restricted classes of structures.

This problem has been investigated for several natural classes of finite relational structures, and the classical homomorphism preservation theorem has been shown to relativize to classes of bounded degree, of bounded treewidth, classes excluding a minor [ADK06], and more generally to the so called *quasi-wide* classes [Daw10], encompassing those.

However structures we are interested in, such as XML with data, fall in none of these classes, as data values generate relational structures of arbitrary treewidth, and can be shown even outside the larger classes.

By possibly combining restricted classes of structures and higher-order languages, this naturally identifies preservation properties which have not been investigated yet.

5.3 Query rewriting over views

When considering partial information provided by views, many problems have not been fully understood or not addressed at all, especially for specific data models, such as graph databases. We discuss some of them next.

Languages for monotone rewritings We have not yet fully understood the properties of rewritings of RPQ queries w.r.t. RPQ views, even under the monotonicity assumption. In particular it is not yet clear which language is really needed to express such rewritings.

In Chapter 4 we have presented our result showing that Datalog can express all monotone rewritings of RPQ queries w.r.t. RPQ views. However there is no evidence that the full expressive power of Datalog is needed, and an interesting line of research could be to look for tighter upper bounds. There are two main reasons for this.

The first reason is the objective to identify a well behaved query and view language with closure properties w.r.t. rewritings. I.e. one would like to have a monotone view and query language guaranteeing that monotone rewritings can be expressed in the language itself. Full Datalog may be too powerful to enjoy this closure properties, while these are more likely for Datalog fragments. All examples we are aware of use only the transitive closure of binary CRPQs as monotone rewritings.

The second reason is related to the complexity of evaluating the rewriting. Datalog rewritings obtained in our proof use a number of variables exponential in the number of states of query and view automata. This has of course a big impact on the complexity of evaluating the rewriting, that although polynomial in data complexity, depends exponentially on the number of variables in rules of the program. Especially when dealing with graph data, given their possible huge size, arbitrary polynomial time solutions may not be satisfactory, and fixed parameter tractability is a more desirable property.

Recent results on CSP have fully characterized CSP problems (whose complement is) expressible in Datalog [BK09], and in doing so they have shown that $\text{Datalog}_{2,3}$ is enough in the case of binary relational structures as ours.

Of course this result does not apply immediately to the CSP associated with certain answers in our case, since it is in general not expressible in Datalog on all instances, but only on view images. However it would be interesting to see whether techniques of [BK09] can be adapted directly to our case to obtain Datalog rewritings using a constant number of variables.

Determinacy and rewriting beyond monotonicity Without the monotonicity assumption the relationship between determinacy and rewriting is largely unexplored (under the exact view assumption).

Towards solving this problem, in [2] we proved that if an RPQ view determines an RPQ query then one can easily find both a rewriting with NP data complexity and one with coNP data complexity. This easily follows from the property that one can always find a polynomial size counter-image of a view instance.

If this counter-image could be constructed in polynomial time this would immediately give a rewriting with PTIME data complexity, in the case that the view determines the query: in fact to evaluate the query over a view instance it would suffice to construct a counter-image and evaluate the query over it. However in [2] we showed that a counter-image cannot be constructed in polynomial time (unless P=NP) and it is open whether a rewriting with PTIME data complexity exists.

Deciding determinacy without the monotonicity assumption is a related question. As discussed in Chapter 4 it is open in many cases (under the exact view assumption), in particular for RPQ views and queries, and is of course an interesting question to look at.

Deciding determinacy may come with a constructive procedure that natu-

rally identifies rewriting languages, as in [Afr11]. We also remark that [Afr11] considers a restricted form of queries and views over relational data: conjunctive path queries. These can be viewed as a form of path queries over graphs without recursion. Therefore the extension of techniques developed in [Afr11] can be a starting point of our investigation. This is ongoing work in the context of Nadime Francis's PhD.

Bibliography

[ABC99] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 68–79, 1999.

[ABFL04] Marcelo Arenas, Pablo Barceló, Ronald Fagin, and Leonid Libkin. Locally Consistent Transformations and Query Answering in Data Exchange. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 229–240, 2004.

[ABLM14] Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.

