Università degli Studi di Roma "La Sapienza"
Dottorato di Ricerca in Ingegneria Informatica

Dealing with Inconsistency and Incompleteness in Data Integration

Ph.D. Thesis

Domenico Lembo

Author's contact information:

Domenico Lembo
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
via Salaria, 113
I-00198 Roma, Italy
e-mail: lembo@dis.uniroma1.it

Acknowledgements

I wish to thank my advisor Maurizio Lenzerini, who constantly guided me throughout my Ph.D. course. Without his support I could never have accomplished this work.

I also want to thank Riccardo Rosati, who contributed to the achievement of several technical results provided in this thesis, and Andrea Calì which, together with Riccardo, has been my co-author on the papers presenting some of the contributions of this thesis. Part of the research has been carried out together with Thomas Eiter and Gianluigi Greco, during my staying at the Vienna University of Technology. I am grateful to them for the research work done together. A special thank goes to Diego Calvanese and Giuseppe De Giacomo for several fruitful and stimulating discussions we had together.

I would like to thank all the people of the Dipartimento di Informatica e Sistemistica of the University of Rome "La Sapienza", the members of the Ph.D. Committee at the Dipartimento di Informatica e Sistemistica, Thomas Either and Sergio Greco as external referees, Silvio Salza and Luigi Palopoli as internal referees, and all the people of which I enjoyed the company during my Ph.D. In particular, I want to express my gratitude my girlfriend Natalia, my parents Angelo and Giuliana, and my sister Teresa.

Contents

Chapter 1

Introduction

Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data [92, 96, 150, 108]. Such unified view is called *global schema*: it represents the intensional level of the integrated and reconciled data, and provides the elements for expressing user queries. In formulating the queries, the user is freed from the knowledge on where data are, how data are structured at the sources, and how data are to be merged and reconciled to fit into the global schema.

The interest in data integration systems has been continuously growing in the last years. The recent developments of Computer and Telecommunication technology, such as the expansion of the Internet and the World Wide Web, have made available to users a huge number of information sources, generally autonomous, heterogeneous and widely distributed. As a consequence, information integration has emerged as a crucial issue in many application domains, e.g., distributed databases, cooperative information systems, data warehousing, data mining, data exchange, as well as in accessing distributed data over the web. Nevertheless, state of the art techniques do not properly support users in effectively gathering data relevant to achieve their aims.

Designing a data integration system is a very complex task, and is characterized by a number of issues, including the following:

1. modelling a data integration system, i.e., how to define both the global schema, and the relationships between the global schema and the sources;
2. materialized vs. virtual integration;
3. querying a data integration system, i.e., how to process queries expressed on the global schema;
4. dealing with incomplete data sources;
5. dealing with inconsistent data sources;
6. dealing with limitations on accessing the sources;
7. data cleaning and reconciliation.

In the following we describe in details each of the above points.

1.1 Modelling Data Integration Systems

Sources that are to be integrated in a data integration system are typically heterogeneous, meaning that they adopt different models and systems for storing data. This poses challenging problems in both representing the sources in a common format within the integration system, and specifying the global schema. As for the former issue, data integration systems make use of suitable software components, called *wrappers*, that present data at the sources in the form adopted within the system, hiding the original structure of the sources and the way in which they are modelled. The representation of the sources in the system, is generally given in terms of a *source schema*.

With regard to the specification of the global schema, the goal is to design it so as to provide an appropriate abstraction of all the data residing at the sources. An aspect deserving special attention is the choice of the language used to express the global schema. Since such a view should mediate among different representations of overlapping worlds, the language should provide flexible and powerful representation mechanisms, i.e., it should allow for the specification of several forms of integrity constraints.

However, the design of the global schema is not the only issue in modelling a data integration system. Indeed, another crucial aspect is the definition of the *mapping*, i.e., the specification of the relationship holding between the sources and the global schema. To this purpose, two basic approaches have been proposed in the literature: the *local-as-view* (or simply LAV) approach, and the *global-as-view* (or simply GAV) approach.

The LAV approach requires the global schema to be specified independently of the sources. In turn, the sources are defined as views, i.e., queries, over the global schema [103, 1, 34]. In particular, to each element of the source schema, a view over the global schema is associated. Thus, in the local-as-view approach, we specify the meaning of the sources in terms of the elements of the global schema. The following example, adapted from [111], shows a LAV system.

Example 1.1.1 Consider a relational global schema with the following relations:

$$\begin{aligned} &movie(Title, Year, Director)\\ &european(Director)\\ &review(Title, Critique) \end{aligned}$$

The source schema consists of three source relations s_1, s_2, and s_3: s_1 stores title, year and director of movies produced since 1960, while s_2 stores title and critique of movies with European director, and s_3 stores European directors.

$$\begin{aligned} &s_1(Title, Year, Director)\\ &s_2(Title, Critique)\\ &s_3(Director) \end{aligned}$$

The LAV mapping is given by the following views (in this example we use conjunctive queries

with arithmetic comparisons [2]):

$$\begin{aligned} s_1(X,Y,Z) &\leftarrow movie(X,Y,Z), (Y \geq 1960) \\ s_2(X,W) &\leftarrow movie(X,Y,Z), review(X,W), european(Z) \\ s_3(X) &\leftarrow european(X) \end{aligned}$$

■

Examples of LAV systems are Infomaster [59, 58], Information Manifold [103], and the system presented in [125].

The GAV approach requires that the global schema is expressed in terms of the data sources. More precisely, to every element of the global schema, a view over the data sources is associated, so that its meaning is specified in terms of the data residing at the sources, as shown in the following example.

Example 1.1.2 Consider a data integration system with source and global schemas as in Example 1.1.1. The GAV mapping associates to each relation symbol of the global schema a view over the sources as follows:

$$\begin{aligned} movie(X,Y,Z) &\leftarrow s_1(X,Y,Z) \\ european(X) &\leftarrow s_3(X) \\ review(X,Y) &\leftarrow s_2(X,Y) \end{aligned}$$

■

Examples of data integration systems based on GAV are TSIMMIS [80], Garlic [42], Squirrel [162], MOMIS [11], DIKE [134] and IBIS [25].

It should be easy to see that the LAV approach favors the extensibility of the integration system, and provides a more appropriate setting for its maintenance. For example, adding a new source to the system requires only to provide the definition of the source, and does not necessarily involve changes in the global schema. On the contrary, in the GAV approach, adding a new source typically requires changing the definition of the concepts in the global schema. On the other hand, defining the views associated to the global elements implies precisely understanding the relationships among the sources, that is in general a non-trivial task. A comparison between the two approaches is reported in [150, 111, 108, 23].

A different approach could be to specify the mapping by combining LAV and GAV views together. This approach, called *global-local-as-view* (GLAV), is more recent and has so far drawn little attention in the literature. [78, 69, 70] are interesting examples of the GLAV approach.

1.2 Materialized vs. Virtual Integration

With respect to the data explicitly managed by a data integration system, it is possible to follow two different approaches, called *materialized* and *virtual*. In the materialized approach, the system computes the extensions of the structures in the global schema by replicating the data

at the sources. In the virtual approach, data residing at the sources are accessed during query processing, but they are not replicated in the integration system.

The materialized approach to data integration is the most closely related to *Data Warehousing* [99, 55, 148, 34, 100]. In this context, data integration activities are relevant for the initial loading and for the refreshing of the Warehouse, but not directly for query processing. Obviously, maintenance of replicated data against updates to the sources is a central aspect in this context, and the effectiveness of maintenance affects timeliness and availability of data. A naïve way to deal with this problem is to recompute materialized data entirely from scratch in the presence of changes at the sources. This is expensive and makes frequent refreshing impractical. The study of the materialized data management is an active research topic that is concerned with both the problem of choosing the data to be materialized into the global schema, and reducing the overhead in the re-computation. However, an in-depth analysis of this problem is outside the scope of this thesis and we refer the interested reader to [156, 97, 14].

As already said, in the virtual approach to data integration, sources are accessed on the fly, i.e., each time a user query is posed to the system, since it provides the user only with a virtual global schema, i.e., a schema whose extension is not stored, e.g., in a Data Warehouse. Despite such differences, most of the issues that arise for the virtual approach, are relevant also for the materialized approach: from the initial loading, where the identification of the relevant data within the sources is critical, to the refreshment process, which may require a dynamic adaptation depending on the availability of the sources, as well as on their reliability and quality that may change over time. Moreover, the extraction of data from a primary Data Warehouse for Data Mart applications, where the primary Warehouse is now regarded as a data source, can be treated in a similar way.

In this thesis, we focus on the virtual approach. In particular we do not address the problem of updates on the sources. However, according to the above discussion, our studies are relevant to some extent for the materialized approach as well.

1.3 Querying Data Integration Systems

The problem of query processing is concerned with one of the most important issues in a data integration system, i.e., the choice of the method for computing the answer to queries posed in terms of the virtual global schema only on the basis of the data residing at the sources. The main issue is that the system should be able to re-express such queries in terms of a suitable set of queries posed to the sources, hand them to the sources, and assemble the results into the final answer.

It is worth noticing that, whereas query processing in the LAV approach has been always regarded as a difficult task, this problem has traditionally been considered much easier in the GAV approach, where it has been assumed that answering a query means *unfolding* its atoms according to their definitions on the sources. The reason can be seen in the fact that, in LAV, the views in the mapping provide in general only a partial knowledge about the data that satisfy the global schema, hence query processing is inherently a form of reasoning in the presence of incomplete information [98, 1]. In other words, since several possibilities of populating the global

schema with respect to the source extensions may exist, the semantics of a LAV system has to be given in terms of several database instances for the global schema, which have to be taken into account in processing a user query. We say that each such database satisfies the global schema with respect to the mapping. Conversely, in GAV, the mapping essentially specifies a single database for the global schema: evaluating the query over this database is equivalent to evaluating its unfolding over the sources [80, 11].

Example 1.3.1 Consider again Example 1.1.2, and suppose we have the following user query:

$$q(X, Z) \leftarrow \mathit{movie}(X, 1998, Y), \mathit{review}(X, Z)$$

asking for title and critique of movies produced in 1998. The unfolding produces the query

$$q(X, Z) \leftarrow s_1(X, 1998, Y), s_2(X, Z)$$

The unfolded query can be directly evaluated on the sources, thus retrieving the answers. ∎

In the LAV approach, query processing has been traditionally solved by means of *query rewriting*, where query processing is forced to be performed in two steps: in the first step the query is reformulated in terms of the views, and in the second the obtained query is evaluated on the view extensions, i.e., a database instance for the source schema. In query rewriting we want to reformulate the user query in terms of a fixed language referring to the alphabet used for specifying the source schema; the problem is that, since such language is fixed, and often coincides with the language used for the user query, there may be no rewriting that is *equivalent* to the original query. To face this problem, works on the LAV approach have concentrated on computing the *maximally contained rewriting*, that is the "best" possible rewriting, in the sense that it contains all other rewritings for the given query.

Example 1.3.2 Consider again Example 1.1.1, and suppose to have the same user query of Example 1.3.1:

$$q(X, Z) \leftarrow \mathit{movie}(X, 1998, Y), \mathit{review}(X, Z)$$

A rewriting of such a query is

$$q_r(X, Z) \leftarrow s_1(X, 1998, Y), s_2(X, Z)$$

Indeed, the body of q_r refers only to the source relations, and hence can be directly evaluated over the source extensions. Moreover, if we unfold q_r according to the view definitions we obtain the following query over the global schema

$$q_r(T, R) \leftarrow \mathit{movie}(T, 1998, D), \mathit{movie}(T, Y, D'), \mathit{review}(T, R), \mathit{european}(D')$$

in which we deleted the atom $1998 \geq 1960$, which evaluate to *true*. It is immediate to verify that q_r is contained in q. Furthermore q_r is th maximally contained rewriting. ∎

A different approach, more general than query rewriting, consists in not posing any limitation on how the query is to be processed: all possible information, in particular the view extensions, can be used now for computing the answers to the query. This approach is commonly called *query answering*. We point out that the ultimate goal of query answering is to provide the *certain answers* to a user query, i.e., compute the intersection of the answer sets obtained by evaluating the query over any database that satisfies the global schema. Therefore, the notions of query answering and query rewriting are different, and we cannot always conclude that the evaluation on the sources of the maximally contained rewriting returns the certain answers to the user query. Nonetheless, this property holds for the common and widely studied case in which the views in the mapping are conjunctive queries [37]. Rewritings that are able to solve the query answering problem are called *perfect rewriting*.

An accurate survey on query processing in both the LAV and the GAV approaches will be given in Chapter 4. However, it is worth noticing since by now that more complex GAV frameworks have been recently considered in the literature, in which the semantics of the system is given in terms of a set of databases rather than the single database constructed according to the mapping, and techniques for query answering have been proposed that are more involved than simple unfolding [22, 24]. These approaches are tightly connected with the problem of query processing under integrity constraints in the presence of incomplete data, which is addressed in the next section.

1.4 Dealing with Incomplete Data

As already said, query processing in LAV has been traditionally considered a form of reasoning in the presence of incomplete information. Hence, sources in LAV data integration systems are generally assumed to be *sound*, but not necessarily complete, i.e., each source concept is assumed to store only a subset of the data that satisfy the corresponding view on the global schema. Not the same approach has been followed for processing queries in GAV, where, actually, the form of the mapping straightforwardly allows for the computation of a global database instance over which the user queries can be directly evaluated. Notice that proceeding in this way is analogous to unfolding the user queries as described in the above section.

Example 1.4.1 Consider again Example 1.1.2, and suppose to have the following source extension $\mathcal{D} = \{s_1(\mathsf{Platoon}, 1980, \mathsf{O.\ Stone}), s_3(\mathsf{F.\ Fellini})\}$, and a user query $q(Z) \leftarrow movie(X, Y, Z)$ asking for all the directors in the relation *movie*. It is easy, according to the GAV mapping, to construct the global database $\mathcal{B} = \{movie(\mathsf{Platoon}, 1986, \mathsf{O.\ Stone}), european(\mathsf{F.\ Fellini})\}$, over which we evaluate the user query, thus obtaining the answer set $\{\mathsf{O.\ Stone}\}$. ∎

Answering a user query as done in the above example, actually means assuming that the views in the mapping are *exact*, i.e., that they provide exactly the data that satisfy the corresponding global relation. However, also in GAV systems it may happen that the sources provide only a subset of the data that satisfy the corresponding relations in the global schema, hence views in the mapping should be considered sound rather than exact. This becomes particularly relevant when integrity constraints are specified on the global schema, as shown in [24, 20].

Example 1.4.1 (contd.) Assume now that a constraint on the global schema schema imposes that every director in the relation *european* is a director of at least one movie. Assume also that the mapping is sound. In this case we know that Fellini is a director of at least one movie, even if we do not know which is the movie and in which year it has been realized. Nonetheless, under the sound assumption, the answer to our user query q asking for all directors should be now $\{\mathsf{O.\ Stone}, \mathsf{F.\ Fellini}\}$. ∎

The above example shows that in the presence of incomplete data with respect to integrity constraints specified on the global schema, unfolding is in general not sufficient to answer a user query in GAV, whereas reasoning on the constraints is needed in order to compute the certain answers to the query. Moreover, it should be easy to see that reasoning on the constraints is needed also in the LAV approach: consider again Example 1.1.1, where now integrity constraints on the global schema, user query and source extensions are as in Example 1.4.1; without taking into account the inclusion dependency, we would not get the certain answers also in this case.

In conclusion, in order to model complex data integration scenarios, the specification of different forms of integrity constraints on the global schema should be allowed, and different assumptions on the mapping should be enabled, in both the LAV and the GAV framework. Roughly speaking, such assumptions indicate how to interpret the data that can be retrieved from the sources with respect to the data that satisfy the corresponding portion of the global schema, and allow for different forms of reasoning on the constraints.

1.5 Dealing with Inconsistent Data

In the above section we have dealt with the problem of query answering in the presence of incomplete information, and we have shown that, in this setting, some forms of integrity constraints expressed on the global schema, e.g., inclusion dependencies as in Example 1.4.1, allow us to deduce missing information that is needed to compute certain answers to user queries. In such a situation, the violation of an inclusion dependency cannot be considered a "real" inconsistency, since data migrating form the sources through the global schema can be actually reconciled in the global schema in such a way that both the constraints and the assumption on the mapping are satisfied.

Of course, situations arise in which this does not happen. Consider, for instance, the case in which the mapping is sound and a key dependency is violated on a relational global schema: the soundness assumption on the mapping does not allow us to disregard tuples with duplicate keys, hence the data are inconsistent with respect such constraint.

This is a common situation in data integration, since integrity constraints are not related to the underlying data sources, but they are derived from the semantics of the global schema, or, in other words, from the real world. Hence, we cannot expect independent and autonomous data sources to produce data which respect these constraints. On the other hand, since most of the data could satisfy such constraints, it seems unreasonable to consider the entire system inconsistent.

Classical assumptions on the views do not allow us to properly handle data inconsistency, since they generally lead to a situation in which no global database exists that satisfy both the integrity constraints and the assumption on the mapping, and it is not possible to provide meaningful answers to user queries. Conversely, we need now a characterization that allows us to get *consistent answers* from inconsistent data integration systems. A possible solution to this problem is to characterize the semantics of a data integration system in terms of those databases that satisfy the integrity constraints on the global schema, and approximate "at best" the satisfaction of the assumptions on the mapping, i.e., that are *as close as possible* to the semantics interpretation of the mapping.

The semantic problem that arises in this context is similar to the one underlying the notion of *database repair* introduced by several works in the area of *inconsistent databases* [71, 17, 7, 8, 123, 57, 86]. Such works propose techniques aiming to both repair databases, i.e, obtaining consistent database instances from inconsistent ones, and compute *consistent query answers* from inconsistent databases.

We point out that such studies basically apply to a single database setting [71, 17, 7, 8], and that almost all the proposed techniques can be employed in a data integration setting only by assuming that the mapping is GAV and views in the mapping are exact [123, 57, 86]. Only recently, data inconsistency in a LAV data integration setting has been studied in [13, 16]. Furthermore, in all mentioned works, the methods proposed for consistent query answering basically apply only to the class of "universally quantified constraints" [2]. Two exceptions are [86], where a preliminary study on existentially quantified inclusion dependencies is presented, and [16], where, however, the same problem is tackled under restricted assumptions. No other proposals explicitly deals with the problem of query answering in data integration systems with inconsistent data sources: actually, the current integration methodologies deal with inconsistency in a preliminarily *data cleaning and reconciliation* phase, in which all inconsistencies are "cleaned", by ad-hoc updating algorithms. Therefore, query answering is performed when data are consistent. We will briefly discuss this matter in Section 1.7.

1.6 Source Access Limitations

Both in the LAV and in the GAV approach, it may happen that a source presents some limitations on the types of accesses it supports. A typical example is a web source accessible through a form where one of the fields must necessarily be filled in by the user. This can be modelled by specifying the source as a relation supporting only queries with a selection on a column. Suitable notations have been proposed for such situations [143], and the consequences of these access limitations on query processing in integration systems have been investigated in several papers [143, 121, 77, 159, 126, 62, 119, 21].

Generally speaking, to answer queries over such sources one generally needs to start from a set of constants (provided e.g., by the user filling in a form, or taken from a source without access limitations) to bind attributes. Such bindings are used to access sources and there obtain new constants which in turn can be used for new accesses, as shown in the following example, taken from [20].

Example 1.6.1 Suppose we have two sources with access limitations, say s_1 and s_2. Source s_1 stores information about cars: given a person (required constant), s_1 provides model, plate number, and color of the cars owned by the person. Source s_2 provides a list of persons with their address. Suppose we are searching for all the plate numbers of Ferrari's. Accessing only source s_1, where the information of interest is stored, is impossible, because we do not know any name of a car owner. But we can retrieve owner names from s_2, use them to query s_1, and select from the obtained tuples those in which the car model is Ferrari. ∎

The example above shows how apparently useless sources can be exploited by retrieving from them values that are used to access other sources. In general, query answering in the presence of access limitations requires the evaluation of a recursive query plan, as shown in [62] for the LAV approach and in [119] for the GAV approach.

Since source accesses are costly, an important issue is how to minimize the number of accesses to the sources while still being guaranteed to obtain all possible answers to a query. [119, 120] discuss several optimizations that can be made at compile time, during query plan generation. The basic idea is to exploit knowledge about the sources expressed by means of integrity constraints over the source schema in order to detect unnecessary source accesses. The technique works for a subclass of the conjunctive queries. In [21], a more general technique is presented that applies to the class of conjunctive queries: besides the integrity constraints on the sources, also the tuples extracted from the sources are taken into account so as to infer information necessary to avoid useless accesses.

In this thesis we do not deal with the problem of limitations in accessing data sources. For more details on this matter we refer the reader to [143, 77, 20].

1.7 Data Cleaning and Reconciliation

Data retrieved from the sources have to be reconciled, converted and combined in order to make them fit into the structures of the global schema. This is especially true when the sources are independent on each other, and they are not under the control of the integration system. Data Cleaning and Reconciliation refers to a number of issues arising when considering integration at the extensional/instance level.

A first issue in this context is the interpretation and merging of the data provided by the sources. Interpreting data can be regarded as the task of casting them into a common representation. Moreover, the data returned by various sources need to be converted and merged to provide the data integration system with the requested information. Accordingly, data cleaning problems can be grouped in two main categories: *differences in representation of the same data* and *invalid data*.

(i) When gathering information from different data sources, it is likely that the same information is represented in different ways in different sources. [142] identifies three main categories of such conflicts. *Naming conflicts* arise when the same name is used for different objects, or when different names are used for the same object. *Structural conflicts*

are more general, and occur when the difference in the representation lies in the structure of the representation itself. Examples of structural conflicts are the use of different data formats in different sources for the same field, or the fact that data that are represented by more than one field in a source may be represented in a single field in another one. Other structural conflicts, in the relational model, are due to different relational modelling in different sources, e.g., attribute vs. relation representation, different data types, etc. Finally, *data conflicts* appear only at the instance level, and are mainly related to different value representations. For example, the currency may be expressed in Japanese Yen in a source and in German Marks in another.

(ii) Invalid data can be caused by extracting data from multiple sources, or they can exist in a single source, due to incorrect data entries. A slightly more complex problems is due to inconsistencies among different fields of the same record; for example, a record regarding a person may have the value "12 December 1973" for the date of birth and the value "12" for the age. Violation of functional dependencies within a table is another typical example of such inconsistencies. We have already discussed this problem when considering incomplete and inconsistent data with respect to integrity constraints expressed on the global schema of an integration system. In particular we have faced the problem from a "semantic" point of view, in the same spirit of several works on reasoning in the presence of incomplete information and on inconsistent databases. We point out here that such "semantic approach" can be seen as an alternative to using data cleaning techniques, in general based on heuristics and characterized by a more pragmatic spirit [15].

A major problem in data cleaning is that of overlapping data [95, 130, 131, 144, 132], also referred as *duplicate elimination problem* or *merge/purge problem*. This problem arises when different records of data representing the same real-world entity are gathered in a data integration system. In this case, duplicates have to be detected and merged. Most efforts in data cleaning have been devoted to the solution of the duplicate detection problem, which proves to be a central issue.

In [95, 131] methods are proposed which are based on the use of a *sliding windows* moving along an ordered list of records to be analyzed: a key is associated to each record and records in the windows having the same key are merged into a single record. Obviously, these methods heavily depends on the sorting, and therefore on the key, both based on heuristics.

An alternative to window scan consists in partition the records in *clusters*, where in each cluster records are stored, that match each other according to a certain criterion (see, e.g., [131]). Duplicate elimination can be performed on each cluster separately, and in parallel.

Finally, *knowledge-based approaches* have been used in data cleaning [79, 124]. This approaches favor the use of declarative specifications for the matching operations. However, human validation and verification of the results is in general needed at the end of the data cleaning process.

In this thesis, we will not deal with all the aspects of the problem of data cleaning and reconciliation, but we will concentrate on the problem of invalid data with respect to integrity constraints on the global schema. As already said, our approach can be seen to some extent as

a "semantic" data cleaning.

1.8 Contribution of the Thesis

This thesis addresses the topic of data integration from several points of view, and faces many of the issues that we have analyzed so far in this introduction. We briefly summarize in the following the main contributions we provide.

(i) First of all, we address the task of modelling data integration systems, described in Section 1.1, and provide a comprehensive formal framework which allows for the specification of powerful forms of integrity constraints on the global schema, and of both LAV and GAV mappings between the global schema and the sources. More specifically, we consider a setting in which both the sources and the global schema are relational, and it is possible to define on the global schema classical *Key Dependencies* (KDs), *Inclusion Dependencies* (IDs), expressing that (a projection of) a relation is included in (a projection of) another relation, and *Exclusion Dependencies* (EDs), expressing that two relations (or projections on them) are disjoint.

It is worth noticing that, even if described in terms of the relational model, our framework can be easily generalized in order to classify all data integration approaches found in the literature that are based on the design of a global schema [31].

(ii) We consider data sources that may result incomplete or inconsistent with respect the integrity constraints allowed in our framework. To properly deal with such scenario, we first consider three classical assumptions on the mapping (either LAV or GAV): besides the *sound* and *exact* assumptions that we have described in Section 1.4, we also consider the *complete* assumption, adopted when data that can be retrieved at the sources are complete with respect to the satisfaction of the corresponding portion of the global schema, but are not necessarily sound. Then, in the spirit of the works on inconsistent databases, we introduce suitable relaxations of such assumptions, thus defining the *loose semantics*, which allow us also to deal with inconsistent data.

(iii) For both strict and loose semantics, we study non-trivial cases of query answering under constraints: we identify the frontier between decidability and undecidability for this problem, and we establish the computational complexity of the decidable cases. More specifically, for KDs and IDs specified on the global schema, we first show undecidability of query answering in the general case, and then we define the maximal class of IDs for which query answering is decidable. We call such IDs *non-key-conflicting* inclusion dependencies (NKCIDs). An interesting property of NKCIDs is that they do not interact with KDs, therefore they can be processed separately from the KDs in order to provide certain answers to user queries. Then, the question arises if also in the presence of EDs the separation property holds. We show that actually EDs preserve such property, and that we can solve query answering by separately taking into account the set of IDs and the set of KDs and EDs that are logical consequence of the IDs and the original EDs.

(iv) We provide a sound and complete query rewriting technique for the GAV framework under the loosely-sound semantics, a suitable relaxation of the sound semantics that allows for consistent query answering of incomplete and inconsistent data. In our technique, the integrity constraints and the user query are intensionally compiled in function-free logic program, which is then evaluated over the source extensions. We point out that the treatment of integrity constraints is strongly modular, since it is based on the separation property described at point (iii). In particular, the logic program we produce is a union of conjunctive queries if the data retrieved at the sources are consistent with KDs and EDs, otherwise it is a Datalog$^\neg$ program, a well-known extension of Datalog that allows for the use of negation in the body of the rules [105, 67]. In the latter case, the query has to be evaluated by a stable model engine, such as DLV [109] or Smodels [133].

(v) We provide a sound and complete query rewriting technique under the sound semantics in the LAV framework. More specifically, we define an off-line compiling technique that, starting from a LAV system specification, produces the specification of a GAV system which is query-equivalent to the initial LAV system. After such a compilation, we can reuse the query processing technique defined for GAV systems also in the LAV case.

(vi) We define optimization techniques for speeding up the evaluation over large databases of the logic programs produced in the rewriting phase. Indeed, a drawback of this approach is that with current implementations of stable model engines the evaluation of queries over large data sets quickly becomes infeasible because of lacking scalability. This calls for suitable optimization methods [16]. We point out that the technique that we propose here apply also to the class of universally quantified constraints (in the absence of existentially quantified IDs).

(vii) Finally, we present the DIS@DIS system, a data integration prototype incorporating techniques described in the thesis. A first version of the system has been released [27], and the first experiments are very encouraging.

To the best of our knowledge, our work is the first to provide a thorough analysis of data integration (both in the GAV and in the LAV approach) in the presence of KDs, EDs, and cyclic IDs. Moreover, our prototype represents the first implementation of query processing techniques in the presence of incomplete and inconsistent data.

It is worth noticing that the intensional and modular characteristics of our techniques greatly simplified the implementation of the algorithm and the architecture of our system, in which components for query processing are decoupled from the extensional layer devoted to the extraction of the data from the sources.

Finally, we point out that the classes of dependencies studied in this thesis, i.e., IDs, KDs and EDs, are the kinds of constraints specified on relational schemas that are constructed on the basis of Entity Relationship (ER) specifications of the domain of interest. Furthermore, these dependencies represent the core constraint language of the ER model itself, and hence integrating data in the presence of such constraints can be seen as a first step towards the integration of (simple) ontologies.

1.9 Organization of the Thesis

The thesis is organized as follows. In Chapter 2 we present some theoretical background. In particular, we illustrate the query languages which we deal with in the thesis, present the basic notions of the relational model, and the computational complexity theory. In Chapter 3 we present a formal framework for data integration which is based on the relational model with integrity constraints described in Chapter 2. We also provide a definition of our loose semantics for inconsistent data. In Chapter 4 we survey the most important query processing algorithms proposed in the literature for LAV, and we describe the principal GAV data integration systems and the form of query processing they adopt. Part of this material has been published in [28]. In Chapter 5 we study decidability and complexity of query answering when inclusion, key and exclusion dependencies are expressed on the global schema. For the sake of clarity, in this chapter we abstract from the integration context, and address the problem in a single database setting. The results provided in this chapter extend our previous studies presented in [29]. In Chapter 6, we study query answering and rewriting in GAV data integration systems. We generalize to this framework undecidability and complexity results obtained in Chapter 5, and provide effective algorithms for query answering by rewriting under the loosely-sound semantics. A preliminary version of this material appeared in [107, 30]. In Chapter 7, we address the LAV framework. We extend here to the LAV case the undecidability and complexity results of Chapter 5, and provide effective algorithms for query answering and rewriting in LAV data integration systems under the sound semantics. In Chapter 8, we provide techniques for the efficient evaluation of logic programs modelling query answering in data integration systems. This chapter is a (slightly) extended version of [63]. In Chapter 9 we present the DIS@DIS system. The material in this chapter is based on [26]. Finally, Chapter 10 concludes the thesis.

Chapter 2

Theoretical Background

In this chapter we recall some theoretical notions that will be useful for following discussions. In particular, we illustrate the classes of queries which we deal with in the thesis, present the basic notions of the relational model, on which we will construct our formal framework for data integration, and briefly recall the complexity classes that will be mentioned in the thesis. This chapter is intended to be a brief introduction to such matters, while an exhaustive treatment of them is out of our scopes. For further background we refer the reader to [82, 2, 136].

2.1 Query Languages

A *term* is either a variable or a constant. An *atom* is an expression $p(T_1, \ldots, T_n)$, where p is a predicate (relation) of arity n and $T_1, \ldots, T_n$ are terms.

A *Datalog*$^\neg$ rule ρ is an expression of the form

$$r(\vec{\mathbf{u}}) \leftarrow r_1(\vec{\mathbf{u}}_1), \ldots, r_k(\vec{\mathbf{u}}_k),\ \mathit{not}\ r_{k+1}(\vec{\mathbf{u}}_{k+1}), \ldots,\ \mathit{not}\ r_{k+m}(\vec{\mathbf{u}}_{k+m}) \tag{2.1}$$

where $k, m \geq 0$, $r(\vec{\mathbf{u}}), r_1(\vec{\mathbf{u}}_1), \cdots, r_{k+m}(\vec{\mathbf{u}}_{k+m})$ are atoms, and there do not exist $1 \leq i \leq k$ and $k+1 \leq j \leq k+m$ such that $r_i = r_j$ (i.e., the rule is *consistent*). Each variable occurring in $\vec{\mathbf{u}}$ must occur in at least one of $\vec{\mathbf{u}}_1, \ldots, \vec{\mathbf{u}}_{k+m}$ (i.e., the rule is *safe*). $r(\vec{\mathbf{u}})$ is the *head* of the rule, denoted $\mathit{head}(\rho)$, while the conjunction $r_1(\vec{\mathbf{u}}_1), \ldots,\ \mathit{not}\ r_{k+m}(\vec{\mathbf{u}}_{k+m})$ constitutes the *body*, denoted $\mathit{body}(\rho)$. The number of terms in $\vec{\mathbf{u}}$ is the arity of ρ. If $\vec{\mathbf{u}}$ is empty, the arity is 0 and the rule is *boolean*. Variables in $\mathit{head}(\rho)$ are called *distinguished variables*, whereas variables appearing only in $\mathit{body}(\rho)$ are called *non-distinguished variables*. If $k + m = 0$, the rule is called *fact*, and we omit the "$\leftarrow$" sign. If $k \neq 0$ and $m = 0$ then ρ is a *positive Datalog* (or simply a Datalog) rule, also called *conjunctive query (CQ)*. A *literal* l is either an atom p, or a negated atom *not* p. In the former case l is *positive*, in the latter l is *negative*.

A *Datalog*$^\neg$ *program* $\mathcal{P}$ is a finite set of Datalog$^\neg$ rules. $\mathcal{P}$ is a *positive* Datalog (or simply, a Datalog) program, if all its rules are positive. We can associate the program $\mathcal{P}$ with a labelled graph $G = \langle V, E \rangle$, where the set of vertices V coincides with the set of predicates appearing in $\mathcal{P}$, and E is the set of edges. An edge from r to s belongs to E if and only if there exists a

rule ρ in $\mathcal{P}$ such that r is the predicate in $head(\rho)$ and s appears in a literal l_s in $body(\rho)$. If l_s is positive the edge is labelled with p, whereas if l_s is negative the edge is labelled with n. If every cycle in G comprises only edges labelled with p, the program $\mathcal{P}$ is said *stratified Datalog*$^\neg$ program, denoted Datalog$^{\neg_s}$. Furthermore, if the graph G is acyclic, $\mathcal{P}$ is said *non-recursive*

A (positive) non-recursive Datalog program in which all the rules present the same head predicate is a also called *union of conjunctive queries (UCQ)*, and is often written in the form

$$r(\vec{\mathbf{u}}) \leftarrow conj_1(\vec{\mathbf{u}}, \vec{\mathbf{w}}_1) \vee \cdots \vee conj_m(\vec{\mathbf{u}}, \vec{\mathbf{w}}_m) \tag{2.2}$$

where for each $i \in \{1, \ldots, m\}$ $conj_i(\vec{\mathbf{u}}, \vec{\mathbf{w}}_i)$ is a conjunction of atoms.

Predicate symbols in $\mathcal{P}$ can be either *extensional* (*EDB predicates*), i.e., defined by the facts of a database (see Section 2.2), or *intensional* (*IDB predicates*), i.e., defined by the rules of the program.

For any program $\mathcal{P}$, let $\mathcal{U}_\mathcal{P}$ (the Herbrand Universe) be the set of all constants appearing in $\mathcal{P}$ (if no constants appear in $\mathcal{P}$, $\mathcal{U}_\mathcal{P}$ contains an arbitrary constant). A *substitution* of variables σ is a finite set of the form $\sigma = \{X_1 \rightarrow T_1, \ldots, X_n \rightarrow T_n\}$ where each X_i is a variable and each T_i is a term, and the variables $X_1, \ldots, X_n$ are distinct. For each i we say that X_i and T_i unify in σ. Given a set of atoms D (resp. a rule ρ, a program $\mathcal{P}$) we indicate with $\sigma(D)$ (resp. $\sigma(\rho)$, $\sigma(\mathcal{P})$) the set obtained from D (resp. ρ, $\mathcal{P}$) by simultaneously replacing each occurrence of X_i with T_i. A term (resp. an atom, a literal, a rule or a program) is *ground*, if no variables occur in it. For any rule ρ in $\mathcal{P}$, $ground(\rho)$ denotes the set of rules obtained by applying all possible substitutions from the variables in ρ to elements of $\mathcal{U}_\mathcal{P}$. Let $B_\mathcal{P}$ be the set of all ground literals constructible from the predicate symbols appearing in $\mathcal{P}$ and the constants of $\mathcal{U}_\mathcal{P}$. An interpretation for $\mathcal{P}$ is any subset of $\mathcal{B}_\mathcal{P}$. The value of a ground positive literal g w.r.t. an interpretation I, $value_I(g)$, is $true$ if $g \in I$ and $false$ otherwise. The value of a ground negative literal not g is $not\ value_I(g)$. The truth value of a conjunction of ground literals $C = g_1, \ldots, g_n$ is the minimum over the values of the g_i, i.e. $value_I(C) = min(\{value_I(g_i) \mid 1 \leq i \leq n\})$. If $n = 0$, $value_I(C) = true$. A ground rule ρ is *satisfied* by I if $head(\rho) \geq body(\rho)$. An interpretation M for $\mathcal{P}$ is a model of $\mathcal{P}$ if M satisfies all ground rules of $\mathcal{P}$, denoted by $ground(\mathcal{P})$.

Let $\mathcal{P}$ be a positive Datalog program, the model-theoretic semantics assigns to $\mathcal{P}$ the set MM($\mathcal{P}$) of its *minimal models*, where a model M for $\mathcal{P}$ is minimal, if no proper subset of M is a model for $\mathcal{P}$.

The stable model semantics applies also to programs with negation. The *reduct* of a ground program $\mathcal{P}$ w.r.t. a set $X \subseteq B_\mathcal{P}$ is the positive ground program $\mathcal{P}^X$, obtained from $\mathcal{P}$ by

- deleting each rule having a ground literal *not* p, such that p is in X;
- deleting the negative body from the remaining rules.

An interpretation M is a stable model of $\mathcal{P}$ if and only if $M \in$ MM($\mathcal{P}^M$). The set of stable models of $\mathcal{P}$ in denoted by SM($\mathcal{P}$)

It is well-known that for positive programs minimal models and stable models coincide, and that positive (resp. stratified) programs have a unique minimal (resp. stable) model.

In the following, given a Datalog$^\neg$ program $\mathcal{P}$, and a set of facts $\mathcal{D}$ that represent the extension of EDB predicates (i.e., facts of a database), we indicate with $\mathcal{P}^\mathcal{D}$ the union $\mathcal{P} \cup \mathcal{D}$, and say that $\mathcal{P}^\mathcal{D}$ is the evaluation of $\mathcal{P}$ over $\mathcal{D}$.

2.2 The Relational Model

We consider to have an infinite, fixed database domain $\mathcal{U}$ whose elements can be referenced by constants $c_1, \ldots, c_n$ under the *unique name assumption*, i.e., different constants denote different real-world objects.

A *relational schema* (or simply *schema*) $\mathcal{RS}$ is a pair $\langle \Psi, \Sigma \rangle$, where:

- Ψ is a set of predicates or relations, each with an associated arity that indicates the number of its attributes. The attributes of a relation $r \in \Psi$ of arity n (denoted as r/n) are represented by the integers $1, \ldots, n$;
- Σ is a set of *integrity constraints* expressed on the relations in Ψ, i.e., assertions on the relations in Ψ that are intended to be satisfied by database instances.

A *database instance* (or simply *database*) $\mathcal{DB}$ for a schema $\mathcal{RS} = \langle \Psi, \Sigma \rangle$ is a set of facts of the form $r(t)$ where r is a relation of arity n in Ψ and t is an n-tuple of constants of $\mathcal{U}$. We denote as $r^{\mathcal{DB}}$ the set $\{t \mid r(t) \in \mathcal{DB}\}$. A database $\mathcal{DB}$ for a schema $\mathcal{RS}$ is said to be *consistent* with $\mathcal{RS}$ if it satisfies all constraints expressed on $\mathcal{RS}$. The notion of satisfaction depends on the type of constraints defined over the schema.

The integrity constraints that we consider are *inclusion dependencies (IDs)*, *functional dependencies (FDs)*, *key dependencies (KDs)* and *exclusion dependencies (EDs)*. Some other minor classes of constraints will be introduced when needed. More specifically,

- an *inclusion dependency* is an assertion of the form $r_1[\mathbf{A}] \subseteq r_2[\mathbf{B}]$, where r_1, r_2 are relations in Ψ, $\mathbf{A} = A_1, \ldots, A_n$ $(n \geq 0)$ is a sequence of attributes of r_1, and $\mathbf{B} = B_1, \ldots, B_n$ is a sequence of distinct attributes of r_2. Therefore, we allow for repetition of attributes in the left-hand side of the inclusion[1]. A database $\mathcal{DB}$ for $\mathcal{RS}$ satisfies an inclusion dependency $r_1[\mathbf{A}] \subseteq r_2[\mathbf{B}]$ if for each tuple $t_1 \in r_1^{\mathcal{DB}}$ there exists a tuple $t_2 \in r_2^{\mathcal{DB}}$ such that $t_1[\mathbf{A}] = t_2[\mathbf{B}]$, where $t[\mathbf{A}]$ indicates the projection of the tuple t over $\mathbf{A}$;
- a *functional dependency* is an assertion of the form $r : \mathbf{A} \rightarrow B$[2], where r is a relation in Ψ, $\mathbf{A} = A_1, \ldots, A_n$, is a sequence of distinct attributes of r, and B is an attribute of r. A database $\mathcal{DB}$ for $\mathcal{RS}$ satisfies a functional dependency $r : \mathbf{A} \rightarrow B$ if for each $t_1, t_2 \in r^{\mathcal{DB}}$ such that $t_1[\mathbf{A}] = t_2[\mathbf{A}]$ we have that $t_1[B] = t_2[B]$;
- a *key dependency* is an assertion the form $key(r) = \mathbf{A}$, where r is a relation in Ψ, and $\mathbf{A} = A_1, \ldots, A_n$ is a sequence of distinct attributes of r. A database $\mathcal{DB}$ for $\mathcal{RS}$ satisfies

[1] Repetitions in the right hand side force equalities between attributes of the relation in left hand side, hence they imply constraints that we do not study in the thesis. See [43, 128] for more details on these dependencies.

[2] Functional dependencies are given in normal form [2], i.e., with a single attribute in the right-hand side. Non-normal form FDs can be always expressed by means of a set of FDs in normal form.

a key dependency $key(r) = \mathbf{A}$ if for each $t_1, t_2 \in r^{\mathcal{DB}}$ with $t_1 \neq t_2$ we have $t_1[\mathbf{A}] \neq t_2[\mathbf{A}]$. We assume that at most one key dependency is specified for each relation;

- an *exclusion dependency* is an assertion of the form $(r_1[\mathbf{A}] \cap r_2[\mathbf{B}]) = \emptyset$, where r_1 and r_2 are relations in Ψ, $\mathbf{A} = A_1, \ldots, A_n$ and $\mathbf{B} = B_1, \ldots, B_n$ are sequences of attributes of r_1 and r_2, respectively. A database $\mathcal{DB}$ for $\mathcal{RS}$ satisfies an exclusion dependency $(r_1[\mathbf{A}] \cap r_2[\mathbf{B}]) = \emptyset$ if there do not exist two tuples $t_1 \in r_1^{\mathcal{DB}}$ and $t_2 \in r_2^{\mathcal{DB}}$ such that $t_1[\mathbf{A}] = t_2[\mathbf{B}]$.

Notice that KDs are particular FDs, and thus they could be expressed in the form that we use for FDs. However, since the class of KDs plays a crucial role in our framework, we make use of the different notations described above.

Sets of inclusion, functional, key and exclusion dependencies expressed on the database schema are denoted by Σ_I, Σ_F, Σ_K, and Σ_E, respectively. Furthermore, we use $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E, \Sigma_F \rangle$ as a shortcut for $\mathcal{RS} = \langle \Psi, \Sigma_I \cup \Sigma_K \cup \Sigma_E \cup \Sigma_F \rangle$. In the absence of some kind of dependencies, we disregard the corresponding symbol. When a database $\mathcal{DB}$ satisfies all dependencies in Σ (resp. Σ_I, Σ_F, Σ_K, or Σ_E), we say that $\mathcal{DB}$ is consistent with Σ (resp. Σ_I, Σ_F, Σ_K, or Σ_E).

Finally, a *relational query* (or simply *query*) over $\mathcal{RS}$ is a formula (generally expressed in a language which amounts to a fragment of Datalog$^\neg$) whose EDB predicates are relations in Ψ. Each query q is intended to extract a set of tuples of constants of $\mathcal{U}$, thus q has an associated arity that indicates the number of constants of each tuple.

2.3 Complexity Classes

We assume that the reader is familiar with the basic notions of computational complexity, and NP-completeness [136, 81]. P^A (NP^A) is the class of problems that are solved in polynomial time by a deterministic (nondeterministic) Turing machine using an oracle for A, i.e., that solves in constant time any problem in A. Furthermore, co-A is the class of problems that are complement of a problem in A. The classes Σ_k^p and Π_k^p of the *Polynomial Hierarchy* are defined as follows:

$$\Sigma_0^p = \Pi_0^p = P \text{ and for all } k \geq 1, \Sigma_k^p = NP^{\Sigma_{k-1}^p} \text{ and } \Pi_k^p = \text{co-}\Sigma_k^p$$

In particular $\Sigma_2^p = NP^{NP}$, and $\Pi_2^p = \text{co-}\Sigma_2^p$, i.e., Σ_2^p is the class of problems that are solved in polynomial time by a nondeterministic Turing machine that uses an NP-oracle, and Π_2^p is the class of problems that are complement of a problem in Σ_2^p. Finally, PSPACE (NPSPACE) is the class of problems that can be solved by a deterministic (nondeterministic) Turing machine that uses a polynomially bounded amount of memory. By Savitch's theorem [146] it follows that PSPACE=NPSPACE (the theorem actually applies to a much more general class of space bounds).