[ACD+07] Yves André, Anne-Cécile Caron, Denis Debarbieux, Yves Roos, and Sophie Tison. Path constraints in semistructured data. *Theoretical Computer Science*, 385(13):11 – 33, 2007.

[AD98] Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 254–263, 1998.

[ADK06] Albert Atserias, Anuj Dawar, and Phokion Kolaitis. On preservation under homomorphisms and unions of conjunctive queries. *Journal of the ACM*, 53(2):208–237, 2006.

[ADLM14] Shun'ichi Amano, Claire David, Leonid Libkin, and Filip Murlak. Xml schema mappings: Data exchange and metadata management. *Journal of the ACM*, 61(2):12:1–12:48, April 2014.

[AFL08] Marcelo Arenas, Wenfei Fan, and Leonid Libkin. On the complexity of verifying consistency of XML specifications. *SIAM Journal on Computing*, 38(3):841–880, 2008.

[Afr11] Foto Afrati. Determinacy and query rewriting for conjunctive queries and views. *Theoretical Computer Science*, 412(11):1005–1021, 2011.

[AG87] Miklós Ajtai and Yuri Gurevich. Monotone versus positive. *Journal of the ACM*, 34(4):1004–1015, 1987.

[AG08] Renzo Angles and Claudio Gutiérrez. Survey of graph database models. *ACM Computing Surveys*, 40(1), 2008.

[AKG91] Serge Abiteboul, Paris Kanellakis, and Gösta Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78(1):158–187, 1991.

[AL08] Marcelo Arenas and Leonid Libkin. XML data exchange: Consistency and query answering. *Journal of the ACM*, 55(2), 2008.

[ALM09] Shun'ichi Amano, Leonid Libkin, and Filip Murlak. XML schema mappings. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 33–42, 2009.

[ALP09] Foto Afrati, Chen Li, and Vassia Pavlaki. Data Exchange: Query Answering for Incomplete Data Sources. In *3rd International Conference on Scalable Information Systems*, 2009.

[AM84] Paolo Atzeni and Nicola M. Morfuni. Functional dependencies in relations with null values. *Information Processing Letters*, 18(4):233–238, 1984.

[APR13] Marcelo Arenas, Jorge Pérez, and Juan L. Reutter. Data exchange beyond complete data. *Journal of the ACM*, 60(4):28, 2013.

[APRR09] Marcelo Arenas, Jorge Pérez, Juan L. Reutter, and Cristian Riveros. Composition and inversion of schema mappings. *SIGMOD Record*, 38(3):17–28, 2009.

[ASV06] Serge Abiteboul, Luc Segoufin, and Victor Vianu. Representing and querying XML with incomplete information. *ACM Transactions on Database Systems*, 31(1):208–254, 2006.

[AV99] Serge Abiteboul and Victor Vianu. Regular path queries with constraints. *Journal of Computer and System Sciences*, 58(3):428 – 452, 1999.

[AYCLS02] Sihem Amer-Yahia, SungRan Cho, Laks V. S. Lakshmanan, and Divesh Srivastava. Tree pattern query minimization. *VLDB Journal*, 11(4):315–331, 2002.

[Bae13] Pablo Barceló Baeza. Querying graph databases. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 175–188, 2013.

[Bar09] Pablo Barceló. Logical foundations of relational data exchange. *SIGMOD Record*, 38(1):49–58, 2009.

[BFW00] Peter Buneman, Wenfei Fan, and Scott Weinstein. Path constraints in semistructured databases. *Journal of Computer and System Sciences*, 61(2):146–193, 2000.

[BK09] Libor Barto and Marcin Kozik. Constraint satisfaction problems of bounded width. In *Annual Symposium on Foundations of Computer Science*, pages 595–603, 2009.

[BLPS10] Pablo Barceló, Leonid Libkin, Antonella Poggi, and Cristina Sirangelo. XML with incomplete information. *Journal of the ACM*, 58(1), 2010.

[BLR14] Pablo Barceló, Leonid Libkin, and Juan Reutter. Querying regular graph patterns. *Journal of the ACM*, 61(1), 2014.

[BM07] Philip A. Bernstein and Sergey Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD Conference*, pages 1–12, 2007.

[BMS08] Henrik Björklund, Wim Martens, and Thomas Schwentick. Optimizing Conjunctive Queries over Trees Using Schema Information. In *Intl. Symp. on Mathematical Foundations of Computer Science (MFCS)*, pages 132–143, 2008.

[BMS11] Henrik Björklund, Wim Martens, and Thomas Schwentick. Conjunctive query containment over trees. *Journal of Computer and System Sciences*, 77(3):450–472, 2011.

[BPR13] Pablo Barceló, Jorge Pérez, and Juan L. Reutter. Schema mappings and data exchange for graph databases. In *Intl. Conf. on Database Theory (ICDT)*, pages 189–200, 2013.

[BtCLW13] Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: a study through disjunctive datalog, CSP, and MMSNP. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 213–224, 2013.

[CDGLV00a] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Answering regular path queries using views. In *Intl. Conf. on Data Engineering (ICDE)*, pages 389–398. IEEE, 2000.

[CDGLV00b] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing and constraint satisfaction. In *ACM/IEEE Symp. on Logic in Computer Science (LICS)*, pages 361–371. IEEE, 2000.

[CDGLV02] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Lossless regular views. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 247–258. ACM, 2002.

[CDLV12] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Query processing under glav mappings for relational and graph databases. *PVLDB*, 6(2):61–72, 2012.

[CDLV13] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. On simplification of schema mappings. *Journal of Computer and System Sciences*, 79(6):816–834, 2013.

[CGL98] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Semi-structured data with constraints and incomplete information. In *Description Logics*, 1998.

[CGL02] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Representing and reasoning on XML documents: a description logic approach. *Journal of Logic and Computation*, 9(3):295–318, 2002.

[CGL+07] Diego Calvanese, Giuseppe Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *Journal of Automated Reasoning*, 39(3):385–429, October 2007.

[CGL12] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.

[CGLV02] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of regular expressions and regular path queries. *Journal of Computer and System Sciences*, 64(3):443–465, 2002.

[CGLV07] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing: On the relationship between rewriting, answering and losslessness. *Theoretical Computer Science*, 371(3):169–182, 2007.

[CGP12] Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence*, 193:87–128, 2012.

[CK90] C.C. Chang and H.Jerome Keisler. *Model Theory*. North Holland, 1990.

[CKS09] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Incorporating constraints in probabilistic xml. *ACM Transactions on Database Systems*, 34(3), 2009.

[CLR03] Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 260–271, 2003.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *ACM Symposium on the Theory of Computing (STOC)*, pages 77–90, 1977.

[CME11] Philippe Cudré-Mauroux and Sameh Elnikety. Graph data management systems for new application domains. *PVLDB*, 4(12):1510–1511, 2011.

[CMW87] Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. *SIGMOD Record*, 16(3):323–330, December 1987.

[Com83] Kevin Compton. Some useful preservation theorems. *Journal of Symbolic Logic*, 48(2):427–440, 1983.

[Dav08] Claire David. Complexity of Data Tree Patterns over XML Documents. In *Intl. Symp. on Mathematical Foundations of Computer Science (MFCS)*, pages 278–289, 2008.

[Daw10] Anuj Dawar. Homomorphism preservation on quasi-wide classes. *Journal of Computer and System Sciences*, 76(5):324–332, 2010.

[DD96] C.J. Date and Hugh Darwen. *A Guide to the SQL Standard*. Addison-Wesley, 1996.

[DGLLR07] Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. On reconciling data exchange, data integration, and peer data management. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 133–142, 2007.

[DK08] Anuj Dawar and Stephan Kreutzer. On datalog vs. lfp. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 160–171, 2008.

[DLM10] Claire David, Leonid Libkin, and Filip Murlak. Certain answers for XML queries. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 191–202, 2010.

[DOM04] DOM. Document object model (dom). w3c recommendation. `http://www.w3.org/TR/DOM-Level-3-Core/`, April 2004.

[FHH+09] Ronald Fagin, Laura M. Haas, Mauricio A. Hernandez, Renée J. Miller, Lucian Popa, and Yannis Velegrakis. Clio: Schema mapping creation and data exchange. In *Conceptual Modeling: Foundations and Applications, Essays in Honor of J. Mylopoulos*, pages 198–236. Springer-Verlag, 2009.