In this thesis we consider the complexity of the query languages described in Section 2.1 in different relational settings. In particular we address two kinds of complexity [153]:

- the *data complexity*, which is the complexity with respect to the size of the underlying database instance, i.e., when the relational query and the schema are considered fixed whereas the database instance is considered as an input to the problem, and

- the *combined complexity*, which is the complexity with respect to the size of the database instance, the query and the schema, i.e., when the query, the schema and the database are considered as the input to the problem[3].

[3]The query (or expression, schema) complexity, which is the complexity with respect to the query and the schema (but not the database) is not addressed in this thesis.

Chapter 3

Formal Framework for Data Integration

Informally, a data integration system consists of a (virtual) global schema, which specifies the global (user) elements, a source schema, which describes the structure of the sources in the system, and a mapping, which specifies the relationship between the sources and the global schema. User queries are posed on the global schema, and the system provides the answers to such queries by exploiting the information supplied by the mapping and accessing the sources that contain relevant data. Thus, from the syntactic viewpoint, the specification of an integration system depends on the following parameters:

- The form of the global schema, i.e., the formalism used for expressing global elements and relationships between global elements. Several settings have been considered in the literature, where, for instance, the global schema can be relational [83], object-oriented [11], semi-structured [125], based on Description Logics [103, 33], etc.;
- The form of the source schema, i.e., the formalism used for expressing data at the sources and relationships between such data. In principle, formalisms commonly adopted for the source schema are the same mentioned for the global schema;
- The form of the mapping. Two basic approach have been proposed in the literature, called respectively *global-as-view* (GAV) and *local-as-view* (LAV) [111, 150, 108] . The GAV approach requires that the global schema is defined in terms of the data sources: more precisely, every element of the global schema is associated with a view, i.e., a query, over the sources, so that its meaning is specified in terms of the data residing at the sources. Conversely, in the LAV approach, the meaning of the sources is specified in terms of the elements of the global schema: more exactly, the mapping between the sources and the global schema is provided in terms of a set of views over the global schema, one for each source element;
- The language of the mapping, i.e., the query language used to express views in the mapping;

- The language of the user queries, i.e., the query language adopted by users to issue queries on the global schema.

Let us now turn our attention on the semantics. According to [108], the semantics of a data integration system is given in terms of instances of the elements of the global schema (e.g., one set of tuples for each global relation if the global schema is relational, one set of objects for each global class if it is object-oriented, etc.). Such instances have to satisfy *(i)* the integrity constraints expressed between elements of the global schema, and *(ii)* the mapping specified between the global and the source schema.

Roughly speaking, the notion of satisfying the mapping depends on how the data that can be retrieved from the sources are interpreted with respect to the data that satisfy the corresponding portion of the global schema. Three different assumptions have been considered for such interpretation: the assumption of *sound* mapping, adopted when all data that can be retrieved at the sources satisfy the corresponding portion of the global schema but may result incomplete; the assumption of *complete* mapping, adopted when no data other than those retrieved at the sources satisfy the corresponding portion of global schema, i.e., they are complete but not all sound; the assumption of *exact* mapping, when data are both sound and complete.

In the following we provide a precise characterization of the concepts informally explained above. In particular, we consider a relational setting, i.e., the global and the source schema are expressed in the relational model. Nonetheless, our framework can be easily generalized to different data models.

3.1 Syntax

A data integration system $\mathcal{I}$ is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where:

- $\mathcal{G}$ is the *global schema* expressed in the relational model with inclusion, key and exclusion dependencies, i.e., $\mathcal{G} = \langle \Psi, \Sigma \rangle$, where $\Sigma = \Sigma_I \cup \Sigma_K \cup \Sigma_E$;
- $\mathcal{S}$ is the *source schema* expressed in the relational model without integrity constraints, i.e., $\mathcal{S} = \langle \Psi_{\mathcal{S}}, \emptyset \rangle$. Dealing with only relational sources is not restrictive, since we can always assume that suitable wrappers present sources in the relational format. Furthermore, assuming that no integrity constraint is specified on $\mathcal{S}$ is equivalent to assuming that data satisfy constraints expressed on the sources in which they are stored. This is a common assumption in data integration, because sources are in general autonomous and external to the integration system, and satisfaction of constraints at the sources should be guaranteed by local data management systems;
- $\mathcal{M}$ is the *mapping* between $\mathcal{G}$ and $\mathcal{S}$. In our framework we consider both the GAV and the LAV mapping. More precisely,
  - the GAV mapping is a set of assertions of the form $\langle r_{\mathcal{G}}, q_{\mathcal{S}} \rangle$, where $r_{\mathcal{G}}$ is a global relation and $q_{\mathcal{S}}$ is the associated query over the source schema $\mathcal{S}$. In this thesis, we study the setting in which the language used to express queries in the GAV mapping is non-recursive Datalog$^{\neg}$;

– the LAV mapping is a set of assertions of the form $\langle r_{\mathcal{S}}, q_{\mathcal{G}} \rangle$, where $r_{\mathcal{S}}$ is a source relation and $q_{\mathcal{G}}$ is the associated query over the global schema $\mathcal{G}$. In this thesis, we study the setting in which queries in the LAV mapping are Conjunctive Queries.

Finally, a *query over* $\mathcal{I}$ (also simply called *user query* in the following) is a formula that specifies which data to extract from the integration system. Each user query is issued over the global schema $\mathcal{G}$, and we assume that the language used to specify user queries is Union of Conjunctive Queries.

3.2 Semantics

Intuitively, to define the semantics of a data integration system, we have to start with a set of data at the sources, and we have to specify which are the data that satisfy the global schema with respect to such data at the sources, according to the assumption adopted for the mapping. Thus, in order to assign the semantics to a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, we start by considering a *source database for* $\mathcal{I}$, i.e., a database $\mathcal{D}$ for the source schema $\mathcal{S}$. Then, we consider the assumption on $\mathcal{M}$, and denote it with $as(\mathcal{M})$, and pose $as(\mathcal{M}) = s$, c, or e, for the sound, complete, and exact assumption, respectively.

Based on $\mathcal{D}$ and $as(\mathcal{M})$, we now specify what the information content of the global schema $\mathcal{G}$ is. We call any database $\mathcal{B}$ for $\mathcal{G}$ a *global database* for $\mathcal{I}$. Formally, the semantics of $\mathcal{I}$ w.r.t. $\mathcal{D}$ and $as(\mathcal{M})$, is the set of global databases $\mathcal{B}$ for $\mathcal{I}$ such that:

(i) $\mathcal{B}$ is consistent with $\mathcal{G}$;

(ii) $\mathcal{B}$ satisfies $as(\mathcal{M})$ with respect to $\mathcal{D}$, i.e.,

– if $\mathcal{M}$ is GAV, $\mathcal{B}$ satisfies $as(\mathcal{M})$ if for each assertion $\langle r_{\mathcal{G}}, q_{\mathcal{S}} \rangle \in \mathcal{M}$ we have that
  (a) $r_{\mathcal{G}}^{\mathcal{B}} \supseteq q_{\mathcal{S}}^{\mathcal{D}}$ if $as(\mathcal{M}) = s$;
  (b) $r_{\mathcal{G}}^{\mathcal{B}} \subseteq q_{\mathcal{S}}^{\mathcal{D}}$ if $as(\mathcal{M}) = c$;
  (c) $r_{\mathcal{G}}^{\mathcal{B}} = q_{\mathcal{S}}^{\mathcal{D}}$ if $as(\mathcal{M}) = e$;
– if $\mathcal{M}$ is LAV, $\mathcal{B}$ satisfies $as(\mathcal{M})$ if for each assertion $\langle r_{\mathcal{S}}, q_{\mathcal{G}} \rangle \in \mathcal{M}$ we have that
  (a) $r_{\mathcal{S}}^{\mathcal{D}} \subseteq q_{\mathcal{G}}^{\mathcal{B}}$ if $as(\mathcal{M}) = s$;
  (b) $r_{\mathcal{S}}^{\mathcal{D}} \supseteq q_{\mathcal{G}}^{\mathcal{B}}$ if $as(\mathcal{M}) = c$;
  (c) $r_{\mathcal{S}}^{\mathcal{D}} = q_{\mathcal{G}}^{\mathcal{B}}$ if $as(\mathcal{M}) = e$;

Intuitively, in GAV (and in LAV) the three different assumptions allow us to model different situations in which queries over $\mathcal{S}$ (resp. relations in $\mathcal{S}$) provide (a) any subset of tuples that satisfy the corresponding relation in $\mathcal{G}$ (resp. query over $\mathcal{G}$), (b) any superset of such tuples, or (c) exactly such tuples.

The semantics of $\mathcal{I}$ w.r.t. $\mathcal{D}$ and $as(\mathcal{M})$, is denoted with $sem_{as(\mathcal{M})}(\mathcal{I}, \mathcal{D})$, where $as(\mathcal{M}) = s, c, e$ respectively for the sound, complete, or exact assumption. Obviously, $sem_{as(\mathcal{M})}(\mathcal{I}, \mathcal{D})$ contains in general several global databases for $\mathcal{I}$.

Finally, we give the semantics of queries. Formally, given a source database $\mathcal{D}$ for $\mathcal{I}$ and an assumption $as(\mathcal{M})$ on $\mathcal{M}$, we call *certain answers* (or simply *answers*) to a query q of arity n with respect to $\mathcal{I}$, $\mathcal{D}$ and $as(\mathcal{M})$, the set

$$ans_{as(\mathcal{M})}(q, \mathcal{I}, \mathcal{D}) = \{\langle c_1, \ldots, c_n\rangle \mid \langle c_1, \ldots, c_n\rangle \in q^{\mathcal{B}} \text{for each } \mathcal{B} \in sem_{as(\mathcal{M})}(\mathcal{I}, \mathcal{D}) \}$$

The problem of *query answering* is the problem of computing the set $ans_{as(\mathcal{M})}(q, \mathcal{I}, \mathcal{D})$. It should be easy to see that query answering in data integration systems is essentially a form of reasoning in the presence of incomplete information [152].

Example 3.2.1 Consider the relational schema $\mathcal{G}_0 = \langle \Psi_0, \Sigma_0\rangle$ where Ψ_0 contains the two relations[1] $player(Pname, Pteam)$ and $team(Tname, Tcity)$, and Σ_0 contains the ID $player[Pteam] \subseteq team[Tname]$, stating that every player is enrolled in a team of a city.

GAV mapping. Let us first construct a GAV data integration system in which $\mathcal{G}_0$ is the global schema, i.e., let us consider $\mathcal{I}_G = \langle \mathcal{G}_0, \mathcal{S}_G, \mathcal{M}_G\rangle$, where $\mathcal{S}_G$ contains three binary relations s_1, s_2 and s_3, and the mapping $\mathcal{M}_G$ is as follows:

$$\begin{aligned} \langle player, \quad & player(X, Y) \leftarrow s_1(X, Y) \\ & player(X, Y) \leftarrow s_2(X, Y)\rangle \\ \langle team, \quad & team(X, Y) \leftarrow s_3(X, Y)\rangle. \end{aligned}$$

Assume to have the source database $\mathcal{D}_G = \{s_1(\mathsf{a}, \mathsf{b}), s_1(\mathsf{e}, \mathsf{f}), s_2(\mathsf{a}, \mathsf{d}), s_3(\mathsf{b}, \mathsf{c})\}$ for $\mathcal{I}_G$, where $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{f}$ are constants of $\mathcal{U}$. It should be easy to see that $sem_e(\mathcal{I}_G, \mathcal{D}_G) = \emptyset$, since s_3 does not store the cities of the teams f and d. This in turn implies that query answering under the exact semantics in this example is meaningless, since every possible fact is a logical consequence of $\mathcal{I}$ and $\mathcal{D}$: for instance, the answer to the query that asks for all team names in *team*, i.e., $q(x) \leftarrow team(x, y)$, is the whole interpretation domain $\mathcal{U}$ (that is, every possible constant belongs to the extension of the query).

On the other hand, $sem_c(\mathcal{I}_G, \mathcal{D}_G) = \{\{player(\mathsf{a}, \mathsf{b}), team(\mathsf{b}, \mathsf{c})\}, \{team(\mathsf{b}, \mathsf{c})\}, \emptyset\}$. Furthermore, let be $\overline{\mathcal{B}} = \{player(\mathsf{a}, \mathsf{b}), player(\mathsf{a}, \mathsf{d}), player(\mathsf{e}, \mathsf{f}), team(\mathsf{b}, \mathsf{c})\}$, $sem_s(\mathcal{I}_G, \mathcal{D}_G)$ contains all global databases that can be obtained by adding to $\overline{\mathcal{B}}$ (among others) at least one fact of the form $team(\mathsf{d}, \alpha)$ and one fact of the form $team(\mathsf{f}, \beta)$, where α and β are constants of the domain $\mathcal{U}$. Notice that, since $\emptyset \in sem_c(\mathcal{I}_G, \mathcal{D}_G)$, $ans_c(q, \mathcal{RS}, \mathcal{D}) = \emptyset$, i.e., there is no answer to the query in the complete semantics, whereas $ans_s(q, \mathcal{I}_G, \mathcal{D}_G) = \{\mathsf{b}, \mathsf{d}, \mathsf{f}\}$.

LAV mapping. Let us now construct a LAV data integration system $\mathcal{I}_L = \langle \mathcal{G}_0, \mathcal{S}_L, \mathcal{M}_L\rangle$, where $\mathcal{G}_0$ is the same as the GAV system $\mathcal{I}_G$, $\mathcal{S}_L$ contains the binary relation s_3 and s_4 and the mapping $\mathcal{M}_L$ is as follows:

$$\begin{aligned} \langle s_4, \quad & s_4(X, Y) \leftarrow player(X, Y)\rangle \\ \langle s_3, \quad & s_3(X, Y) \leftarrow team(X, Y)\rangle. \end{aligned}$$

Consider the source database $\mathcal{D}_L = \{s_3(\mathsf{b}, \mathsf{c}), s_4(\mathsf{a}, \mathsf{b}), s_4(\mathsf{e}, \mathsf{f}), s_4(\mathsf{a}, \mathsf{d}), \}$. It is easy to see that $sem_e(\mathcal{I}_L, \mathcal{D}_L) = sem_e(\mathcal{I}_G, \mathcal{D}_G) = \emptyset$, $sem_s(\mathcal{I}_L, \mathcal{D}_L) = sem_s(\mathcal{I}_G, \mathcal{D}_G)$, and $sem_c(\mathcal{I}_L, \mathcal{D}_L) =$

[1] For the sake of clarity, in the examples we use names to denote attributes, rather than integers.

$sem_c(\mathcal{I}_G, \mathcal{D}_G)$. Obviously, the answers to the query q, under the sound, complete, and exact semantics are the same of the GAV case.

■

Commonly, the problem of query answering in data integration has been addressed by means of *query rewriting* techniques. In query rewriting the computation is separated in two steps: the first one exploits the mapping $\mathcal{M}$ to reformulate the query q into another query q_r, the *rewriting*, that can be evaluated on the source database $\mathcal{D}$, and the second one evaluates q_r on $\mathcal{D}$. Conversely, in generic query answering we do not pose any limit to the method adopted to compute the answers to the query and exploit also the data at the sources to this aim [36].

We say that q_r is a *perfect rewriting* of q w.r.t. $\mathcal{I}$ and an assumption $as(\mathcal{M})$ on $\mathcal{M}$ if $q_r^{\mathcal{D}} = ans_{as(\mathcal{M})}(q, \mathcal{I}, \mathcal{D})$ for each $\mathcal{D}$.

3.3 Semantics for Inconsistent Data Sources

Since sources are in general autonomous and heterogeneous, data retrieved from them are likely not to satisfy the constraints expressed on the global schema $\mathcal{G}$. According to the semantics above defined, the situation may arise in which data retrieved from the sources cannot be reconciled in the global schema in such a way that both the constraints expressed on $\mathcal{G}$ and the assumption on the mapping are satisfied, thus the integration system should be regarded as *inconsistent*. In particular, the simple violation of a single dependency (under the sound and exact semantics) may lead to the non-interesting case in which $sem_e(\mathcal{I}, \mathcal{D}) = \emptyset$ (see Example 3.2.1), and $sem_s(\mathcal{I}, \mathcal{D}) = \emptyset$, for both the GAV and the LAV mapping. On the other hand, since most of the data could satisfy global constraints, it seems unreasonable to consider the entire system inconsistent and unable to provide significant answers to queries. If we do not want to conclude in these cases that no global database exists in the semantics of a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ with respect to a source database $\mathcal{D}$ and an assumption $as(\mathcal{M})$, we need a different characterization of the mapping.

Example 3.2.1 (contd.) Assume now to specify the key dependency $key(player) = \{Pname\}$ on $\mathcal{G}_0$, stating that a player cannot be enrolled in more than one team. For both the GAV and the LAV approach (hence we adopt now the same symbol $\mathcal{I}_0$ for both $\mathcal{I}_G$ and $\mathcal{I}_L$, and $\mathcal{D}_0$ for both $\mathcal{D}_G$ and $\mathcal{D}_L$), it is now also $sem_s(\mathcal{I}_0, \mathcal{D}_0) = \emptyset$, other than $sem_e(\mathcal{I}_0, \mathcal{D}_0) = \emptyset$. Indeed, every database $\mathcal{B}$ in $sem_s(\mathcal{I}_0, \mathcal{D}_0)$ should contain the facts $player(\mathsf{a}, \mathsf{b})$ and $player(\mathsf{a}, \mathsf{d})$ that are not consistent with the key dependency, and it is not possible to make $\mathcal{B}$ satisfy this constraint by adding other facts to $\mathcal{B}$. On the other hand, each such $\mathcal{B}$ should contain also $team(\mathsf{b}, \mathsf{c})$ and $player(\mathsf{e}, \mathsf{f})$, where $team(\mathsf{b}, \mathsf{c})$ is consistent with the dependencies in the schema, whereas the inconsistency caused by $player(\mathsf{e}, \mathsf{f})$ can be resolved under the sound semantics by adding a suitable fact to $\mathcal{B}$ of the form $team(\mathsf{f}, \alpha)$, where α is a constant of $\mathcal{U}$. Therefore, rather than the whole domain $\mathcal{U}$, the query $q(x) \leftarrow team(x, y)$ should return the answer set $\{\mathsf{b}, \mathsf{f}\}$ under the sound semantics. For the exact assumption, we can show in an analogous way that, taking into account those tuples that can be retrieved from the sources and that do not violate the constraints, the answer set to our query might be $\{\mathsf{b}\}$, for both the LAV and the GAV system. ■

A possible solution to this problem is to characterize the semantics of $\mathcal{I}$ in terms of those databases that

- satisfy the integrity constraints on $\mathcal{G}$, and
- approximate "at best" the satisfaction of the assumptions on $\mathcal{M}$, i.e., are *as close as possible* to the semantics interpretation of $\mathcal{M}$.

In other words, the integrity constraints of $\mathcal{G}$ are considered "hard", whereas the assumptions on $\mathcal{M}$ are considered "soft", and we resort to satisfy the mapping at best when we are not able to satisfy it in rigorous way.

In the spirit of a common approach in the literature on inconsistent databases [71, 123, 7], we now propose a modified definition of the semantics that reflects the above idea. Given a source database $\mathcal{D}$ for $\mathcal{I}$ and an assumption $as(\mathcal{M})$ on the mapping $\mathcal{M}$, we define an ordering on the set of all global databases for $\mathcal{I}$ that are consistent with $\mathcal{G}$. If $\mathcal{B}_1$ and $\mathcal{B}_2$ are two such databases, we define $\mathcal{B}_1 \leq_{\mathcal{D}} \mathcal{B}_2$ as follows:

- if $\mathcal{M}$ is GAV, for each assertion $\langle r_{\mathcal{G}}, q_{\mathcal{S}} \rangle \in \mathcal{M}$
  - (a) $r_{\mathcal{G}}^{\mathcal{B}_1} \cap q_{\mathcal{S}}^{\mathcal{D}} \supseteq r_{\mathcal{G}}^{\mathcal{B}_2} \cap q_{\mathcal{S}}^{\mathcal{D}}$ if $as(\mathcal{M}) = s$;
  - (b) $r_{\mathcal{G}}^{\mathcal{B}_1} - q_{\mathcal{S}}^{\mathcal{D}} \subseteq r_{\mathcal{G}}^{\mathcal{B}_2} - q_{\mathcal{S}}^{\mathcal{D}}$ if $as(\mathcal{M}) = c$;
  - (c) $r_{\mathcal{G}}^{\mathcal{B}_1} \cap q_{\mathcal{S}}^{\mathcal{D}} \supseteq r_{\mathcal{G}}^{\mathcal{B}_2} \cap q_{\mathcal{S}}^{\mathcal{D}}$ and $r_{\mathcal{G}}^{\mathcal{B}_1} - q_{\mathcal{S}}^{\mathcal{D}} \subseteq r_{\mathcal{G}}^{\mathcal{B}_2} - q_{\mathcal{S}}^{\mathcal{D}}$ if $as(\mathcal{M}) = e$;
- if $\mathcal{M}$ is LAV, for each assertion $\langle r_{\mathcal{S}}, q_{\mathcal{G}} \rangle \in \mathcal{M}$
  - (a) $q_{\mathcal{G}}^{\mathcal{B}_1} \cap r_{\mathcal{S}}^{\mathcal{D}} \supseteq q_{\mathcal{G}}^{\mathcal{B}_2} \cap r_{\mathcal{S}}^{\mathcal{D}}$ if $as(\mathcal{M}) = s$;
  - (b) $q_{\mathcal{G}}^{\mathcal{B}_1} - r_{\mathcal{S}}^{\mathcal{D}} \subseteq q_{\mathcal{G}}^{\mathcal{B}_2} - r_{\mathcal{S}}^{\mathcal{D}}$ if $as(\mathcal{M}) = c$;
  - (c) $q_{\mathcal{G}}^{\mathcal{B}_1} \cap r_{\mathcal{S}}^{\mathcal{D}} \supseteq q_{\mathcal{G}}^{\mathcal{B}_2} \cap r_{\mathcal{S}}^{\mathcal{D}}$ and $q_{\mathcal{G}}^{\mathcal{B}_1} - r_{\mathcal{S}}^{\mathcal{D}} \subseteq q_{\mathcal{G}}^{\mathcal{B}_2} - r_{\mathcal{S}}^{\mathcal{D}}$ if $as(\mathcal{M}) = e$;

Furthermore, $\mathcal{B}_1 <_{\mathcal{D}} \mathcal{B}_2$ stands for $\mathcal{B}_1 \leq_{\mathcal{D}} \mathcal{B}_2 \wedge \mathcal{B}_2 \not\leq_{\mathcal{D}} \mathcal{B}_1$.

With this notion in place, we can modify the definition of semantics of a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ with respect to a source database $\mathcal{D}$ for $\mathcal{I}$ and an assumption $as(\mathcal{M})$ on $\mathcal{M}$. In order to distinguish between the semantics of Section 3.2 and their modified version, in the following we refer to the former as *strict* semantics, while we call the latter *loose* semantics, and denote them with $sem_{l\,as(\mathcal{M})}(\mathcal{I}, \mathcal{D})$ (where, as usual, $as(\mathcal{M}) \in \{s, c, e\}$). Namely, we call *strictly-sound*, *strictly-complete*, and *strictly-exact* the sound, complete, and exact semantics, whereas we respectively call *loosely-sound*, *loosely-complete*, and *loosely-exact* the three corresponding loose semantics. Formally, a global database $\mathcal{B}$ for $\mathcal{I}$ is in $sem_{l\,as(\mathcal{M})}(\mathcal{I}, \mathcal{D})$, if

(i) $\mathcal{B}$ is consistent with $\mathcal{G}$;

(ii) $\mathcal{B}$ is minimal with respect to $\leq_{\mathcal{D}}$, i.e., for no other global database $\mathcal{B}'$ for $\mathcal{I}$ consistent with $\mathcal{G}$, we have that $\mathcal{B}' <_{\mathcal{D}} \mathcal{B}$.

The notion of answers to a user query q coincides with the one given for the strict semantics, and we indicate the set of answers to q under the loose semantics with $ans_{l\,as(\mathcal{M})}(q, \mathcal{I}, \mathcal{D})$. It is immediate to verify that, if $sem_{as(\mathcal{M})}(\mathcal{I}, \mathcal{D}) \neq \emptyset$ for any $as(\mathcal{M}) \in \{s, c, e\}$, then the strict semantics and the loose one coincide, in the sense that, for each query q, $ans_{as(\mathcal{M})}(q, \mathcal{I}, \mathcal{D}) = ans_{l\,as(\mathcal{M})}(q, \mathcal{I}, \mathcal{D})$. Consequently, since (as illustrated in Section 3.2) $sem_c(\mathcal{I}, \mathcal{D}) \neq \emptyset$ for each $\mathcal{I}$ and for each $\mathcal{D}$, it follows that the strictly-complete and the loosely-complete semantics always coincide.

Moreover, notice that the loose semantics is never empty, i.e., it always holds that $sem_{l\,as(\mathcal{M})}(\mathcal{I}, \mathcal{D}) \neq \emptyset$ for any $as(\mathcal{M}) \in \{s, c, e\}$, even if $sem_{as(\mathcal{M})}(\mathcal{I}, \mathcal{D}) = \emptyset$.

Example 3.2.1 (contd.) With regard to our ongoing example, for both the LAV and GAV system we have that:

1. $sem_{le}(\mathcal{I}_0, \mathcal{D}_0)$ contains the database $\mathcal{B}_1 = \{player(\mathsf{a}, \mathsf{b}),\ team(\mathsf{b}, \mathsf{c})\}$, and all the databases of the form $\mathcal{B}_2 = \{player(\mathsf{a}, \mathsf{d}),\ team(\mathsf{b}, \mathsf{c}),\ team(\mathsf{d}, \alpha)\}$ for each $\alpha \in \mathcal{U}$, $\mathcal{B}_3 = \{player(\mathsf{a}, \mathsf{b}),\ player(\mathsf{e}, \mathsf{f}),\ team(\mathsf{b}, \mathsf{c}),\ team(\mathsf{f}, \alpha)\}$ for each $\alpha \in \mathcal{U}$, and $\mathcal{B}_4 = \{player(\mathsf{a}, \mathsf{d}),\ player(\mathsf{e}, \mathsf{f}),\ team(\mathsf{b}, \mathsf{c}),\ team(\mathsf{d}, \alpha),\ team(\mathsf{f}, \beta)\}$ for each $\alpha, \beta \in \mathcal{U}$;

2. $sem_{ls}(\mathcal{I}_0, \mathcal{D}_0)$ contains the databases of the form $\mathcal{B}_3$ and $\mathcal{B}_4$, and each global database for $\mathcal{I}_0$ consistent with $\mathcal{G}_0$ that can be obtained by consistently adding facts to a database of the form $\mathcal{B}_3$ or $\mathcal{B}_4$;

3. $sem_{lc}(\mathcal{I}_0, \mathcal{D}_0) = sem_c(\mathcal{I}_0, \mathcal{D}_0)$.

Therefore, under the three semantics, the answers to the query $q(x) \leftarrow team(x, y)$ are respectively $ans_{le}(q, \mathcal{I}_0, \mathcal{D}_0) = \{\mathsf{b}\}$, $ans_{ls}(q, \mathcal{I}_0, \mathcal{D}_0) = \{\mathsf{b}, \mathsf{f}\}$ and $ans_{lc}(q, \mathcal{I}_0, \mathcal{D}_0) = \emptyset$. ∎

3.4 Related Work

Our loose semantics follows the principles that underlie the notion of *database repair* introduced by previous works in the area of inconsistent databases that have proposed several semantics relying either on set-containment or cardinality-based ordering [71, 57, 17, 123, 7, 8]. However, such studies basically apply to a single database setting [71, 17, 7, 8], and to GAV integration systems under the exact assumption on the mapping [123, 57, 86]. Conversely, our approach considers a pure data integration framework, and our semantics seriously takes into account the interpretation of the mapping. Nonetheless, with this consideration in place, we can say that our loosely-sound semantics is similar in the spirit to the semantics described in [71], whereas the loosely-exact semantics has been thoroughly studied in [86, 7, 8][2].

More recently, data inconsistency in a LAV data integration setting has been studied in [13] and [16]. The semantics proposed in [13] and [16] turns out to be different from the loose semantics for LAV of Section 3.3. Indeed, whereas our proposal focuses on the mapping and define a suitable relaxation of it in the presence of inconsistency, [13, 16] characterize the semantics in

[2] A more detailed description of the works on inconsistent databases will be given at the end of Chapter 5.

terms of the repairs of the different global databases that can be obtained populating the global schema according to the LAV mapping. More specifically, [13, 16] assume that the sources are sound, and consider the set $min(\mathcal{G})$ of the minimal (w.r.t. set inclusion) global databases that satisfy the mapping. Then, the repairs are the global databases consistent with the constraints that are minimal w.r.t. $\leq_{\mathcal{DB}}$ for some $\mathcal{DB} \in min(\mathcal{G})$, where $\mathcal{B} \leq_{\mathcal{DB}} \mathcal{B}'$ if $\triangle(\mathcal{B}, \mathcal{DB}) \subseteq \triangle(\mathcal{B}', \mathcal{DB})$, where in turn $\triangle(X, Y)$ indicates the symmetric difference between X and Y.

We point out that, in this semantics, even if the sources are assumed to be sound, the repairs are computed as if the retrieved data were exact. Under such assumption, the repair semantics can be different from the sound semantics even when the latter is not empty. In other words, one resorts to repair the system even when the system actually does not need to be repaired. This is shown in the following example.

Example 3.4.1 Consider a simple LAV system $\mathcal{I}$ in which there are two global relations $g_1/2$ and $g/2$, an inclusion dependency $g_1[1,2] \subseteq g_2[1,2]$, a source relation $s/2$, and a mapping assertion $s(x, y) \leftarrow g_1(x, y)$. Then, let $\mathcal{D} = \{s(\mathsf{a}, \mathsf{b})\}$ be a source database for $\mathcal{I}$. In this case we have $min(\mathcal{G}) = \{g_1(\mathsf{a}, \mathsf{b})\}$. This unique minimal global database is inconsistent w.r.t. the ID expressed on the global schema. According to [13, 16], two repairs have to be considered in this situation $\mathcal{R} = \emptyset$ and $\mathcal{R}' = \{g_1(\mathsf{a}, \mathsf{b}), g_2(\mathsf{a}, \mathsf{b})\}$. Hence, every query over the global schema has an empty consistent answer. ∎

Notice that, in our framework, when we assume the sources to be sound, we are able to solve the inconsistency in the above example without actually relaxing the mapping, since in this case $sem_s(\mathcal{I}, \mathcal{D}) = sem_{ls}(\mathcal{I}, \mathcal{D})$. Furthermore, given a simple query $q(X, Y) \leftarrow g_1(X, Y)$ we have $ans_s(q, \mathcal{I}, \mathcal{D}) = \{\langle \mathsf{a}, \mathsf{b} \rangle\}$, rather than the empty set. Other details on the techniques proposed in [13, 16] will be given at the end of Chapter 6.

We point out that, in all mentioned works, an in-depth investigation of the role of inclusion dependencies expressed on the database schema is actually missing, and the methods proposed to get consistent answers from inconsistent databases basically apply only to the class of "universally quantified constraints". Two exceptions are [86], where a preliminary study on existentially quantified inclusion dependencies is presented, and [16], where, however, the problem is tackled by taking into account only a particular class of finite global databases, viz. the global repairs that can be computed by using only constants appearing in the source database.

Conversely, in the following chapters we will provide a thoroughly study of query answering under the different semantics, when inclusion, key and exclusion dependencies are expressed on the global schema.

Chapter 4

State of the Art of Query Processing in Data Integration Systems

The problem of query processing is to find efficient methods for answering queries posed to the global (virtual) schema of a data integration system on the basis of the data stored at sources.

In this chapter, we examine some algorithms and techniques proposed in the literature to solve this problem in both the LAV and the GAV frameworks.

As already noticed in the introduction, whereas query processing in the LAV approach has been always regarded as a difficult task, this problem has commonly been considered much easier in the GAV approach. Indeed, query processing in LAV has been traditionally seen as a form of reasoning in the presence of incomplete information, whereas in GAV it has been assumed that answering a query means unfolding its atoms according to their definitions on the sources. As shown in Section 1.4, this means assuming that in GAV only a single database satisfies the global schema, i.e., the database obtained by evaluating the views in the mapping over the source extensions. Only recently, more complex GAV frameworks have been studied, in which the semantics of the system is given in terms of a set of databases rather than a single one, and more complex techniques than simple unfolding have been proposed.

We survey here the most important query answering algorithms proposed in the literature for LAV, and we describe the principal GAV data integration systems and the form of query processing they adopt. Furthermore, we review recent studies showing that query processing in GAV is harder than simple unfolding.

4.1 Query Processing in LAV

Since in LAV the mapping between the sources and the global schema is described as a set of views over the global schema, query processing amounts to finding a way to answer a query posed over a database schema using a set of views over the same schema. This problem, called

answering queries using views, is widely studied in the literature, since it has applications in many areas. In query optimization [47], the problem is relevant because using materialized views may speed up query processing. A data warehouse can be seen as a set of materialized views, and, therefore, query processing reduces to query answering using views [94, 148, 157, 91]. In the context of database design, using views provides means for maintaining the physical perspective of the data independent from its logical perspective [149]. It is also relevant for query processing in distributed databases [102] and in federated databases [117].

The most common approach proposed in the literature to deal with the problem of query answering using views is by means of *query rewriting*. In query rewriting, a query and a set of view definitions over a database schema are provided, and the goal is to reformulate the query into an expression, the *rewriting*, whose evaluation on the view extensions supplies the answer to the query. Consider, for instance a data integration system $\mathcal{I}$ in which both the user query q and its rewriting q_r are Datalog queries. In such a case, the EDB predicates of q are relations of the global schema, while the EDB predicates of q_r are relations of the source schema. Hence, q_r can be evaluated directly over a source database for $\mathcal{I}$, i.e., an extension for the views in the mapping. Notice that, the user query language $\mathcal{L}_\mathcal{Q}$ and the rewriting language $\mathcal{L}'_\mathcal{Q}$ may be different.

In conclusion, query answering via query rewriting is divided in two steps, where the first one consists of reformulating the query in terms of the given query language $\mathcal{L}'_\mathcal{Q}$ over the alphabet of the views (possibly augmented with auxiliary predicates), and the second one evaluates the rewriting over the view extensions.

In general, given a data integration system $\mathcal{I}$, a source database $\mathcal{D}$ for $\mathcal{I}$, and a user query q, we are interested in computing the set of certain answers to q (see Section 3.2). In the query rewriting approach, we do not take into account the source extensions in the reformulation step, so that, the capability of the rewriting to retrieve the set of certain answers depends on the query language $\mathcal{L}'_\mathcal{Q}$ in which it is expressed. Such a situation can constitute a limitation on computing the set of certain answers for q.

A more general approach, simply called query answering using views [1, 85, 36, 38, 33], is that to consider, besides the query and the views, also the extensions of the views, i.e., the source database. Here, no limits are posed to query processing, and the only goal is to compute the set of certain answers to the query by exploiting all possible information, in particular the view extensions.

According to [39], many of the papers concerning query processing in LAV do not distinguish between query answering and query rewriting using views, and give rise to a sort of confusion between the two notions. Part of the problem comes from the fact that when the query and the views are conjunctive queries, there are algorithms (like the bucket or the inverse rules algorithm that are described in the following) for computing the best possible rewriting as union of conjunctive queries, where the best possible rewriting is a rewriting that actually is able to provide the certain answers to the user query. Therefore, in these cases the best possible rewriting is basically expressible in the same language as the original query and views. However for other query languages this does not happen, and state of the art query rewriting techniques are not able to solve the query answering problem. So, in spite of the large amount of work on the subject, the relationship between query rewriting and query answering using views is not completely clarified

yet.

In order to shed light on this problem, [39, 40] define a *rewriting* of a query q with respect to a set $\mathcal{V}$ of views as a function that, given the extension $\mathcal{D}$ of the views, returns a set of tuples that are contained in the certain answers of q w.r.t. $\mathcal{V}$ and $\mathcal{D}$. The rewriting that returns precisely the set of certain answers for each source database is called the *perfect* rewriting of the query w.r.t. the views. Actually, the above definition coincides with the definition of perfect rewriting given in Section 3.2, when applied to the LAV framework.

In the following we survey several papers that deal with query processing in LAV, and distinguish between those concerning query rewriting and those referring to generic query answering.

Notice that, with respect to our framework described in Chapter 3, these algorithms assume that the views in the mapping are all sound.

4.1.1 Query Rewriting

First of all we recall the definition of rewriting traditionally adopted in the literature [111, 150, 92].

To be able to compare different reformulations of queries, we first introduce the notion of containment between queries. Given two queries q_1 and q_2 over a database schema $\mathcal{RS}$, we say that q_1 is *contained* in q_2 if for all databases $\mathcal{DB}$ for $\mathcal{RS}$ we have that $q_1^{\mathcal{DB}} \subseteq q_2^{\mathcal{DB}}$. We say that q_1 and q_2 are *equivalent* if q_1 is contained in q_2 and q_2 is contained q_1.

The problem of query containment has been studied in various settings. In [46], NP-completeness has been established for conjunctive queries, and in [50] a multi-parameter analysis has been performed for the same case, showing that the intractability is due to certain types of cycles in the queries. In [104, 151], Π_2^p-completeness of containment of conjunctive queries with inequalities was proved, and in [145] the case of queries with the union and difference operators was studied. For various classes of Datalog queries with inequalities, decidability and undecidability results were presented in [48] and [151], respectively. Query containment under constraints has also been the subject of several investigations. For example, decidability of conjunctive query containment was investigated in [5] under functional and multi-valued dependencies, in [101] under functional and inclusion dependencies, in [45, 116, 118] under constraints representing *is-a* hierarchies and complex objects, and in [56] in the case of constraints represented as Datalog programs. For queries over semistructured data, query containment for conjunctive regular path queries, in which the atoms in the body are regular expressions over binary predicates, has been studied in [32, 76], and EXPSPACE completeness has been established in [37].

To formally define the notion of rewriting, we consider a relational data integration setting and assume that queries and views defining the source relations, i.e., views in the mapping, are unions of conjunctive queries over the global schema.

Given a data integration system $\mathcal{I}$, where the mapping is given in terms of the set of views $\mathcal{V} = \{V_1, \ldots, V_m\}$ over the global schema $\mathcal{G}$, and given a user query q over $\mathcal{I}$, the query q_r is a *rewriting* of q using $\mathcal{V}$ if

- $q_r \cup \mathcal{V}$ is contained in q, and
- q_r does not refer to the relations in $\mathcal{G}$, i.e., the relations of the global schema appear only in the view definitions and not in the bodies of the clauses of q_r.

In general, the set of available sources may not store all the data needed to answer a user query, and therefore the goal is to find a query expression that provides all the answers that can be obtained from the views. So, whereas in different contexts, i.e., query optimization or maintaining physical data independence, the focus is on finding rewritings that are logically equivalent to the original query, in data integration systems the interest has been mainly devoted to *maximally contained* rewritings [92], formally defined as follows.

Given a query q over $\mathcal{I}$, a set of views $\mathcal{V} = \{V_1, \ldots, V_m\}$ over $\mathcal{G}$, and a query language $\mathcal{L}_\mathcal{Q}$, the query q_m is a *maximally contained rewriting* of q using $\mathcal{V}$ w.r.t. $\mathcal{L}_\mathcal{Q}$ if

- q_m is a rewriting of q using $\mathcal{V}$,
- there is no rewriting $q' \neq q_m$ in $\mathcal{L}_\mathcal{Q}$ such that $q_m \cup \mathcal{V}$ is contained in $q' \cup \mathcal{V}$.

Notice that the maximally contained rewriting is defined in terms of a specified query language $\mathcal{L}_\mathcal{Q}$. As already said, dealing with a fixed language might not lead us to obtain the best possible rewriting for a user query q. Hence, in spite of its original aim, the evaluation over the source extensions of the maximally contained rewriting does not in general provide the set of certain answers to the query. However, in the practical (and widely studied) case in which queries are union of conjunctive queries and views are conjunctive queries the two notion coincide, and hence computing the maximally contained rewriting means actually solve the query answering problem.

An important theoretical result concerning the problem of query rewriting, in the case that views and queries are conjunctive queries, is presented in [112]. In that paper the authors show that, when a query a q is a union of conjunctive queries, if an equivalent conjunctive rewriting of q exists, then such a rewriting has at most as many atoms as q. Such a result leads immediately to nondeterministic polynomial-time algorithms to find either equivalent conjunctive rewritings or maximally contained rewritings that are union of conjunctive queries [150]. In both cases it is sufficient to consider each possible conjunction of views that produces a *candidate rewriting* whose size is less or equal to the size of the original query, and then check the correctness of the rewriting. Note that the number of candidate rewritings is exponential in the size of the query.

Bucket algorithm and its extensions In order to compute all the rewritings that are contained in (and not necessarily equivalent to) the original query, the *bucket algorithm*, presented in [114], improves the technique described above since it exploits a suitable heuristic for pruning the space of candidate rewritings. The algorithm was proposed in the context of the Information Manifold (IM) system [117], a project developed at AT&T. IM handles the presence of inclusion and functional dependencies over the global schema and limitations in accessing the sources, and uses conjunctive queries as the language for describing the sources and querying the system.

To compute the rewriting of a query q, the bucket algorithm proceeds in two steps:

1. for each atom g in q, create a bucket that contains the views from which tuples of g can be retrieved, i.e., the views whose definition contains an atom to which g can be mapped in a rewriting of the query;
2. consider as candidate rewriting each conjunctive query obtained by combining one view

from each bucket, and check by means of a containment algorithm whether such a query is contained in q. If so, the candidate rewriting is added to the answer.

If the candidate rewriting is not contained in q, before discarding it, the algorithm checks if it can be modified by adding comparison predicates in such a way that it is contained in q. The proof that the bucket algorithm generates the maximal contained rewriting when the query language is union of conjunctive queries, is given in [85].

Note that, on the basis of the results in [112], the algorithm considers only rewritings that have at most the same number of atoms as the original query. As shown in the same paper and in [143], the given bound on the size of the rewriting does not hold in the presence of arithmetic comparison predicates, functional dependencies over the global schema, or limitations in accessing the sources. In such cases we have to consider rewritings that are longer than the original query. More precisely, let p be the number of atoms in the query q:

- In the presence of functional dependencies, a minimal rewriting has at most $p + d$ atoms, where d is the sum of the arities of the atoms in q;
- In the presence of limitations in accessing the sources, a minimal rewriting has at most $p + m$ atoms, where m is the number of different variables in q;
- In the presence of comparison predicates, the size of a minimal rewriting is at most exponential in the size of q.

According to [113] the bucket algorithm, in practice, does not miss solutions because of the length of the rewritings it considers, but other results [61, 89, 88] demonstrate that in the presence of functional dependencies and limitations in accessing the sources, union of conjunctive queries does not suffice to obtain the maximal contained rewritings, and one needs to resort to recursive rewritings. We refer the interested reader to [61, 106] for a more detailed treatment of the problem.

Two improved versions of the bucket algorithm are the *MiniCon algorithm* [140] and the *shared-variable bucket algorithm* [129]. In both the algorithms the basic idea is to examine the interaction among variables of the original query and variables in the views, in order to reduce the number of views inserted in the buckets, and hence the number of candidate rewritings to be considered. Experimental results related to the performance of MiniCon show that this algorithm scales very well and outperform the bucket algorithm.

Inverse rules algorithm The previous algorithms search the space of possible candidate rewritings by using buckets and then checks whether each computed rewriting is contained in the original query. In the following, we describe a different algorithm, namely the *inverse rules algorithm* [61], that generates a rewriting (query plan) in time that is polynomial in the size of the query. The algorithm was developed in the context of the Infomaster system [59], an information integration tool developed at Stanford University . The inverse rules algorithm constructs a set of rules that invert the view definitions and provides the system with an inverse mapping which establish how to obtain the data of the global concepts from the data of the sources. The basic idea is to replace existential variables in the body of each view definition by a Skolem function.

Example 4.1.1 Given a view definition

$$v(X) \leftarrow a_1(X,Y), a_2(X,Y)$$

the set of inverse rules obtained from it is

$$\begin{aligned} a_1(X, f(X)) &\leftarrow v(X) \\ a_2(X, f(X)) &\leftarrow v(X) \end{aligned}$$

■

Given a non-recursive Datalog query q and a set of view definitions $\mathcal{V}$, the rewriting is the Datalog program consisting of both the query and the inverse rules obtained from $\mathcal{V}$.

Note that in the skolemization phase, we just introduce function symbols in the head of the inverse rules and never introduce a symbol within another, which leads to a finite evaluation process. Since bottom-up evaluation of the rewriting can produce tuples with function symbols, these need to be discarded. A polynomial time procedure to eliminate these function symbols by adding new predicates is described in [60]. It is also shown that the inverse rules algorithm returns a maximally contained rewriting w.r.t. union of conjunctive queries, in time that is polynomial in the size of the query.Even if the computational cost of constructing the query plan is polynomial, the obtained rewriting contains rules which may cause accessing views that are irrelevant for the query. In [110] it is shown that the problem of eliminating irrelevant rules has exponential time complexity. The inverse rules algorithm can handle also recursive Datalog queries, the presence of functional dependencies over the global schema, or the presence of limitations in accessing the sources, by extending the obtained query plan with other specific rules.