[FKMP05] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. *Theoretical Computer Science*, 336:89–124, 2005.

[FKP05] Ronald Fagin, Phokion Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Transactions on Database Systems*, 30(1):174–210, 2005.

[FKPT11] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Reverse data exchange: coping with nulls. *ACM Transactions on Database Systems*, 36(2):11:1–11:42, June 2011.

[FV98] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1998.

[FV03] Tomás Feder and Moshe Y. Vardi. Homomorphism closed vs. existential positive. In *ACM/IEEE Symp. on Logic in Computer Science (LICS)*, pages 311–, Washington, DC, USA, 2003.

[GHM04] Claudio Gutierrez, Carlos Hurtado, and Alberto O. Mendelzon. Foundations of semantic web databases. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 95–106, 2004.

[GKS06] Georg Gottlob, Christoph Koch, and Klaus U. Schulz. Conjunctive queries over trees. *Journal of the ACM*, 53(2):238–272, 2006.

[GLT12] Amélie Gheerbrant, Leonid Libkin, and Tony Tan. On the complexity of query answering over incomplete XML documents. In *Intl. Conf. on Database Theory (ICDT)*, pages 169–181, 2012.

[GN08] Georg Gottlob and Alan Nash. Efficient Core Computation in Data Exchange. *Journal of the ACM*, 55(2), 2008.

[GO12] Gösta Grahne and Adrian Onet. Representation systems for data exchange. In *Intl. Conf. on Database Theory (ICDT)*, pages 208–221, 2012.

[GOP11] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. Ontological query answering via rewriting. In *Advances in Databases and Information Systems*, volume 6909 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin Heidelberg, 2011.

[Gra02] Gösta Grahne. Information integration and incomplete information. *IEEE Data Eng. Bull.*, 25(3):46–52, 2002.

[GT03] Gösta Grahne and Alex Thomo. Query containment and rewriting using views for regular path queries under constraints. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 111–122, 2003.

[GZ88] Georg Gottlob and Roberto Zicari. Closed world databases opened through null values. In *International Conference on Very Large Data Bases (VLDB)*, pages 50–61, 1988.

[Hal00] Alon Halevy. Theory of answering queries using views. *SIGMOD Record*, 29(1):40–47, 2000.

[Her11] André Hernich. Answering non-monotonic queries in relational data exchange. *Logical Methods in Computer Science*, 7(3), 2011.

[HHH+05] Laura M. Haas, Mauricio A. Hernández, Howard Ho, Lucian Popa, and Mary Roth. Clio grows up: from research prototype to industrial tool. In *ACM Intl. Conference on Management of Data (SIGMOD)*, pages 805–810, 2005.

[HLS11] André Hernich, Leonid Libkin, and Nicole Schweikardt. Closed world data exchange. *ACM Transactions on Database Systems*, 36(2):14:1–14:40, 2011.

[HN92] Pavol Hell and Jaroslav Nešetřil. The core of a graph. *Discrete Mathematics*, 109(1-3):127–126, 1992.

[Hul97] Richard Hull. Managing semantic heterogeneity in databases: A theoretical prospective. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 51–61, 1997.

[IL84] Tomasz Imielinski and Witold Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.

[Kei65a] H. Jerome Keisler. Finite approximations of infinitely long formulas. In *Symposium on the Theory of Models*, pages 158–169. North Holland, 1965.

[Kei65b] H. Jerome Keisler. Some applications of infinitely long formulas. *Journal of Symbolic Logic*, 30(3):339–349, 1965.

[Kei71] H. Jerome Keisler. *Model theory for infinitary logic: Logic with countable conjunctions and finite quantifiers*. Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Co., 1971.

[KLT+10] Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev. The combined approach to query answering in DL-Lite. In *Principles of Knowledge Representation and Reasoning (KR)*, 2010.

[KNS02] Yaron Kanza, Werner Nutt, and Yehoshua Sagiv. Querying incomplete information in semistructured data. *Journal of Computer and System Sciences*, 64(3):655–693, 2002.

[Kol05] Phokion G. Kolaitis. Schema Mappings, Data Exchange, and Metadata Management. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 61–75, 2005.