Other algorithms and results Finally, we cite the *unification-join algorithm* [141], an exponential-time query answering algorithm based on a skolemization phase and on the representation of conjunctive queries by means of hypergraphs. In [141] is also described a polynomial time version of the algorithm developed for the case in which queries are acyclic conjunctive queries. The unification join algorithm is extended in [90] in order to deal with inclusion or functional dependencies expressed over the global schema.

Other studies are concerned with the problem of query rewriting using views, under the different assumptions that queries are recursive queries [4], description logics queries [10, 34], queries for semi-structured data [35, 139], queries with aggregates [87, 53], or in presence of limitations in accessing the views [143, 106, 115, 77].

4.1.2 Query Answering

The complexity of answering queries using views for different languages (both for the query and for the views) is studied in [1]. The authors deal with the two assumptions that all the views are sound (called *open world assumption* in the paper), or that all the views are exact (called *closed world assumption* in the paper). Under open world assumption they show that in the case where the views are conjunctive queries, the problem becomes coNP-hard already in the case

Sound	CQ	CQ$^{\neq}$	PQ	Datalog	FOL
CQ	*PTIME*	*coNP*	*PTIME*	*PTIME*	*undec.*
CQ$^{\neq}$	*PTIME*	*coNP*	*PTIME*	*PTIME*	*undec.*
PQ	*coNP*	*coNP*	*coNP*	*coNP*	*undec.*
Datalog	*coNP*	*undec.*	*coNP*	*undec.*	*undec.*
FOL	*undec.*	*undec.*	*undec.*	*undec.*	*undec.*
Exact	CQ	CQ$^{\neq}$	PQ	Datalog	FOL
CQ	*coNP*	*coNP*	*coNP*	*coNP*	*undec.*
CQ$^{\neq}$	*coNP*	*coNP*	*coNP*	*coNP*	*undec.*
PQ	*coNP*	*coNP*	*coNP*	*coNP*	*undec.*
Datalog	*undec.*	*undec.*	*undec.*	*undec.*	*undec.*
FOL	*undec.*	*undec.*	*undec.*	*undec.*	*undec.*

Table 4.1: Complexity of query answering using views

where the query is a conjunctive query with inequality. Instead, assuming queries be conjunctive queries, the problem is already coNP-hard when the views are positive queries (PQ). As shown in [39, 40], the problem is already coNP-hard for very simple query languages containing union. Under the different assumption that the views are exact (closed world assumption) it is shown in [1] that the problem is coNP already in the case that views and queries are conjunctive queries. Finally, it is sketched an effective way to compute the certain answers by representing the global database by means of conditional tables and querying them using the techniques for databases with incomplete information [98]. Table 4.1 summarizes the results presented in [1].

In [36], the problem is studied for a setting where the global schema models a semistructured database, i.e., a labeled directed graphs. It follows that both the user queries, and the queries used in the LAV mapping should be expressed in a query language for semistructured data. The main difficulty arising in this context is that languages for querying semistructured data enable expressing regular-path queries [3, 19, 74]. A regular-path query asks for all pairs of nodes in the database connected by a path conforming to a regular expression, and therefore may contain a restricted form of recursion. Note that, when the query contains unrestricted recursion, both query rewriting and query answering using views become undecidable, even when the views are not recursive [60]. Table 4.2 summarizes the results presented in [36][1].

While regular-path queries represent the core of any query language for semistructured data, their expressive power is limited. Several authors point out that extensions are required for making them useful in real settings (see for example [18, 19, 127]). Indeed, the results in [36] have been extended to query language with the inverse operator in [38], and to the class of union of conjunctive regular-path queries in [41].

[1] In the table, also expression complexity (complexity with respect to the size of the query and the view definitions) is taken into account.

domain	views	Complexity		
		data	expression	combined
closed	all sound	*coNP*	*coNP*	*coNP*
	all exact	*coNP*	*coNP*	*coNP*
	arbitrary	*coNP*	*coNP*	*coNP*
open	all sound	*coNP*	*PSPACE*	*PSPACE*
	all exact	*coNP*	*PSPACE*	*PSPACE*
	arbitrary	*coNP*	*PSPACE*	*PSPACE*

Table 4.2: Complexity of query answering for regular-path queries

4.2 Query Processing in GAV

As already said, in GAV integration systems it is in general assumed that, to answer a query q posed over the global schema, it is sufficient to unfold each atom of q with the corresponding view over the sources. Even if the process of unfolding is a simple mechanism, defining the views associated to the global elements implies precisely understanding the relationships among the sources, that is in general a non-trivial task. We can say that, in the GAV approach, query processing by unfolding is realized at design time, and the main issue turns out to be the definition of *wrappers* and *mediators* [155]. As already mentioned, wrappers are software components that present data at the sources in a suitable form adopted in the integration system, and handle the access to the data taking into account source capabilities to answer queries posed to the sources [121, 158]. Mediators are modules that establish how to access and merge data residing at the sources, i.e., synthesize the view definitions associated to the global relations.

In the first GAV integration systems proposed in the literature the notion of global schema was actually missing, in the sense that the global schema was simply the collection of relations exported by the mediators and no rules or integrity constraints were actually expressed over the global concepts. In these systems, mediators simply realize an explicit layer between the user application and the data sources. Furthermore, even when a global schema was provided, the assumption that the views over source schemas were exact was implicitly considered (closed world assumption). In such a situation exactly one global legal database exists, i.e., the retrieved global database that is obtained by populating the global schema according to the view definitions, and evaluating the query over this unique database is equivalent to evaluating its unfolding over the sources.

In the rest of this section, we first describe some integration systems that follow the GAV approach in the simplified setting we have described above, then we turn our attention to more recent approaches that deal with GAV integration in more complex scenarios.

4.2.1 Methods and Systems Based on Unfolding

We now describe the main integration systems following the GAV approach that are essentially based on query unfolding.

The TSIMMIS Project TSIMMIS (The Stanford-IBM Manager of Multiple Information Sources) is a joint project of the Stanford University and the Almaden IBM database research group [49]. It is based on an architecture that presents a hierarchy of wrappers and mediators, in which wrappers convert data from each source into a common data model called OEM (Object Exchange Model) and mediators combine and integrate data exported by wrappers or by other mediators. Hence, the global schema is essentially constituted by the set of OEM objects exported by wrappers and mediators. Mediators are defined in terms of a logical language called MSL (Mediator Specification Language), which is essentially Datalog extended to support OEM objects. OEM is a semistructured and self-describing data model, in which each object has an associated label, a type for the value of the object and a value (or a set of values). User queries are posed in terms of objects synthesized at a mediator or directly exported by a wrapper. They are expressed in MSL or in a specific query language called LOREL (Lightweight Object REpository Language), an object-oriented extension of SQL.

Each query is processed by a module, the Mediator Specification Interpreter (MSI) [137, 158], consisting of three main components:

- The *View Expander*, which uses mediator specification to reformulate the query into a *logical plan* by expanding the objects exported by the mediator according to their definitions. The logical plan is a set of MSL rules which refer to information at the sources.
- The *Plan Generator*, also called *Cost-Based Optimizer*, which develops a *physical plan* specifying which queries will be sent to the sources, the order in which they will be processed, and how the results of the queries will be combined in order to derive the answer to the original query.
- The *Execution engine*, which executes the physical plan and produces the answer.

The problem of query processing in TSIMMIS in the presence of limitations in accessing the sources is addressed in [121] by devising a more complex Plan Generator comprising three modules:

- a *matcher*, which retrieves queries that can process part of the logical plan;
- a *sequencer*, which pieces together the selected source queries in order to construct feasible plans;
- an *optimizer*, which selects the most efficient feasible plan.

It has to be stressed that in TSIMMIS no global integration is ever performed. Each mediator performs integration independently. As a result, for example, a certain concept may be seen in completely different and even inconsistent ways by different mediators. This form of integration can be called *query-based*, since each mediator supports a certain set of queries, i.e., those related to the view it provides.

The Garlic Project The Garlic project [42], developed at IBM Almaden Research Center, provides the user with an integrated data perspective by means of an architecture comprising a

middleware layer for query processing and data access software called *Query Services & RunTime System*. The middleware layer presents an object-oriented data model based on the ODMG standard [44] that allows data from various information sources to be represented uniformly. In such a case the global schema is simply constituted by the union of local schemas, and no integrity constraints are defined over the OMDG objects. The objects are exported by the wrappers using the Garlic Data Language (GDL), which is based on the standard Object Definition Language (ODL). Each wrapper describes data at a certain source in the OMDG format and gives descriptions of source capabilities to answer queries in terms of the query plans it supports. Note that the notion of mediator, central in the TSIMMIS system, is missing in Garlic, and most of the mediator tasks, as the integration of objects from different sources, are submitted to the wrappers. Users pose queries in terms of the objects of the global schema in an object-oriented query language which is an object-extended dialect of SQL. The *Query Services & RunTime System* produces a query execution plan in which a user query is decomposed in a set of sub-queries to the sources, by expanding each involved object with its definition provided by the relative wrapper. The query execution plan is obtained by means of some sequence of parsing, semantic checking, query reformulation and optimization, based on the information (provided by a system catalog) about where data is stored, what wrapper it is associated with, its schema, any available statistics, etc. Each involved wrapper translates the sub-queries into the source native query language taking into account the query processing power of the source. The description of the source query capabilities is used by the query optimizer to create a set of feasible plans and to select the best for execution.

The Squirrel Project In the approaches to data integration described so far, the data at the sources are not replicated in the integration system (virtual approach). In contrast, in the materialized approach, the system computes the extension of the concepts in the global schema and maintains it in a local repository. Maintenance of replicated data against updates to the sources is a central aspect in this context and the effectiveness of maintenance affects timeliness and availability of data.

The *Squirrel* System, developed at the University of Colorado [162, 161, 160, 97], provides a GAV framework for data integration based on the notion of *integration mediator*. Integration mediators are active modules that support data integration using a hybrid of virtual and materialized data approaches.

The initial work on Squirrel [162, 160] does not consider virtual views and studies the problems related to data materialization. A key feature of Squirrel mediators, which consist of software components implementing materialized integrated views over multiple sources, is their ability to incrementally maintain the integrated views by relying on the active capabilities of sources. More precisely, at startup the mediator sends to the source databases a specification of the incremental update information needed to maintain the views and expects sources to actively provide such information.

Integration by means of hybrid virtual or materialized views is addressed in [97] in the context of the relational model. However, the presented techniques can also be applied in the context of object-oriented database models. In the sequel we refer to classes and objects to indicate

data exported by mediators. The architecture of a Squirrel mediator described in [97] consists of components that deal with the problem of refreshing the materialized portion of the supported view, namely the update queue and the Incremental Update Processor (IUP), and components related to the problem of query processing, namely the Query Processor (QP) and the Virtual Attribute Processor (VAP). The QP provides the interface of the integrated view to the users. When it receives a user query, it tries to answer the query on the basis of the materialized portion of the view, maintained in a local store. If the QP needs virtual data to answer the query, the VAP constructs temporary relations containing the relevant data. To obtain temporary relations the VAP uses information provided by a Virtual Decomposition Plan (VDP) maintained in the local store. The notion of VDP is analogous to that of Query Decomposition Plan in query optimization. More specifically, the VDP specifies the classes that the mediator maintains (materialized, virtual, or hybrid), and provides the basic structure for retrieving data from the sources or supporting incremental maintenance[2].

Either for the materialized version of the mediator or for the hybrid one, an automatic generator of Squirrel integrators has been developed. Such a module takes as input a specification of the mediator expressed in a high-level Integration Specification Language (ISL). A specification in ISL includes a description of the relevant subschemas of the source databases, and the match criteria between objects of families of corresponding classes, in particular a list of the classes that are matched and a binary matching predicate specifying correspondences between objects of two classes. The output of this module is an implementation of the mediator in the *Heraclitus* language, a database programming language whose main feature is the ability of representing and manipulating collections of updates to the current database state (deltas). An object-oriented extension of Heraclitus, called *H20*, is used in [161].

The problem of object matching in Squirrel mediators is addressed in [162]. In particular, a framework is presented for supporting intricate object identifier (OID) match criteria, such as key-based matching, lookup-table-based matching, historical-based-matching, and comparison-based matching. The last criterion allows for both considering attributes other than keys in object matching, and using arbitrary boolean functions in the specification of object matching.

The MOMIS Project The MOMIS system [11, 12], jointly developed at the University of Milano and the University of Modena and Reggio Emilia, provides semi-automatic techniques for the extraction and the representation of properties holding in a single source schema (*intraschema relationships*), or between different source schemas (*interschema relationships*), and for schema clustering and integration, to identify candidates to integration and synthesize candidates into an integrated global schema. The relationships are both intensional (e.g., lexical relationships extracted according to a Wordnet supplied ontology) and extensional (e.g., lexical relationships which strengthen the corresponding intensional relationships), either defined by the designer or automatically inferred by the system. The integration process is based on a source independent object-oriented model called ODM_{I^3}, used for modeling structured and semistructured data sources in a common way. The model is described by means of the ODL_{I^3} language. In MOMIS mediators are composed of two modules:

[2]For a description of the update process and the relative update queue and IUP see [97].

1. the *Global Schema Builder*, which constructs the global schema by integrating the ODL_{I^3} source descriptions provided by the wrappers, and by exploiting the intraschema and the interschema relationships.

2. the *Query Manager*, which performs query processing and optimization. The Query Manager exploits extensional relationships to first identify all sources whose data are needed to answer a user query posed over the global schema. Then, it reformulates the original query into queries to the single sources, sends the obtained sub-queries to the wrappers, which execute them, and report the results to the Query Manager. Finally, the Query Manager combines the single results to provide the answer to the original query.

Currently, the query processing aspect is in a preliminary stage of development, and still needs further investigation.

The DIKE Project The DIKE system [134], developed at the University of Calabria in collaboration with the University of Reggio Calabria, exploits a conceptual model called SDR-Network [135, 147] in order to uniformly handle and represent heterogeneous data sources. Each source has an associated SDR-Network, constituted by a set of nodes representing concepts and a set of arcs representing relationships between pairs of concepts. DIKE provides the user with an algorithm for data source integration which takes as input two SDR-Networks, which are either associated directly to the sources or are derived from previously constructed SDR-Networks, and computes an output *global* SDR-Network. The process is carried out by exploiting synonymies, and homonymies between arcs and nodes of the SDR-Networks, and similarity relationships between sub-nets. To each derived node (resp., arc) a mapping is associated that describes the way the node has been obtained from one or more nodes (resp., arcs) belonging to the original SDR-Networks. Finally, to each mapping between nodes a view is associated that allows to obtain the instances of the target node of the mapping, from the instances of the source nodes of the mapping. At the moment the system is under development and does not present a completely defined method to answer queries posed over the global SDR-Networks. To answer a user query it simply uses the views associated to the global nodes involved in the query.

4.2.2 Beyond Unfolding

All the approaches described so far, do not provide an in-depth study of the problem of query processing in GAV, which basically is solved via unfolding. As shown in [108, 20], this limits the ability of such systems in dealing with complex integration scenarios, e.g., in the presence of incomplete data sources and integrity constraints on the global schema. On the other hand, the importance of allowing integrity constraints on the global schema has been stressed in several works on data integration [34, 75, 74]. A first attempt to deal with integration under constraints in GAV is [24], which presents a technique to process queries in the case when the global schema and sources are relational, key and foreign key constraints are expressed on the global schema, views are assumed to be sound, and queries over the global schema are union of conjunctive queries. In that paper the authors show that, starting from the retrieved global database, it

is possible to build a *canonical* database that has the property of faithfully representing all the databases that satisfy the global schema, in the sense that the set of certain answers to a given query can be obtained by evaluating the query over it. Furthermore, an algorithm is given that makes it possible to find the answers to a query over the given canonical database without actually building it. The algorithm makes use of Skolem functions and SLD-resolution to construct a suitable reformulation of the query, called *query expansion*, such that the answer to the expansion computed over the retrieved global database is equal to the answer to the original query over the canonical database. In [22], the same technique is used to deal with the mandatory participation and functional attribute constraints imposed by a conceptual model adopted to represent the global schema. However, notice that in these papers it is assumed that data at the sources are such that the global retrieved database does not violate the key constraints on the global schema, while foreign key constraints may not be satisfied.

The algorithms described above have been implemented in the IBIS system, which is described in the following.

The IBIS system The Internet-Based Information System (IBIS) [25] is a tool for the semantic integration of heterogeneous data sources, developed in the context of a collaboration between the University of Rome "La Sapienza" and CM Sistemi. IBIS adopts innovative solutions to deal with all aspects of a complex data integration environment, including source wrapping, limitations on source access, and query answering under integrity constraints. IBIS uses a relational global schema to query the data at the sources, and is able to cope with a variety of heterogeneous data sources, including data sources on the Web, relational databases, and legacy sources. Each non-relational source is wrapped to provide a relational view on it. Also, each source is considered sound. The system allows for the specification of integrity constraints on the global schema; in addition, IBIS considers the presence of some forms of constraints on the source schema, in order to perform runtime optimization during data extraction. In particular, key and foreign key constraints can be specified on the global schema, and functional dependencies and full-width inclusion dependencies, i.e., inclusions between entire relations, can be specified on the source schema.

Query processing in IBIS is separated in three phases:

1. the query is expanded to take into account the integrity constraints in the global schema;
2. the atoms in the expanded query are unfolded according to their definition in terms of the mapping, obtaining a query expressed over the sources;
3. the expanded and unfolded query is executed over the retrieved source databases (see below), to produce the answer to the original query.

Query unfolding and execution are the standard steps of query processing in GAV data integration systems, while for the expansion phase IBIS makes use of the algorithm presented in [24] that we have described above. As we already said, the expanded query has to be evaluated over the retrieved global database in order to produce the certain answers to the original query. As the construction of the retrieved global database is computationally costly, the IBIS Expander

module does not construct it explicitly. Instead, it unfolds the expanded query and evaluates the unfolded query over the *retrieved source database*, whose data are extracted by the Extractor module that retrieves from the sources all the tuples that may be used to answer the original query.

Studies on limitations in accessing the sources Other studies deal with the problem of query processing in GAV in the presence of limitations on how sources can be accessed [119, 120, 21]. Generally speaking, to answer queries over such sources one generally needs to start from a set of constants (provided e.g., by the user filling in a form, or taken from a source without access limitations) to bind attributes. Such bindings are used to access sources and there obtain new constants which in turn can be used for new accesses. Hence, query processing in GAV in the presence of access limitations in general requires the evaluation of a recursive query plan [119]. It is worth noticing that also in this case unfolding is not sufficient to answer the query. Since source accesses are costly, an important issue is how to minimize the number of accesses to the sources while still being guaranteed to obtain all possible answers to a query. [119, 120] discuss several optimizations that can be made at compile time, during query plan generation. In these papers, the authors adapt to the GAV approach the ideas of [62], where the problem is studied in the LAV approach. Roughly speaking, the proposed technique exploits knowledge about data sources, expressed by means of integrity constraints on the source schema, in order to detect unnecessary accesses. However, the optimizations proposed in [119, 120] work only for the so-called *connection queries*, a restricted form of conjunctive queries, while the problem of finding an optimization technique for the full class of conjunctive queries is left open by the authors. The problem is solved in [21], where a run-time optimization technique is presented, that takes into account, besides the integrity constraints on the sources, the tuples extracted from the sources at a certain step so as to infer, during the extraction of the data, information necessary to avoid useless accesses.

Chapter 5

Decidability and Complexity of Query Answering

In this chapter we focus on the relationship between the expressiveness of the global schema and the decidability and complexity of query answering. We recall that, in our data integration framework described in Chapter 3, the formalism adopted for modelling the global schema allows us to specify complex dependencies between global relations, by making use of different forms of integrity constraints. Furthermore, different assumptions on the mapping (strict semantics), enable for the specification of different interpretations on the data at the sources with respect to data satisfying the corresponding portion of the global schema. Then, suitable relaxations of such assumptions (loose semantics) properly allow us also to deal with inconsistency of data at the sources with respect to the constraints on the global schema.

The interaction between the classes of constraints allowed, namely, inclusion, key and exclusion dependencies, makes query answering difficult under the different semantics described, and may lead to non decidable cases. Hence, it is in general necessary to impose some restrictions on the simultaneous presence of different kinds of constraints on the global schema in order to obtain correct answers to user queries.

For the sake of simplicity, in this chapter we study query answering in the setting of a single database. Such a setting allows us to overlook here the mapping, and concentrate on the formalism adopted for the global schema, i.e., on the interaction between the constraints. Undecidability and complexity results obtained in this setting will be extended to the data integration framework in Chapters 6 and 7.

Briefly, In in the following,

(i) we identify the frontier between decidability and undecidability of query answering for both strict and loose semantics;

(ii) for the decidable cases, we establish the computational complexity of the query answering problem.

We remark that the results we have obtained for the sound semantics in a single database

setting extend previous studies on query containment under integrity constraints [101], while the results for the loose semantics extend known results in the field of inconsistent databases [86, 52], by taking into account inclusion dependencies, which add significant complexity to the problem. In particular, the key issue in our work is that we are able to deal with infinite models for a database schema, that are to be taken into account when cyclic inclusion dependencies are present in the schema. In this chapter, we present the results on IDs and KDs recently published in [29], and provide new results on EDs and their interaction with IDs and KDs.

Before proceeding we formalize in the following the connection between a single database and a data integration system, and provide both strict and loose semantics definition in this setting.

5.1 Preliminary Discussion

Actually, a relational schema $\mathcal{RS}$ corresponds to the global schema $\mathcal{G}$ of a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ in which relations in $\mathcal{G}$ are in a one-to-one correspondence with relations in $\mathcal{S}$, i.e., if $s_1/h_1, \ldots, s_n/h_n$ are the source relations, then the global relations are $g_1/h_1, \ldots, g_n/h_n$, and the GAV (or LAV) mapping is given by the n assertions $\langle g_i, g_i(X_1, \ldots, X_{h_i}) \leftarrow s_i(X_1, \ldots, X_{h_i}) \rangle$ (resp. $\langle s_i, s_i(X_1, \ldots, X_{h_i}) \leftarrow g_i(X_1, \ldots, X_{h_i}) \rangle$) for each i, $1 \leq i \leq n$. Let us now construct from a source database $\mathcal{D}_\mathcal{S}$ for $\mathcal{I}$ a global database for $\mathcal{I}$, namely, a database instance $\mathcal{D}$ for the schema $\mathcal{RS}$ such that $g_i^\mathcal{D} = s_i^{\mathcal{D}_\mathcal{S}}$. It is easy now to shift the assumptions that can be adopted on the mapping $\mathcal{M}$ directly to the database instance $\mathcal{D}$, denoted $as(\mathcal{D})$. More precisely,

- if $\mathcal{M}$ is sound, then $\mathcal{D}$ can be assumed sound, that is the semantics of $\mathcal{RS}$ with respect to $\mathcal{D}$, denoted by $sem_s(\mathcal{RS}, \mathcal{D})$, is the set of database instances $\mathcal{B}$ for $\mathcal{RS}$ such that $\mathcal{B} \supseteq \mathcal{D}$;
- if $\mathcal{M}$ is complete, then $\mathcal{D}$ can be assumed complete, that is the semantics of $\mathcal{RS}$ with respect to $\mathcal{D}$, denoted by $sem_c(\mathcal{RS}, \mathcal{D})$, is the set of database instances $\mathcal{B}$ for $\mathcal{RS}$ such that $\mathcal{B} \subseteq \mathcal{D}$;
- if $\mathcal{M}$ is exact, then $\mathcal{D}$ can be assumed exact, that is the semantics of $\mathcal{RS}$ with respect to $\mathcal{D}$, denoted by $sem_e(\mathcal{RS}, \mathcal{D})$, is actually the single database instances $\mathcal{D}$, i.e., $sem_e(\mathcal{RS}, \mathcal{D}) = \{\mathcal{D}\}$.

We also indicate with $ans_{as(\mathcal{D})}(q, \mathcal{RS}, \mathcal{D})$ the set of *answers* to a query q posed to $\mathcal{RS}$, where $as(\mathcal{D}) = s, c$ or e for the sound, complete and exact semantics, respectively. As usual, a tuple t is in the answer to q iff $t \in q^\mathcal{B}$ for each $\mathcal{B} \in sem_{as(\mathcal{D})}(\mathcal{RS}, \mathcal{D})$. Furthermore, we denote with $sem_{ls}(\mathcal{RS}, \mathcal{D})$, $sem_{lc}(\mathcal{RS}, \mathcal{D})$ and $sem_{le}(\mathcal{RS}, \mathcal{D})$ respectively the loosely-sound, loosely-complete and loosely-exact semantics, defined analogously to the loose semantics for a data integration system described in Section 3.3. More precisely, given a possibly inconsistent database $\mathcal{D}$ for $\mathcal{RS}$ an assumption $as(\mathcal{D})$ on $\mathcal{D}$, and two databases $\mathcal{B}_1$ and $\mathcal{B}_2$ consistent with $\mathcal{RS}$, we define $\mathcal{B}_1 \leq_\mathcal{D} \mathcal{B}_2$ if:

- $\mathcal{B}_1 \cap \mathcal{D} \supseteq \mathcal{B}_2 \cap \mathcal{D}$, if $as(\mathcal{D}) = s$;
- $\mathcal{B}_1 - \mathcal{D} \subseteq \mathcal{B}_2 - \mathcal{D}$, if $as(\mathcal{D}) = c$;

- $\mathcal{B}_1 \cap \mathcal{D} \supseteq \mathcal{B}_2 \cap \mathcal{D}$ and $\mathcal{B}_1 - \mathcal{D} \subseteq \mathcal{B}_2 - \mathcal{D}$, if $as(\mathcal{D}) = e$.

As usual, $\mathcal{R}_1 <_{\mathcal{D}} \mathcal{R}_2$ stands for $\mathcal{R}_1 \leq_{\mathcal{D}} \mathcal{R}_2 \wedge \mathcal{R}_2 \not\leq_{\mathcal{D}} \mathcal{R}_1$.

Hence, a database $\mathcal{B}$ for $\mathcal{RS}$ is in $sem_{l\,as(\mathcal{D})}(\mathcal{RS}, \mathcal{D})$, if

(i) $\mathcal{B}$ is consistent with $\mathcal{RS}$;

(ii) $\mathcal{B}$ is minimal with respect to $\leq_{\mathcal{D}}$.

Example 3.2.1 (contd.) Disregard now the mapping and focus only on the relational schema $\mathcal{G}_0$. Consider the database instance

$$\mathcal{D} = \{player(\mathsf{a}, \mathsf{b}), player(\mathsf{a}, \mathsf{d}), player(\mathsf{e}, \mathsf{f}), team(\mathsf{b}, \mathsf{c})\}$$

where $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{f}$ are constants of $\mathcal{U}$. It is easy to see that the strict and the loose semantics are the same of the integration setting. ∎

In the following we provide an in-depth analysis of the query answering problem in relational databases, when inclusion, key and exclusion dependencies are expressed over a schema $\mathcal{RS}$, and different assumptions on database instances for $\mathcal{RS}$ are adopted. In particular, we address the decision problem associated to query answering, that is, given a relational schema $\mathcal{RS} = \langle \Psi, \Sigma \rangle$, where $\Sigma = \Sigma_I \cup \Sigma_K \cup \Sigma_E$, a database instance $\mathcal{D}$ for $\mathcal{RS}$, an assumption $as(\mathcal{D})$ on $\mathcal{D}$, a query q of arity n and an n-tuple t of constants of the domain $\mathcal{U}$, to establish whether $t \in ans_{as(\mathcal{D})}(q, \mathcal{RS}, \mathcal{D})$.

5.2 Query Answering under the Strict Semantics

We focus here on query answering under the sound, complete and exact semantics. As illustrated in Section 3.2, when the data are considered complete the empty database always belongs to $sem_c(\mathcal{RS}, \mathcal{D})$, independently of Σ_I, Σ_K and Σ_E; therefore, for any query q we have immediately $ans_c(q, \mathcal{RS}, \mathcal{D}) = \emptyset$; hence, the answer to the decision problem is always negative for any tuple t. When the data are considered exact, we have two cases:

1. $\mathcal{D}$ satisfies Σ, therefore $sem_e(\mathcal{RS}, \mathcal{D}) = \{\mathcal{D}\}$ and $ans_e(q, \mathcal{RS}, \mathcal{D}) = q^{\mathcal{D}}$. So, query answering reduces to classical query processing in relational databases, and it is immediate to establish whether $t \in ans_e(q, \mathcal{RS}, \mathcal{D})$;

2. $\mathcal{D}$ violates at least one constraint in Σ, therefore $sem_e(\mathcal{RS}, \mathcal{D}) = \emptyset$ and $ans_e(q, \mathcal{RS}, \mathcal{D})$ consists of all tuples of the same arity as q; the answer to the decision problem is therefore affirmative, independently of q and t.

The case where the data are considered sound is more interesting: in fact, if the inclusion dependencies in Σ_I are not satisfied, we may think of adding suitable facts to $\mathcal{D}$ in order to satisfy them (according to the sound semantics, we are not allowed to repair such violations by deleting facts). In this case, if $\mathcal{D}$ key and exclusion dependencies on $\mathcal{G}$, the semantics of $\mathcal{RS}$ is constituted in general by several (possibly infinite) databases, each of which may have infinite size, since there are several ways of adding facts to $\mathcal{D}$. Query answering with sound data is therefore a

difficult task, that is not decidable in all cases. Therefore, in the following we concentrate on the sound semantics and investigate the interaction between IDs, KDs, and EDs. In particular, we identify decidable classes of integrity constraints, i.e., classes of inclusion, key and exclusion dependencies that allow for sound and complete query answering.

5.2.1 Query Answering under the Strictly-sound Semantics

First of all, it is worth noticing that, if Σ contains only KDs and EDs, a situation arises that is analogous to the one described for the exact semantics. Indeed, according to the sound semantics, we are only allowed to add facts to $\mathcal{D}$ in order to satisfy constraints, but KDs and EDs cannot be satisfied by adding facts. Hence, if $\mathcal{D}$ is not consistent with constraints in Σ_K or Σ_E, then $sem_s(\mathcal{RS}, \mathcal{D}) = \emptyset$, thus query answering is meaningless, whereas if $\mathcal{D}$ satisfies $\Sigma_K \cup \Sigma_E$, then $ans_s(q, \mathcal{RS}, \mathcal{D}) = q^{\mathcal{D}}$. Furthermore, KDs and EDs do not interfere with each other, where interfering means that satisfying a constraint may lead to the violation of other constraints. This immediately allows us to "repair" KDs and EDs separately. Notice that also different KDs (or different EDs) do not interfere with each other, whereas this is not true for IDs. Hence, we can assert that the presence of IDs significantly complicates query answering under the sound semantics.

For the sake of clarity, we next separately study query answering in the presence of IDs only, in the presence of IDs and EDs, and in the presence of IDs and KDs. Finally, we tackle the general setting.

Query Answering in the presence of IDs

In this section, we extend previous studies on query containment under integrity constraints, and show that query answering in the presence of IDs is decidable. To this aim, we need some preliminary results, presented in the milestone paper of Johnson and Klug [101], which addresses the problem of conjunctive query containment in a relational schema, in the presence of inclusion dependencies[1]. According to this paper, given a relational schema $\mathcal{RS} = \langle \Psi, \Sigma_I \rangle$, and two conjunctive queries of the same arity q_1 and q_2 expressed over $\mathcal{RS}$, to test whether $q_1 \subseteq q_2$, we first have to "freeze" the body of q_1, considering its atoms as facts in a database instance $\mathcal{D}$ for $\mathcal{RS}$, and then applying the *chase* procedure to such a database. The resulting (possibly infinite) database, denoted as $chase(\mathcal{RS}, \mathcal{D})$, is constructed by repeatedly applying, as long as it is applicable, the following rule:

> Inclusion Dependency Chase Rule.
> Suppose there is a tuple t in $r^{chase(\mathcal{RS},\mathcal{D})}$, and there is an ID in Σ_I of the form $r[\mathbf{X}_r] \subseteq s[\mathbf{X}_s]$. If there is no tuple t' in $s^{chase(\mathcal{RS},\mathcal{D})}$ such that $t'[\mathbf{X}_s] = t[\mathbf{X}_r]$, then we add a new tuple t_{chase} in $s^{chase(\mathcal{RS},\mathcal{D})}$ such that $t_{chase}[\mathbf{X}_s] = t[\mathbf{X}_r]$, and for any attribute A of s, such that $A \notin \mathbf{X}_s$, $t_{chase}[A]$ is a fresh constant from $\mathcal{U}$, not appearing elsewhere in $chase(\mathcal{RS}, \mathcal{D})$.

[1] In [101], results on query containment of CQs under IDs are also generalized to the case in which there is a limited interaction between IDs and FDs. This generalization is actually not interesting for our discussion.

It should be easy to see that $chase(\mathcal{RS}, \mathcal{D})$ can be infinite. In particular, this happens if IDs in Σ_I are cyclic. Nonetheless, it enjoys a fundamental property, as stated by the following Lemma (adapted from [24]) .

Lemma 5.2.1 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I \rangle$ be a relational schema, and $\mathcal{D}$ a database instance for $\mathcal{RS}$, then $chase(\mathcal{RS}, \mathcal{D})$ does not violate any inclusion dependencies in Σ_I.*

Proof. Suppose by contradiction that the inclusion dependency $r[\mathbf{X}] \subseteq s[\mathbf{Y}]$ is violated in $chase(\mathcal{RS}, \mathcal{D})$. This implies that there is a tuple t in $r^{chase(\mathcal{RS},\mathcal{D})}$ such that, for no tuple t' in $s^{chase(\mathcal{RS},\mathcal{D})}$, $t'[\mathbf{Y}] = t[\mathbf{X}]$. This would imply that we can apply the chase rule and insert a new tuple t_{chase} in $s^{chase(\mathcal{RS},\mathcal{D})}$ such that $t_{chase}[\mathbf{Y}] = t[\mathbf{X}]$, and for each attribute A of s such that $A \notin \mathbf{Y}$, $t_{chase}[A]$ is a fresh constant; but this contradicts the assumption. □

Johnson and Klug have proved that $q_1 \subseteq q_2$ if and only if the tuple t_f in the frozen head of q_1 belongs to $q_2^{chase(\mathcal{RS},\mathcal{D})}$. Moreover, they have shown that, to check whether $t_f \in q_2^{chase(\mathcal{RS},\mathcal{D})}$, only a finite portion of $chase(\mathcal{RS}, \mathcal{D})$ needs to be considered. Based on this property, they have defined a PSPACE algorithm that checks if $t_f \in q_2^{chase(\mathcal{RS},\mathcal{D})}$[2].

To this aim, we need to maintain distinct the set of constants of the domain $\mathcal{U}$ that occur in the database $\mathcal{D}$, that is the *active domain* of $\mathcal{D}$, from the other constants of $\mathcal{U}$, which can be used in the construction of the chase. Hereinafter, we indicate with $\mathcal{U}_\mathcal{D}$ the active domain of a database $\mathcal{D}$.

In the case of query answering, we are able to exploit the technique of Johnson and Klug. More specifically, we make use of the notion of chase as specified by the following result.

Theorem 5.2.2 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I \rangle$ be a relational schema and $\mathcal{D}$ a database instance for $\mathcal{RS}$; let q be a conjunctive query of arity n, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. We have that $t \in ans_s(q, \mathcal{RS}, \mathcal{D})$ if and only if $t \in q^{chase(\mathcal{RS},\mathcal{D})}$.*

Proof.

"⇒"Since $chase(\mathcal{RS}, \mathcal{D})$ satisfies $\mathcal{I}$, it belongs to $sem_s(\mathcal{RS}, \mathcal{D})$. From the definition of $ans_s(q, \mathcal{RS}, \mathcal{D})$, it follows that $t \in q^{chase(\mathcal{RS},\mathcal{D})}$.

"⇐"Analogously to [24], it can be proved by induction on the structure of $chase(\mathcal{RS}, \mathcal{D})$ that, for any database instance $\mathcal{B} \in sem_s(\mathcal{RS}, \mathcal{D})$, there exists a homomorphism μ that sends the tuples of $chase(\mathcal{RS}, \mathcal{D})$ to the tuples of $\mathcal{B}$. By hypothesis $t \in q^{chase(\mathcal{RS},\mathcal{D})}$, so there exists a homomorphism λ from the atoms of q to the facts of $chase(\mathcal{RS}, \mathcal{D})$; the composition $\lambda \circ \mu$ witnesses that $t \in ans_s(q, \mathcal{RS}, \mathcal{D})$. □

Based on the above property, we can apply the algorithm of [101] for query answering, i.e., to check whether a tuple t belongs to $ans_s(q, \mathcal{RS}, \mathcal{D})$. More specifically, given an UCQ $q(\vec{\mathbf{x}}) \leftarrow conj_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1) \vee \ldots \vee conj_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m)$ of arity n, an n-tuple t of constants of $\mathcal{U}_\mathcal{D}$, and a database instance $\mathcal{D}$ for $\mathcal{RS}$, we construct a query q_1 of the form $q_1(t) \leftarrow d_1, \ldots, d_k$ where $d_1, \ldots, d_k$ are all facts in $\mathcal{D}$. Then, $t \in ans_s(q, \mathcal{RS}, \mathcal{D})$ iff q_1 is contained in at least one CQ of the form

[2]Actually, Johnson and Klug have considered the case of boolean queries, for which $q_1 \subseteq q_2$ iff $q_2^{chase(\mathcal{RS},\mathcal{D})}$ is non-empty. According to our version of the algorithm, this is equivalent to check that the empty tuple $\langle\rangle$, which represents the frozen head of q_1, is contained in $q_2^{chase(\mathcal{RS},\mathcal{D})}$.

$q(\vec{\mathbf{x}}) \leftarrow conj_i(\vec{\mathbf{x}}, \vec{\mathbf{y}}_i)$, for each i, $1, \leq i \leq n$. We indicate with $\mathsf{Answer}_{JK}(\mathcal{RS}, \mathcal{D}, q, t)$ the algorithm for query answering that makes use of the algorithm for query containment of [101] as above described. From the results of [101] it easily follows that Answer_{JK} is a PSPACE algorithm.

Query Answering in the presence of IDs and EDs

Let us now add EDs to the set of integrity constraints specified on $\mathcal{RS}$, i.e., let be $\Sigma = \Sigma_I \cup \Sigma_E$. It should be easy to see that, in general, $chase(\mathcal{RS}, \mathcal{D})$ is not a database in $sem_s(\mathcal{RS}, \mathcal{D})$.

Example 5.2.3 Consider the relational schema $\mathcal{RS}_0 = \langle \Psi_0, \Sigma_0 \rangle$, where Ψ_0 contains three unary relations r_1, r_2 and r_3, and $\Sigma_0 = \Sigma_{I_0} \cup \Sigma_{E_0}$ contains the constraints

$$\begin{array}{rl} r_1[1] & \subseteq r_2[1], \text{ and} \\ (r_2[1] & \cap\, r_3[1]) = \emptyset, \end{array}$$

and assume that $\mathcal{D}_0 = \{r_1(a), r_3(a)\}$ is a database instance for $\mathcal{RS}_0$. Such database is consistent with Σ_{E_0}, but it is not consistent with Σ_{I_0}. By applying just once the inclusion dependency chase rule we obtain $chase(\mathcal{RS}_0, \mathcal{D}_0) = \{r_1(a), r_2(a), r_3(a)\}$. $chase(\mathcal{RS}_0, \mathcal{D}_0)$ is now consistent with Σ_{I_0}, but it is not consistent with Σ_{E_0}. Furthermore, no database satisfying both Σ_0 and the soundness of $\mathcal{D}_0$ exists, i.e., $sem_s(\mathcal{RS}_0, \mathcal{D}_0) = \emptyset$.

Consider now $\mathcal{D}_1 = \{r_1(a)\}$. It is easy to see that $chase(\mathcal{RS}_0, \mathcal{D}_1) = \{r_1(a), r_2(a)\}$ is consistent with both Σ_{I_0} and Σ_{E_0}, and that all databases in $sem_s(\mathcal{RS}_0, \mathcal{D}_1)$ can be obtained by consistently adding facts to $chase(\mathcal{RS}_0, \mathcal{D}_1)$. ∎

From the above example, we can argue that it should still be possible to answer a query posed on $\mathcal{RS}$ by looking only at $chase(\mathcal{RS}, \mathcal{D})$, provided some mechanism for checking if $sem_s(\mathcal{RS}, \mathcal{D})$ is not empty.

Our feeling is that, in order to assert that $sem_s(\mathcal{RS}, \mathcal{D}) \neq \emptyset$, we should consider explicitly the set of exclusion dependencies that are logical consequences of Σ_I and Σ_E and then verify wether $\mathcal{D}$ is consistent with it. We recall that a constraint δ is a logical consequence of (or is logically implied by) a set of constraints Σ, denoted $\Sigma \models \delta$, if all database instances that satisfy Σ satisfy also δ. The set of EDs that are logically implied by the set $\Sigma = \Sigma_I \cup \Sigma_E$ is denoted with $\Sigma_E^{*,\Sigma}$, or simply Σ_E^* when Σ is clear from the context.

For instance, in Example 5.2.3, $\Sigma_{E_0}^* = \Sigma_{E_0} \cup \{(r_1[1] \cap r_3[1]) = \emptyset\}$, and $\mathcal{D}_0$ is not consistent with $\Sigma_{E_0}^*$, whereas $\mathcal{D}_1$ is consistent with it.

In the following we formally prove that we can make use of the chase to answer queries in the presence of IDs and EDs.

Lemma 5.2.4 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_E \rangle$ be a relational schema, and $\mathcal{D}$ a database instance for $\mathcal{RS}$. Then $chase(\mathcal{RS}, \mathcal{D})$ satisfies $\Sigma_I \cup \Sigma_E$ if and only if $\mathcal{D}$ is consistent with Σ_E^*.*

Proof. "⇒"We first show that if $\mathcal{D}$ violates any exclusion dependency in Σ_E^*, then $chase(\mathcal{RS}, \mathcal{D})$ is not consistent with $\Sigma_I \cup \Sigma_E$. By contradiction, let us assume that $chase(\mathcal{RS}, \mathcal{D})$

satisfies $\Sigma_I \cup \Sigma_E$. This would imply that $chase(\mathcal{RS}, \mathcal{D})$ satisfies also Σ_E^*, but, since in the construction of $chase(\mathcal{RS}, \mathcal{D})$ facts are only added and never removed, also $\mathcal{D}$ would satisfy Σ_E^*, and this contradicts the assumption.

"$\Leftarrow$"We now prove that if $\mathcal{D}$ is consistent with Σ_E^*, then $chase(\mathcal{RS}, \mathcal{D})$ satisfies $\Sigma_I \cup \Sigma_E$. From Lemma 5.2.1 it follows that $chase(\mathcal{RS}, \mathcal{D})$ does not violate any inclusion dependency in Σ_I, hence it remains to prove that, $chase(\mathcal{RS}, \mathcal{D})$ satisfies Σ_E. We first show that $chase(\mathcal{RS}, \mathcal{D})$ satisfies Σ_E^*. The proof is by induction on the structure of $chase(\mathcal{RS}, \mathcal{D})$. In the following we indicate with $chase_i(\mathcal{RS}, \mathcal{D})$ the portion of $chase(\mathcal{RS}, \mathcal{D})$ obtained by applying i times the chase rule.

As for the base step, $chase_0(\mathcal{RS}, \mathcal{D}) = \mathcal{D}$, then, by the assumption, it satisfies Σ_E^*. As for the inductive step, assume by contradiction that $chase_i(\mathcal{RS}, \mathcal{D})$ is consistent with Σ_E^*, but $chase_{i+1}(\mathcal{RS}, \mathcal{D})$ is not. Let $r_1[\mathbf{X}] \subseteq r_2[\mathbf{Y}]$ be an ID in Σ_I that is violated by $chase_i(\mathcal{RS}, \mathcal{D})$, $t_1 \in r_1^{chase_i(\mathcal{RS},\mathcal{D})}$ is a tuple that violates such ID, $t_2 \in r_2^{chase_{i+1}(\mathcal{RS},\mathcal{D})}$ is the tuple generated by the chasing rule, i.e., let be $t_2[\mathbf{Y}] = t_1[\mathbf{X}]$, and let $(r_2[\mathbf{Z}] \cap r_3[\mathbf{W}]) = \emptyset$ be the ED in Σ_E violated in $chase_{i+1}(\mathcal{RS}, \mathcal{D})$. Then, there must exist $t_3 \in r_3^{chase_{i+1}(\mathcal{RS},\mathcal{D})}$ such that $t_3[\mathbf{W}] = t_2[\mathbf{Z}]$. According to the chase rule, for any attribute A of r_2 such that $A \notin \mathbf{Y}$, $t_2[A]$ is a fresh constant of $\mathcal{U}$ that does not appear in $chase_i(\mathcal{RS}, \mathcal{D})$. Hence, since also $t_3 \in r_3^{chase_i(\mathcal{RS},\mathcal{D})}$, we have that $\mathbf{Z} \subseteq \mathbf{Y}$. It should be easy to see now that Σ_E^* contains the ED $(r_1[\mathbf{X}'] \cap r_3[\mathbf{W}]) = \emptyset$, where $\mathbf{X}'$ is a sequence of attributes of r_1 such that $\mathbf{X}' \subseteq \mathbf{X}$ and $t_1[\mathbf{X}'] = t_2[\mathbf{Z}]$. Hence, $t_1[\mathbf{X}'] = t_3[\mathbf{W}]$, and so $chase_i(\mathcal{RS}, \mathcal{D})$ does not satisfy Σ_E^*, but this contradicts the inductive assumption.