[Len02] Maurizio Lenzerini. Data integration: a theoretical perspective. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 233–246, 2002.

[Lib06] Leonid Libkin. Data exchange and incomplete information. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 60–69, 2006.

[Lib11] Leonid Libkin. Incomplete information and certain answers in general data models. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 59–70, 2011.

[Lib14] Leonid Libkin. Certain answers as objects and knowledge. In *Principles of Knowledge Representation and Reasoning (KR)*, 2014.

[Lip79] Witold Lipski. On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems*, 4(3):262–296, 1979.

[LL98] Mark Levene and George Loizou. Axiomatisation of functional dependencies in incomplete relations. *Theoretical Computer Science*, 206(1-2):283–300, 1998.

[LLR02] Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Source inconsistency and incompleteness in data integration. In *Description Logics*, 2002.

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 95–104, 1995.

[Min82] Jack Minker. On indefinite databases and the closed world assumption. In *CADE*, pages 292–308, 1982.

[MS04] Gerome Miklau and Dan Suciu. Containment and equivalence for a fragment of XPath. *Journal of the ACM*, 51(1):2–45, 2004.

[NK13] Charalampos Nikolaou and Manolis Koubarakis. Incomplete information in rdf. In *Intl. Conf. on Web Reasoning and Rule Systems (RR)*, pages 138–152, 2013.

[NSV10] Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *ACM Transactions on Database Systems*, 35(3), 2010.

[OKA08] Dan Olteanu, Christoph Koch, and Lyublena Antova. World-set decompositions: expressiveness and efficient algorithms. *Theoretical Computer Science*, 403(2-3):265–284, 2008.

[Rei77] Raymond Reiter. On closed world data bases. In *Logic and Data Bases*, pages 55–76, 1977.

[Ros08] Benjamin Rossman. Homomorphism preservation theorems. *Journal of the ACM*, 55(3), 2008.

[RS09] Royi Ronen and Oded Shmueli. Soql: A language for querying and creating data in social networks. In *Intl. Conf. on Data Engineering (ICDE)*, pages 1595–1602, 2009.

[SA07] Pierre Senellart and Serge Abiteboul. On the complexity of managing probabilistic XML data. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 283–292, 2007.

[Sto95] Alexei P. Stolboushkin. Finitely monotone properties. In *ACM/IEEE Symp. on Logic in Computer Science (LICS)*, pages 324–330, 1995.

Publications

[1] Amélie Gheerbrant, Leonid Libkin, and Cristina Sirangelo. Naïve evaluation of queries over incomplete databases. *ACM Transactions on Database Systems*, 2014, *to appear*.

[2] Nadime Francis, Luc Segoufin, and Cristina Sirangelo. Datalog rewritings of regular path queries using views. In *Intl. Conf. on Database Theory (ICDT)*, pages 107–118, 2014.

[3] Amélie Gheerbrant, Leonid Libkin, and Cristina Sirangelo. When is naïve evaluation possible? In *ACM Symp. on Principles of Database Systems (PODS)*, pages 75–86, 2013.

[4] Amélie Gheerbrant, Leonid Libkin, and Cristina Sirangelo. Reasoning about pattern-based xml queries. In *Intl. Conf. on Web Reasoning and Rule Systems (RR)*, pages 4–18, 2013.

[5] Leonid Libkin and Cristina Sirangelo. Data exchange and schema mappings in open and closed worlds. *Journal of Computer and System Sciences*, 77(3):542–571, 2011.

[6] Pablo Barceló, Leonid Libkin, Antonella Poggi, and Cristina Sirangelo. XML with incomplete information. *Journal of the ACM*, 58(1), 2010.

[7] Leonid Libkin and Cristina Sirangelo. Disjoint pattern matching and implication in strings. *Information Processing Letters*, 110(4):143–147, 2010.

[8] Leonid Libkin and Cristina Sirangelo. Reasoning about XML with temporal logics and automata. *Journal of Applied Logic*, 8(2):210–232, June 2010.

[9] Pablo Barceló, Leonid Libkin, Antonella Poggi, and Cristina Sirangelo. XML with incomplete information: models, properties, and query answering. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 237–246, 2009.