From the fact that $chase(\mathcal{RS}, \mathcal{D})$ satisfies Σ_E^*, it immediately follows that $chase(\mathcal{RS}, \mathcal{D})$ satisfies Σ_E. □

With this result in place, we are able to extend the result of [101] to the case of IDs and EDs.

Theorem 5.2.5 (IDs-EDs Separation) *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_E \rangle$ be a relational schema, and let $\mathcal{RS}_1 = \langle \Psi, \Sigma_I \rangle$ be the relational schema obtained from $\mathcal{RS}$ by removing the EDs. Let $\mathcal{D}$ be a database instance for $\mathcal{RS}$ and $\mathcal{RS}_1$, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. We have that $t \notin ans_s(q, \mathcal{RS}, \mathcal{D})$ iff $\mathcal{D}$ is consistent with Σ_E^* and $t \notin ans_s(q, \mathcal{RS}_1, \mathcal{D})$.*

Proof. "$\Rightarrow$"By hypothesis $t \notin ans_s(q, \mathcal{RS}, \mathcal{D})$; this means that there exists $\mathcal{B} \in sem_s(\mathcal{RS}, \mathcal{D})$, such that $t \notin q^\mathcal{B}$. Since $\mathcal{B}$ satisfies $\Sigma_I \cup \Sigma_E$, and $\mathcal{D} \subseteq \mathcal{B}$, it easily follows that $\mathcal{D}$ satisfies Σ_E^*. It is also immediate to verify that $\mathcal{B} \in sem_s(\mathcal{RS}_1, \mathcal{D})$, therefore $t \notin ans_s(q, \mathcal{RS}_1, \mathcal{D})$, thus proving the claim.

"$\Leftarrow$"By hypothesis $t \notin ans_s(q, \mathcal{RS}_1, \mathcal{D})$ and $\mathcal{D}$ satisfies Σ_E^*. From the former condition, according to Theorem 5.2.2, it follows that $t \notin q^{chase(\mathcal{RS}_1,\mathcal{D})}$, whereas from the latter condition, according to Lemma 5.2.4, it follows that $chase(\mathcal{RS}, \mathcal{D})$ satisfies $\Sigma_I \cup \Sigma_E$. Since $chase(\mathcal{RS}, \mathcal{D}) = chase(\mathcal{RS}_1, \mathcal{D})$ by construction, then $chase(\mathcal{RS}_1, \mathcal{D}) \in sem_s(\mathcal{RS}, \mathcal{D})$. The claim follows immediately. □

Before concluding, we provide now a set of inference rules $\mathcal{I}_{ED}$ that allows for the computation of $\Sigma_E^{*,\Sigma}$, where $\Sigma = \Sigma_I \cup \Sigma_E$, and we prove that such inference rules are correct and complete with respect to the problem of inferring logical implication of EDs by IDs and EDs.

In the following set $\mathcal{I}_{ED}$ of inference rules, r, s, r_1, r_2 are relations, whereas $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$, $\mathbf{W}$, $\mathbf{X}'$ and $\mathbf{Y}'$ range over sequence of attributes of such relations.

ID-ED1: **If** $(r[\mathbf{X}] \cap r[\mathbf{Y}]) = \emptyset$, where $\mathbf{X} = X_1, \ldots, X_n$ and $\mathbf{Y} = Y_1, \ldots, Y_n$, **then** $(r[\mathbf{X}'] \cap s[\mathbf{Y}']) = \emptyset$, where $\mathbf{X}'$ is every set of attributes in r such that $\mathbf{X}' = X_1, \ldots, X_n, X_{n+1}, \ldots, X_m$ and $\mathbf{Y}'$ is every set of attributes in s such that $\mathbf{Y}' = Y_1, \ldots, Y_n, Y_{n+1}, \ldots, Y_m$.

ID-ED2: **If** $(r[\mathbf{Z}] \cap r[\mathbf{Z}]) = \emptyset$, **then** $(r[\mathbf{X}] \cap s[\mathbf{Y}]) = \emptyset$ for all relations s and for all sequences of attributes $\mathbf{X}$ and $\mathbf{Y}$[3].

ID-ED3: **If** $r_1[\mathbf{X}] \subseteq r_2[\mathbf{Y}]$ and $(r_2[\mathbf{Z}] \cap r_3[\mathbf{W}]) = \emptyset$ such that $\mathbf{Z} \subseteq \mathbf{Y}$, **then** $(r_1[\mathbf{X}'] \cap r_3[\mathbf{W}]) = \emptyset$ where $\mathbf{X}' \subseteq \mathbf{X}$ is such that $r_1[\mathbf{X}'] \subseteq r_2[\mathbf{Z}]$.

We now prove soundness and completeness of the above rules, i.e., we show that $\Sigma \vdash_{\mathcal{I}_{ED}} \delta$ if and only if $\Sigma \models \delta$, where δ is an ED, and $\Sigma \vdash_{\mathcal{I}_{ED}} \delta$ indicates that δ is provable from Σ using $\mathcal{I}_{ED}$ (we omit the subscript $\mathcal{I}_{ED}$ when clear from the context). We say that the ED δ is *provable* from Σ using $\mathcal{I}_{ED}$ if there exists a sequence of IDs and EDs (called *proof*) $\delta_1, \ldots, \delta_n = \delta$, $(n \geq 1)$ such that for each $i \in \{1, \ldots, n\}$ either

(a) $\delta_i \in \Sigma$, or

(b) there is a substitution for some rule $\rho \in \mathcal{I}_{ED}$ such that δ_i corresponds to the consequence of ρ, and such that for each ID and ED in the antecedent of ρ, the corresponding ID and ED is in the set $\{\delta_j \mid 1 \leq j < i\}$.

Theorem 5.2.6 *The set $\mathcal{I}_{ED} = \{$ID-ED1,ID-ED2,ID-ED3$\}$ is sound and complete for logical implication of EDs by IDs and EDs.*

Proof. Soundness of the rules is easily verified. For completeness, let $\mathcal{RS} = \langle \Psi, \Sigma \rangle$ be a relational schema where $\Sigma = \Sigma_I \cup \Sigma_E$ is a set of IDs and EDs over $\mathcal{RS}$, and let δ be the exclusion dependency $(r[\mathbf{X}] \cap s[\mathbf{Y}]) = \emptyset$ such that $\Sigma \models \delta$.

We construct a database instance $\mathcal{DB}$ for $\mathcal{RS}$ such that $\mathcal{DB} \not\models \delta$ and we use it to demonstrate that $\Sigma \vdash \delta$.

Let r/n and s/m, be two relations in Ψ, and $\mathbf{X} = X_1, \ldots, X_k$ and $\mathbf{Y} = Y_1, \ldots, Y_k$ be two sequences of attributes of r and s, respectively. Suppose by contradiction that $\Sigma \not\vdash \delta$, and consider a database instance $\mathcal{D}$ for $\mathcal{RS}$ such that $\mathcal{D} = \{r(a_1, \ldots, a_k, b_1, \ldots, b_{n-k}), s(a_1, \ldots, a_k, c_1, \ldots, c_{m-k})\}$[4], where $a_i \neq a_l$, $a_i \neq b_j$, $a_i \neq c_h$ and $b_j \neq c_h$ for $i = 1, \ldots, k$, $l = 1, \ldots, k$ and $l \neq i$, $j = 1, \ldots, n-k$, and $h = 1, \ldots, m-k$. It's easy to see that $\mathcal{D}$ does not satisfy δ, and that $chase(\mathcal{RS}, \mathcal{D})$ does not satisfy δ as well.

We next show that $chase(\mathcal{RS}, \mathcal{D})$ is consistent with Σ (thus contradicting the assumption that $\Sigma \models \delta$).

As for IDs, from Lemma 5.2.1 it follows that $chase(\mathcal{RS}, \mathcal{D})$ is consistent with Σ_I.

[3]Notice that, $(r[\mathbf{Z}] \cap r[\mathbf{Z}]) = \emptyset$ is satisfied by a database instance $\mathcal{B}$ if $r^{\mathcal{B}}$ is empty.

[4]Without loss of generality we assume that $\mathbf{X}$ and $\mathbf{Y}$ are the first k attributes in r and s, respectively.

We consider now the EDs. Let Σ' be the set of constraints obtained by adding to Σ the EDs inferred by Σ using $\mathcal{I}_{ED}$. We first show that $chase(\mathcal{RS}, \mathcal{D})$ does not violate EDs in Σ'. The proof is by induction on the construction of $chase(\mathcal{RS}, \mathcal{D})$. We indicate with $chase(\mathcal{RS}, \mathcal{D})_i$ the portion of $chase(\mathcal{RS}, \mathcal{D})$ obtained by applying i times the chase rule.

As for the base step, $chase(\mathcal{RS}, \mathcal{D})_0 = \mathcal{D}$, hence, according to the construction of $\mathcal{D}$, $chase(\mathcal{RS}, \mathcal{D})_0$ does not violate (i) EDs that do not involve r and s, (ii) EDs of the form $(r[\mathbf{X}'] \cap s[\mathbf{Y}']) = \emptyset$ where $\mathbf{X}' \not\subseteq \mathbf{X}$ or $\mathbf{Y}' \not\subseteq \mathbf{Y}$, and (iii) EDs of the form $(r[\mathbf{Z}] \cap r[\mathbf{W}]) = \emptyset$ (resp. $(s[\mathbf{Z}] \cap s[\mathbf{W}]) = \emptyset$) where $\mathbf{Z} = Z_1, \ldots, Z_k$ and $\mathbf{W} = W_1, \ldots, W_k$ and there exists $i \in \{1, \ldots, k\}$ such that $Z_i \neq W_i$.

It remains to prove that $chase(\mathcal{RS}, \mathcal{D})_0$ does not violate

(iv) EDs of the form $(r[\mathbf{X}'] \cap s[\mathbf{Y}']) = \emptyset$, where $\mathbf{X}'$ is every set of attributes in r such that $\mathbf{X}' = X_1, \ldots, X_k, X_{k+1}, \ldots, X_h$ and $\mathbf{Y}'$ is every set of attributes in s such that $\mathbf{Y}' = Y_1, \ldots, Y_k, Y_{k+1}, \ldots, Y_h$;

(v) EDs of the form $(r[\mathbf{X}] \cap r[\mathbf{X}]) = \emptyset$ and $(s[\mathbf{X}] \cap s[\mathbf{X}]) = \emptyset$).

However, constraints of the form (iv) (reps. (v)) cannot be in Σ', otherwise according to rule ID-ED1 (resp. ID-ED2) it should be $\Sigma' \vdash \delta$.

As for the inductive step, assume by contradiction that $chase_i(\mathcal{RS}, \mathcal{D})$ is consistent with Σ', but $chase_{i+1}(\mathcal{RS}, \mathcal{D})$ is not. Let $r_1[\mathbf{X}] \subseteq r_2[\mathbf{Y}]$ be an ID in Σ' that is violated by $chase_i(\mathcal{RS}, \mathcal{D})$, $t_1 \in r_1^{chase_i(\mathcal{RS},\mathcal{D})}$ is a tuple that violates such ID, $t_2 \in r_2^{chase_{i+1}(\mathcal{RS},\mathcal{D})}$ is the tuple generated by the chasing rule, i.e., let be $t_2[\mathbf{Y}] = t_1[\mathbf{X}]$, and let $(r_2[\mathbf{Z}] \cap r_3[\mathbf{W}]) = \emptyset$ be the ED in Σ' violated in $chase_{i+1}(\mathcal{RS}, \mathcal{D})$. Then, there must exist $t_3 \in r_3^{chase_{i+1}(\mathcal{RS},\mathcal{D})}$ such that $t_3[\mathbf{W}] = t_2[\mathbf{Z}]$. According to the chase rule, for any attribute A of r_2 such that $A \notin \mathbf{Y}$, $t_2[A]$ is a fresh constant of $\mathcal{U}$ that does not appear in $chase_i(\mathcal{RS}, \mathcal{D})$. Hence, since also $t_3 \in r_3^{chase_i(\mathcal{RS},\mathcal{D})}$, we have that $\mathbf{Z} \subseteq \mathbf{Y}$. It should be easy to see now that, according to the ID-ED3 rule, from the above ID and ED we can prove the exclusion dependency $(r_1[\mathbf{X}'] \cap r_3[\mathbf{W}]) = \emptyset$, where $\mathbf{X}'$ is a sequence of attributes of r_1 such that $\mathbf{X}' \subseteq \mathbf{X}$ and $t_1[\mathbf{X}'] = t_2[\mathbf{Z}]$. Hence, $t_1[\mathbf{X}'] = t_3[\mathbf{W}]$, and so $chase_i(\mathcal{RS}, \mathcal{D})$ does not satisfy $(r_1[\mathbf{X}'] \cap r_3[\mathbf{W}]) = \emptyset$, but this contradicts the inductive assumption.

Since $\Sigma_E \subseteq \Sigma'$, it immediately follows that $chase(\mathcal{RS}, \mathcal{D})$ is consistent with Σ_E, but since it does not satisfy δ, we can conclude that δ is not a logical consequence of Σ. □

Finally, we provide a complexity result for logical implication of EDs by IDs and EDs.

Theorem 5.2.7 *Let $\mathcal{RS} = \langle \Psi, \Sigma \rangle$ be a relational schema, where $\Sigma = \Sigma_I \cup \Sigma_E$, and let δ be an exclusion dependency. Then, the problem of deciding wether $\Sigma \models \delta$ is PSPACE-complete.*

Proof. As for the membership, we show that there exist a non-deterministic polynomial space algorithm for deciding logical implication of δ by Σ. Then, by Savitch's theorem [146] this can be transformed into a deterministic algorithm that runs in polynomial space. More precisely, in order to prove that $\Sigma \models \delta$, we check if $\Sigma \vdash_{\mathcal{I}_{ED}} \delta$, i.e., we look for a proof $\delta_1, \ldots, \delta_n$, such that $\delta_n = \delta$. Without loss of generality, we assume that no useless dependencies are in the proof, i.e., every dependency, except δ_n, appears in the antecedent of at least one inference rule of $\mathcal{I}_{ED}$. According to the set $\mathcal{I}_{ED}$, to generate in the proof δ_i, such that $1 \leq i \leq n$ and $\delta_i \notin \Sigma$,

we need only δ_{i-1} (if δ_i is generated by applying ID-ED1 or ID-ED2) or δ_{i-1} and δ_j such that $1 \leq j < i-1$ (if δ_i is generated by applying ID-ED3). Since in this last situation one among δ_{i-1} and δ_j is an ID belonging to Σ_I, the additional space required in both cases is bounded to the space required for storing a single ED, rather than the complete proof at the $(i-1)$-th step. Moreover, the number of EDs that are logical consequence of Σ is bounded to $O(2^{n \log n})$, where n is the maximum between the number of relations in Ψ and the maximum arity of the relations in Ψ. Hence, a counter of $n \log n$ bits allows us to stop the non-deterministic computation after $2^{n \log n}$ steps if δ has not still been proved.

Hardness easily follows from the fact that deciding logical implication of EDs by IDs and EDs actually subsumes deciding logical implication of IDs. Since logical implication of IDs is a PSPACE-complete problem [2], the hardness follows. □

Query Answering in the presence of IDs and KDs

Let us now study the case in which IDs and KDs are expressed over a relational schema, i.e., consider the schema $\mathcal{RS} = \langle \Psi, \Sigma \rangle$, where $\Sigma = \Sigma_I \cup \Sigma_K$. Differently from the situation studied before, query answering in such a setting is not decidable in all cases.

We now define a restricted class of dependencies under which query answering is decidable.

Definition 5.2.8 Given a relational schema $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K \rangle$, an inclusion dependency in Σ_I of the form $r_1[\mathbf{A}_1] \subseteq r_2[\mathbf{A}_2]$ is a *non-key-conflicting inclusion dependency* (NKCID) with respect to Σ_K if either: (i) no KD is defined on r_2, or (ii) the KD $key(r_2) = \mathbf{K}$ is in Σ_K, and $\mathbf{A}_2$ is not a strict superset of $\mathbf{K}$, i.e., $\mathbf{A}_2 \not\supset \mathbf{K}$. Moreover, the schema $\mathcal{RS}$ is *non-key-conflicting* (NKC) if all the IDs in Σ_I are NKCIDs with respect to Σ_K.

Informally, a set of dependencies is NKC if no ID in Σ_I propagates a proper subset of the key of the relation in its right-hand side. We point out that the class of NKCIDs comprises the well-known class of *foreign key dependencies*, which corresponds to IDs of the form $r_1[\mathbf{A_1}] \subseteq r_2[\mathbf{A_2}]$ such that $key(r_2) = \mathbf{A_2}$.

We first show that, as soon as we extend the class of dependencies beyond the non-key-conflicting case, query answering is undecidable. In particular, we introduce, together with KDs, inclusion dependencies of the form $r_1[\mathbf{A}_1] \subseteq r_2[\mathbf{A}_2]$ such that, if the KD $key(r_2) = \mathbf{K}$ is in Σ_K, $\mathbf{A}_2$ is allowed to cover $\mathbf{K}$ plus at most one attribute of r_2. We will call such IDs *1-key-conflicting IDs* (1KCIDs) with respect to Σ_K. A *1-key-conflicting* (1KC) relational schema is defined analogously to a NKC schema. We first show undecidability of implication of KDs and 1KCIDs.

Theorem 5.2.9 *The problem of implication for KDs and 1KCIDs is undecidable.*

Proof. The proof is by reduction from the more general problem of implication of functional dependencies and inclusion dependencies. Consider a generic instance of this problem, i.e., given a relational schema $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_F \rangle$, where Σ_I is a set of IDs and Σ_F is a set of FDs, and an inclusion dependency δ. We assume that all FDs in Σ_F are in *normal form*, i.e. of the form

$r : \mathbf{A} \rightarrow B$, where a single attribute B is in the right-hand side. We construct an *ad hoc* problem of implication of KDs and 1KCIDs, consisting of a relational schema $\mathcal{RS}_1 = \langle \Psi_1, \Sigma_{I_1}, \Sigma_{K_1} \rangle$, where Σ_{I_1} is a set of 1KCID with respect to Σ_{K_1}, and the same dependency δ. We will show that the two problems are equivalent, i.e. $(\Sigma_I \cup \Sigma_F) \models \delta$ if and only if $(\Sigma_{I_1} \cup \Sigma_{K_1}) \models \delta$. The dependencies Σ_{I_1} and Σ_{K_1}, defined in a new relational schema $\mathcal{RS}_1 = \langle \Psi_1, \Sigma_{I_1}, \Sigma_{K_1} \rangle$, are constructed as follows.

- The new set of relations Ψ_1 includes all relations in Ψ (plus those added as below).
- Σ_{I_1} includes all IDs in Σ_I (plus those added as below).
- For each FD φ in Σ_F of the form
$$r : \mathbf{A} \rightarrow B,$$
we add to Ψ_1 an auxiliary relation r_φ of arity $|\mathbf{A}| + 1$, and we add to Σ_{I_1} the dependencies
$$\begin{array}{lrcl} \gamma_1 : & r_\varphi[\mathbf{A}, B] & \subseteq & r[\mathbf{A}, B] \\ \gamma_2 : & r[\mathbf{A}, B] & \subseteq & r_\varphi[\mathbf{A}, B], \end{array}$$
and to Σ_{K_1} key dependency
$$\varkappa : \quad key(r_\varphi) = \mathbf{A}.$$

Note that all the IDs in Σ_{I_1} are 1KCIDs with respect to Σ_{K_1}, since keys in Σ_{K_1} are only those expressed on the auxiliary relations r_φ, and the only IDs on r_φ are of the form γ_2. The following result will be used in the rest of the proof.

Lemma 5.2.10 *For any database $\mathcal{B}_1$ for $\mathcal{RS}_1$, we have that $\mathcal{B}_1$ satisfies $\{\varphi, \gamma_1, \gamma_2\}$ if and only if $\mathcal{B}_1$ satisfies $\{\varkappa, \gamma_1, \gamma_2\}$.*

Proof. "⇒"By hypothesis, for any two tuples $t_1, t_2 \in R^{\mathcal{B}_1}$, it cannot happen that $t_1[\mathbf{A}] = t_2[\mathbf{A}]$ and $t_1[B] = t_2[B]$. Since by dependencies γ_1, γ_2 we have $R^{\mathcal{B}}[\mathbf{A}, B] = R^{\mathcal{B}}_\varphi[\mathbf{A}, B]$, we deduce that the same property holds for all tuples in $R^{\mathcal{B}_1}_\varphi$; therefore $\varkappa$ is satisfied by definition.

"⇐"By hypothesis, for any two tuples $t_1, t_2 \in R^{\mathcal{B}_1}_\varphi$, it cannot happen that $t_1[\mathbf{A}] = t_2[\mathbf{A}]$ and $t_1[B] = t_2[B]$. Since by dependencies γ_1, γ_2 we have $R^{\mathcal{B}_1}[\mathbf{A}, B] = R^{\mathcal{B}_1}_\varphi[\mathbf{A}, B]$, we deduce that the same property holds for all tuples in $R^{\mathcal{B}_1}$; therefore φ is satisfied by definition. □

From this result it follows that we are able to simulate general FDs by using KDs and 1KCIDs only. Now we end the reduction by showing that $(\Sigma_I \cup \Sigma_F) \models \delta$ if and only if $(\Sigma_{I_1} \cup \Sigma_{K_1}) \models \delta$.

"⇒"By contradiction, suppose $(\Sigma_{I_1} \cup \Sigma_{K_1}) \not\models \delta$; then there exists a database $\mathcal{B}_1$ for $\mathcal{RS}_1$ such that $\mathcal{B}_1$ satisfies $(\Sigma_{I_1} \cup \Sigma_{K_1})$ and violates δ. Consider a database $\mathcal{B}$ for $\mathcal{RS}$ obtained from $\mathcal{B}_1$ by removing the facts associated with the relations of the form r_φ introduced in the reduction. By Lemma 5.2.10, $\mathcal{B}$ satisfies $(\Sigma_I \cup \Sigma_F)$; moreover, $\mathcal{B}$ cannot satisfy δ because $\mathcal{B}$ coincides with $\mathcal{B}_1$ on the relations in Ψ.

"⇐"By contradiction, suppose $(\Sigma_I \cup \Sigma_F) \not\models \delta$; then there exists a database $\mathcal{B}$ for $\mathcal{RS}$ such that $\mathcal{B}$ satisfies $(\Sigma_I \cup \Sigma_F)$ and violates δ. We construct a database $\mathcal{B}_1$ for $\mathcal{RS}_1$ that coincides with $\mathcal{B}$ on the relations in Ψ, and such that the facts associated with the relations of the form r_φ, introduced in the reduction, are such that the dependencies of the form γ_1, γ_2 are satisfied.