[10] Leonid Libkin and Cristina Sirangelo. Open and closed world assumptions in data exchange. In *Description Logics*, 2009.

[11] Cristina Sirangelo. Relational algebra operators. In *Encyclopedia of Database Systems*. Springer, 2009.

[12] Leonid Libkin and Cristina Sirangelo. Data exchange and schema mappings in open and closed worlds. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 139–148, 2008.

[13] Leonid Libkin and Cristina Sirangelo. Reasoning about xml with temporal logics and automata. In *Intl. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR)*, pages 97–112, 2008.

[14] Luc Segoufin and Cristina Sirangelo. Constant-memory validation of streaming xml documents against dtds. In *Intl. Conf. on Database Theory (ICDT)*, pages 299–313, 2007.

[15] Filippo Furfaro, Giuseppe M. Mazzeo, Domenico Saccà, and Cristina Sirangelo. Compressed hierarchical binary histograms for summarizing multi-dimensional data. *Intl. Journal on Knowledge And Information Systems (KAIS)*, 15(3), July 2007.

[16] Filippo Furfaro, Giuseppe M. Mazzeo, and Cristina Sirangelo. Exploiting cluster analysis for constructing multi-dimensional histograms on both static and evolving data. In *International Conference on Extending Database Technology (EDBT)*, March 2006.

[17] Luciano Caroprese, Sergio Greco, Cristina Sirangelo, and Ester Zumpano. Declarative semantics of production rules for integrity maintenance. In *Intl. Conf. on Logic Programming (ICLP)*, pages 26–40, 2006.

[18] Filippo Furfaro, Giuseppe M. Mazzeo, and Cristina Sirangelo. Clustering-based histograms for multi-dimensional data. In *Intl. Conf. on Data Warehousing and Knowledge Discovery (DaWaK)*, pages 478–487, 2005.

[19] Filippo Furfaro, Giuseppe M. Mazzeo, Domenico Saccà, and Cristina Sirangelo. Hierarchical binary histograms for summarizing multi-dimensional data. In *Symposium on Applied Computing (SAC)*, pages 598–603, 2005.

[20] Luciano Caroprese, Sergio Greco, Cristina Sirangelo, and Ester Zumpano. A logic based approach to p2p databases. In *Italian Symposium on Advanced Database Systems (SEBD)*, pages 67–74, 2005.

[21] Sergio Greco, Cristina Sirangelo, Irina Trubitsyna, and Ester Zumpano. Preferred repairs for inconsistent databases. In *Encyclopedia of Database Technologies and Applications*, pages 480–485. Idea Group, 2005.

[22] Sergio Greco, Cristina Sirangelo, Irina Trubitsyna, and Ester Zumpano. Feasibility conditions and preference criteria in querying and repairing inconsistent databases. In *Intl. Conf. on Database and Expert Systems Applications (DEXA)*, pages 44–55, 2004.

[23] Filippo Furfaro, Giuseppe M. Mazzeo, Domenico Saccà, and Cristina Sirangelo. A new histogram-based technique for compressing multi-dimensional data. In *Italian Symposium on Advanced Database Systems (SEBD)*, pages 18–29, 2004.

[24] Alfredo Cuzzocrea, Filippo Furfaro, Elio Masciari, and Cristina Sirangelo. Approximate query answering on sensor network data streams. In *Italian Symposium on Advanced Database Systems (SEBD)*, pages 93–108, 2003.

[25] Sergio Greco, Cristina Sirangelo, Irina Trubitsyna, and Ester Zumpano. Preferred repairs for inconsistent databases. In *International Database Engineering and Applications Symposium (IDEAS)*, pages 202–211, 2003.

[26] Alfredo Cuzzocrea, Filippo Furfaro, Elio Masciari, Domenico Saccà, and Cristina Sirangelo. Approximate query answering on sensor network data streams. In *Intl. Workshop on Geo Sensor Networks*, 2003.

[27] Francesco Buccafurri, Filippo Furfaro, Domenico Saccà, and Cristina Sirangelo. A quad-tree based multiresolution approach for two-dimensional summary data. In *Intl. Conf. on Scientific and Statistical Database Management (SSDBM)*, pages 127–140, 2003.