By Lemma 5.2.10, $\mathcal{B}_1$ satisfies $(\Sigma_{I_1} \cup \Sigma_{K_1})$; moreover, it cannot satisfy δ because $\mathcal{B}_1$ coincides with $\mathcal{B}$ on the relations in Ψ.

The reduction is clearly computable in a finite amount of time. Since implication of IDs and FDs is undecidable, the thesis follows. □

As similar result on undecidability of the implication problem for multiple key dependencies and foreign key dependencies has been established in [73, 72]. The connection between the two problems can be intuitively seen from the fact that a foreign key constraint that presents in its right-hand side a key $\mathbf{K}_1$, may also be a key conflicting inclusion dependency with respect to another key $\mathbf{K}_2$ of the same relation, if $\mathbf{K}_2 \subset \mathbf{K}_1$. However, our result (that has been developed independently of the one in [73, 72]) is slightly more general since proves undecidability of implication for 1KCIDs.

We now show that query answering is undecidable in the presence of KDs and 1KCIDs.

Theorem 5.2.11 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K \rangle$ be a 1KC schema, $\mathcal{D}$ a database instance for $\mathcal{RS}$, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. The problem of establishing whether $t \in ans_s(q, \mathcal{RS}, \mathcal{D})$ is undecidable.*

Proof. The proof is analogous to a proof of PSPACE-hardness of an analogous result (addressed in the context of query containment) proved by Vardi and published in [101]. We will show a counterexample in which the problem is undecidable. Let δ be the following inclusion dependency:

$$r[A_1, \ldots, A_k] \subseteq s[B_1, \ldots, B_k]$$

where r has arity n and s has arity m. Without loss of generality, δ involves the first k attributes of r and s respectively. We choose a database instance $\mathcal{D}$ for $\mathcal{RS}$ containing the single fact $r(c_1, \ldots, c_n)$. Then we consider the following boolean query:

$$q \leftarrow r(X_1, \ldots, X_n), s(X_1, \ldots, X_k, Y_{k+1}, \ldots, Y_m)$$

Note that the query q has a positive answer (i.e., $\langle\rangle \in ans_s(q, \mathcal{RS}, \mathcal{D})$) if and only if the fact $s(c_1, \ldots, c_k, d_{k+1}, \ldots, d_m)$ is in all databases in $sem_s(\mathcal{RS}, \mathcal{D})$. It is immediate to see that this is true if and only if $(\Sigma_I \cup \Sigma_K) \models \delta$. Since implication of 1KCIDs and KDs is undecidable, the thesis follows. □

As an immediate consequence of this theorem, undecidability of query answering in the presence of KDs and general IDs follows. Moreover, the problem is still undecidable if we restrict to the class of instances consistent with the key dependencies.

Corollary 5.2.12 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K \rangle$ be a 1KC schema, $\mathcal{D}$ a database instance for $\mathcal{RS}$ consistent with Σ_K, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. The problem of establishing whether $t \in ans_s(q, \mathcal{RS}, \mathcal{D})$ is undecidable.*

Proof. The case where $\mathcal{D}$ does not satisfy Σ_K is clearly decidable, since in that case the answer to the problem is always affirmative. The claim follows immediately. □

Now we come to query answering in the case of NKCIDs and KDs, and prove that this problem is decidable. The most relevant property of NKCIDs is that they do not interfere with KDs, so that we can operate with NKCIDs just as if the KDs were not defined in the schema. In particular, we can again make use of the chase to answer queries in the presence of IDs and KDs. This property is expressed by the following result.

Lemma 5.2.13 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K \rangle$ be a NKC schema, and $\mathcal{D}$ a database instance for $\mathcal{RS}$. Then $chase(\mathcal{RS}, \mathcal{D})$ satisfies Σ_I and Σ_K if and only if $\mathcal{D}$ is consistent with Σ_K.*

Proof. "⇒"We show that if $\mathcal{D}$ violates any key dependency in Σ_K, then $chase(\mathcal{RS}, \mathcal{D})$ is not consistent with $\Sigma_I \cup \Sigma_K$. By contradiction, let us assume that $chase(\mathcal{RS}, \mathcal{D})$ satisfies $\Sigma_I \cup \Sigma_K$. Since in the construction of $chase(\mathcal{RS}, \mathcal{D})$ facts are only added and never removed, also $\mathcal{D}$ would satisfy Σ_K, and this contradicts the assumption.

"⇐"We now prove that if $\mathcal{D}$ is consistent with Σ_K, then $chase(\mathcal{RS}, \mathcal{D})$ satisfies $\Sigma_I \cup \Sigma_K$. From Lemma 5.2.1 it follows that $chase(\mathcal{RS}, \mathcal{D})$ does not violate any inclusion dependency in Σ_I, hence it remains to prove that, $chase(\mathcal{RS}, \mathcal{D})$ satisfies Σ_K. The proof is by induction on the structure of $chase(\mathcal{RS}, \mathcal{D})$. First, by hypothesis $\mathcal{D}$ is consistent with Σ_K, and this verifies the base step. For the induction step, suppose we insert in $chase(\mathcal{RS}, \mathcal{D})$ a tuple t into a relation r, on which a key dependency $key(r) = \mathbf{K}$ is defined, according to the ID $s[\mathbf{A}] \subseteq r[\mathbf{B}]$. We will show that there is no violation of the key dependencies on r, by showing that t does not agree on $\mathbf{K}$ with any pre-existing tuple t in $r^{chase^*(\mathcal{RS},\mathcal{D})}$, where $chase^*(\mathcal{RS}, \mathcal{D})$ is the portion of $chase(\mathcal{RS}, \mathcal{D})$ constructed until the insertion of t.

According to the definition of NKCIDs, the possible cases are the following.

1. $\mathbf{B} = \mathbf{K}$. In this case we have a foreign key dependency; t and $\bar{t}$ cannot agree on $\mathbf{K}$, because in that case t wouldn't have been added.

2. $\mathbf{B} \subset \mathbf{K}$. The two tuples differ on the constants of $\mathbf{B}$ (otherwise only one of the two would have been added), so they differ also on $\mathbf{K}$.

3. $\mathbf{B} \cap \mathbf{K} \neq \emptyset$ and $\mathbf{B} - \mathbf{K} \neq \emptyset$. In this case $\mathbf{B}$ partially overlaps with $key(r)$; we necessarily have $\mathbf{K} - \mathbf{B} \neq \emptyset$, otherwise $\mathbf{B}$ would be a strict superset of $\mathbf{K}$. Therefore t and $\bar{t}$ differ in the constants in $\mathbf{K} - \mathbf{B}$, where t has fresh constants, thus they differ *a fortiori* on $\mathbf{K}$.

4. $\mathbf{B} \cap \mathbf{K} = \emptyset$. In this case the two tuples differ in the constants in $\mathbf{K}$, where t has fresh constants.

□

Analogously to the case of IDs and EDs, we exploit the above lemma to extend the result of [101] to the case of IDs and KDs.

Theorem 5.2.14 (IDs-KDs Separation) *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K \rangle$ be a NKC relational schema, and let $\mathcal{RS}_1 = \langle \Psi, \Sigma_I \rangle$ be the relational schema obtained from $\mathcal{RS}$ by removing the KDs. Let $\mathcal{D}$ be a database instance for $\mathcal{RS}$ and $\mathcal{RS}_1$, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. We have that $t \notin ans_s(q, \mathcal{RS}, \mathcal{D})$ iff $\mathcal{D}$ is consistent with Σ_K and $t \notin ans_s(q, \mathcal{RS}_1, \mathcal{D})$.*

Proof. Easily obtainable from the proof of Theorem 5.2.5 by replacing Σ_E and Σ_E^* with Σ_K, and Lemma 5.2.4 with Lemma 5.2.13. □

Query Answering in the presence of IDs, KDs, and EDs

Finally, we consider the general case in which IDs, KDs and EDs are expressed over a relational schema $\mathcal{RS}$. Obviously, according to Theorem 5.2.11, in the presence of 1KCIDs with respect to the KDs on $\mathcal{RS}$, query answering is undecidable, being the case in which only IDs and KDs are expressed over $\mathcal{RS}$ a special case of the setting studied here. In particular, the problem is undecidable if we restrict to the class of instances consistent with the key and exclusion dependencies, as stated by the following corollary that is a straightforward generalization of Corollary 5.2.12 and that will be useful in the following.

Corollary 5.2.15 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ be a 1KC schema[5], $\mathcal{D}$ a database instance for $\mathcal{RS}$ consistent with $\Sigma_K \cup \Sigma_E^*$, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. The problem of establishing whether $t \in ans_s(q, \mathcal{RS}, \mathcal{D})$ is undecidable.*

As for the case of NKC schemas, the following theorem states that query answering is decidable. The theorem, whose proof follows straightforwardly from Theorem 5.2.5 and Theorem 5.2.14, integrates the results that we have presented separately for IDs and EDs, and for IDs and KDs.

Theorem 5.2.16 (IDs-EDs-KDs Separation) *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ be a NKC schema, and let $\mathcal{RS}_1 = \langle \Psi, \Sigma_I, \rangle$ be the relational schema obtained from $\mathcal{RS}$ by removing the KDs and EDs. Let $\mathcal{D}$ be a database instance for $\mathcal{RS}$ and $\mathcal{RS}_1$, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. We have that $t \notin ans_s(q, \mathcal{RS}, \mathcal{D})$ iff $\mathcal{D}$ is consistent with Σ_K and Σ_E^* and $t \notin ans_s(q, \mathcal{RS}_1, \mathcal{D})$.*

Based on the above theorem, we define the algorithm $\mathsf{Answer_S}$, that solves query answering in the case of NKCIDs, KDs and EDs.

Algorithm $\mathsf{Answer_S}(\mathcal{RS}, \mathcal{D}, q, t)$
Input: NKC database schema $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$, database instance $\mathcal{D}$,
query q of arity n over $\mathcal{RS}$, n-tuple t of constants of $\mathcal{U}_\mathcal{D}$;
Output: *true* if $t \in ans_s(q, \mathcal{RS}, \mathcal{D})$, *false* otherwise;
if $\mathcal{D}$ is not consistent with $\Sigma_K \cup \Sigma_E^*$
then return *true*
else return $\mathsf{Answer}_{JK}(\langle \Psi, \Sigma_I \rangle, \mathcal{D}, q, t)$

Soundness and completeness of the algorithm follow immediately from Theorem 5.2.16 and from soundness and completeness of Answer_{JK} [101].

To conclude the section, we present a complexity result for query answering in our setting under the sound semantics.

[5]Hereinafter, we use the terns NKC and 1KC schemas even when also EDs are issued over the global schema.

Theorem 5.2.17 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ be a NKC schema, $\mathcal{D}$ a database instance for $\mathcal{RS}$, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. The problem of establishing whether $t \in ans_s(q, \mathcal{RS}, \mathcal{D})$ is PSPACE-complete with respect to combined complexity. Moreover, it is in PTIME in data complexity.*

Proof. From the results in [101], it follows directly that the problem in the case of IDs alone is PSPACE-complete in combined complexity; being such a case a particular case of NKCIDs, KDs and EDs (when no KD is defined, any ID is non-key-conflicting), PSPACE-hardness in our general case follows trivially.

Membership is proved by showing that the algorithm $\mathsf{Answer_S}$ runs in PSPACE. Consider a database schema $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ where Σ_I, Σ_K and Σ_E are sets of NKCIDs, KDs, and EDs, respectively. Our algorithm $\mathsf{Answer_S}$ proceeds as follows. The first step, checks whether $\mathcal{D}$ satisfies $\Sigma_K \cup \Sigma_E^*$. Checking if $\mathcal{D}$ satisfies Σ_K is clearly feasible in PTIME (and *a fortiori* in PSPACE). Analogously, given an ED $\delta \in \Sigma_E^*$, checking if $\mathcal{D}$ satisfies δ can be done in PTIME. From Theorem 5.2.7, it follows that checking wether $\delta \in \Sigma_E^*$ is a PSPACE complete problem. Then, the first step of the algorithm is computable in PSPACE. If $\mathcal{D}$ does not satisfy $\Sigma_K \cup \Sigma_E^*$, the answer is trivial; if it does, we disregard Σ_K and Σ_E, and apply the PSPACE algorithm Answer_{JK}. Hence, All steps of $\mathsf{Answer_S}$ are computable in PSPACE.

Membership in PTIME in data complexity follows immediately since Answer_{JK} runs in time polynomial in data complexity. □

Furthermore, the above complexity characterization of the problem holds even if we restrict to instances consistent with the key and the exclusion dependencies.

Corollary 5.2.18 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ be a NKC database schema, $\mathcal{D}$ a database instance for $\mathcal{RS}$ consistent with $\Sigma_K \cup \Sigma_E^*$, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. The problem of establishing whether $t \in ans_s(q, \mathcal{RS}, \mathcal{D})$ is PSPACE-complete with respect to combined complexity.*

Proof. Membership follows immediately from the general case treated in Theorem 5.2.17. Also hardness can be proved as done in Theorem 5.2.17. Indeed, as already said, from the results of [101] it immediately follows that the problem of query answering in the presence of IDs only is PSPACE-complete; moreover, the case of only IDs is a particular case of NKCIDs, KDs and EDs, in which the database instance is consistent with the KDs and the EDs that are logical consequence of the IDs and EDs originally issued over the schema. □

It should be easy to see that the above results also hold for the case in which only EDs and IDs are expressed over the relational schema, and for the case of only KDs and NKCIDs (proofs for these cases are analogous to the above proofs).

5.3 Query Answering under the Loose Semantics

In this section we analyze the problem of computing answers to queries under the loose semantics. In particular, since (as shown in Section 3.3) the loosely-complete and the strictly-complete

semantics coincide, we study query answering under the loosely-sound semantics and the loosely-exact semantics.

5.3.1 Query Answering under the Loosely-sound Semantics

As we already said, differently from the strictly-sound semantics, given a relational schema $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ and a database instance $\mathcal{D}$, it always holds that $sem_{ls}(\mathcal{RS}, \mathcal{D}) \neq \emptyset$, because we are now allowed to also eliminate facts from $\mathcal{D}$ in order to satisfy integrity constraints. Notice that, while to satisfy key and exclusion dependencies we are forced to delete facts from $\mathcal{D}$, we always try to satisfy inclusion dependencies by adding new facts, and resort to facts deletion also in this case only if there is no other chance to obtain a database consistent with $\mathcal{RS}$. This because databases in $sem_{ls}(\mathcal{RS}, \mathcal{D})$ are the ones that are "as sound as possible", thus we have to consider only databases consistent with the constraints that "minimize" elimination of facts from $\mathcal{D}$.

We first show undecidability of query answering for 1-key-conflicting relational schemas.

Theorem 5.3.1 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ be a 1KC relational schema, $\mathcal{D}$ a database instance for $\mathcal{RS}$, q a query of arity n over $\mathcal{RS}$, and t a n-tuple of constants of $\mathcal{U}_\mathcal{D}$. The problem of establishing whether $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ is undecidable.*

Proof. Consider the case in which $\mathcal{D}$ is consistent with $\Sigma_K \cup \Sigma_E^*$. Undecidability in this case follows from Corollary 5.2.15 since, in the case in which $\mathcal{D}$ is consistent with $\Sigma_K \cup \Sigma_E^*$, $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ iff $t \in ans_s(q, \mathcal{RS}, \mathcal{D})$. □

As for the class of non-key-conflicting relational schemas, we give a method for computing answers to a query q under the loosely-sound semantics that can be informally explained as follows: we first identify the maximal subsets of $\mathcal{D}$ that are consistent with $\Sigma_K \cup \Sigma_E^*$, then for each such database $\mathcal{D}'$ we make use of the algorithm $\mathsf{Answer_S}$ presented in Section 5.2.1. Indeed, it can be shown that a tuple t is a consistent answer to a query q with respect to $\mathcal{RS}$ and $\mathcal{D}$, i.e., $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$, iff $\mathsf{Answer_S}(\mathcal{RS}, \mathcal{D}', q, t)$ returns true for each such database $\mathcal{D}'$.

More specifically, we define the following algorithm:

Algorithm $\mathsf{Answer_{LS}}(\mathcal{RS}, \mathcal{D}, q, t)$
Input: non-key-conflicting relational schema $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$, database instance $\mathcal{D}$,
query q of arity n over $\mathcal{RS}$, n-tuple t of constants of $\mathcal{U}_\mathcal{D}$;
Output: *true* if $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$, *false* otherwise;
if there exists $\mathcal{D}_1 \subseteq \mathcal{D}$
such that
(1) $\mathcal{D}_1$ is consistent with $\Sigma_K \cup \Sigma_E^*$;
(2) **for each** $r(t) \in \mathcal{D} - \mathcal{D}_1$,
$\mathcal{D}_1 \cup \{r(t)\}$ is not consistent with $\Sigma_K \cup \Sigma_E^*$;
(3) $\mathsf{Answer_S}(\mathcal{RS}, \mathcal{D}_1, q, t)$ returns *false*
then return *false*
else return *true*

Informally, conditions (1) and (2) together check that $\mathcal{D}_1$ is a maximal subset of $\mathcal{D}$ consistent with $\Sigma_K \cup \Sigma_E^*$; this implies the existence of a database $\mathcal{B} \in sem_{ls}(\mathcal{RS}, \mathcal{D})$ such that $\mathcal{B} \cap \mathcal{D} = \mathcal{D}_1$. Then, condition (3) verifies that $t \notin q^{\mathcal{B}}$.

Theorem 5.3.2 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ be a NKC relational schema, $\mathcal{D}$ a database instance for $\mathcal{RS}$, q a query of arity n over $\mathcal{RS}$, and t a n-tuple of constants of $\mathcal{U}_\mathcal{D}$. Then, $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ iff $\mathsf{Answer}_{\mathsf{LS}}(\mathcal{RS}, \mathcal{D}, q, t)$ returns true.*

Proof. We first provide the following result that is needed in the proof

Lemma 5.3.3 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ be a NKC relational schema and $\mathcal{D}$ a database instance for $\mathcal{RS}$. Then, $sem_{ls}(\mathcal{RS}, \mathcal{D}) = \bigcup_{\mathcal{D}'} sem_s(\mathcal{RS}, \mathcal{D}')$ for each $\mathcal{D}'$ maximal subset of $\mathcal{D}$ consistent with $\Sigma_K \cup \Sigma_E^*$.*

Proof. Given $\mathcal{B} \in sem_{ls}(\mathcal{RS}, \mathcal{D})$, then $\mathcal{D}' = \mathcal{B} \cap \mathcal{D}$ is a maximal subset of $\mathcal{D}$ consistent with $\Sigma_K \cup \Sigma_E^*$. Indeed, since $\mathcal{B}$ satisfies $\Sigma_K \cup \Sigma_E^*$, then every subset of $\mathcal{B}$ satisfies such constraints, and if there would exist $\mathcal{D}'' \subseteq \mathcal{D}$ such that $\mathcal{D}''$ satisfies $\Sigma_K \cup \Sigma_E^*$ and $\mathcal{D}'' \supset \mathcal{D}'$, $\mathcal{B}$ did not belong to $sem_{ls}(\mathcal{RS}, \mathcal{D})$. It easily follows that $\mathcal{B} \in sem_s(\mathcal{RS}, \mathcal{D}')$, since $\mathcal{B}$ is consistent with $\mathcal{RS}$, and $\mathcal{B} \supseteq \mathcal{D}'$. Consider now $\mathcal{D}'$ maximal subset of $\mathcal{D}$ consistent with $\Sigma_K \cup \Sigma_E^*$. From Theorem 5.2.16 it follows that $sem_s(\mathcal{RS}, \mathcal{D}')$ is not empty. Hence, for each $\mathcal{B}' \in sem_s(\mathcal{RS}, \mathcal{D}')$ we have also that $\mathcal{B}' \in sem_{ls}(\mathcal{RS}, \mathcal{D})$. Indeed, $\mathcal{B}'$ is consistent with $\mathcal{RS}$, and is minimal w.r.t. $\leq_\mathcal{D}$. The last property can be easily verified by assuming that there exist $\mathcal{B}'' <_\mathcal{D} \mathcal{B}'$, i.e., $\mathcal{B}''$ is consistent with $\mathcal{RS}$ and $\mathcal{B}'' \cap \mathcal{D} \supset \mathcal{B}' \cap \mathcal{D}$. Since $\mathcal{B}' \cap \mathcal{D} \supset \mathcal{B}' \cap \mathcal{D}' = \mathcal{D}'$, this would imply $\mathcal{B}'' \cap \mathcal{D} \supset \mathcal{D}'$, thus contradicting the assumption. □

We now provide the proof of the theorem.

"⇒"If $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ then $t \in q^{\mathcal{B}}$ for each $\mathcal{B} \in sem_{ls}(\mathcal{RS}, \mathcal{D})$. From Lemma 5.3.3 it follows that $t \in ans_s(q, \mathcal{RS}, \mathcal{D}_1)$ for each $\mathcal{D}_1$ maximal subset of $\mathcal{D}$ consistent with $\Sigma_K \cup \Sigma_E^*$, and from soundness and completeness of algorithm $\mathsf{Answer}_{\mathsf{S}}$, it follows that $\mathsf{Answer}_{\mathsf{S}}(\mathcal{RS}, \mathcal{D}_1, q, t)$ returns *true* for each such database $\mathcal{D}_1$. Hence, $\mathsf{Answer}_{\mathsf{LS}}(\mathcal{RS}, \mathcal{D}, q, t)$ returns *true*.

"⇐"Suppose by contradiction that $t \notin ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ and $\mathsf{Answer}_{\mathsf{LS}}(\mathcal{RS}, \mathcal{D}, q, t)$ returns *true*. This means that for each $\mathcal{D}_1$ maximal subset of $\mathcal{D}$ consistent with $\Sigma_K \cup \Sigma_E^*$, $\mathsf{Answer}_{\mathsf{S}}(\mathcal{RS}, \mathcal{D}_1, q, t)$ returns *true*. From soundness and completeness of algorithm $\mathsf{Answer}_{\mathsf{S}}$, it follows that $t \in ans_s(q, \mathcal{RS}, \mathcal{D}_1)$ for each such database $\mathcal{D}_1$, i.e., $t \in q^{\mathcal{B}}$ for each $\mathcal{B} \in sem_s(\mathcal{RS}, \mathcal{D}_1)$. From Lemma 5.3.3 it follows that $t \in q^{\mathcal{B}}$ for each $\mathcal{B} \in sem_{ls}(\mathcal{RS}, \mathcal{D})$, but this contradicts the assumption. □

We give now the computational characterization of the problem of query answering under the loosely-sound semantics in the presence of NKCIDs, KDs, and EDs. We start by analyzing data complexity. For the sake of clarity, we first provide an upper bound for the complexity of the problem, and then study its hardness.

Lemma 5.3.4 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ be a NKC relational schema, $\mathcal{D}$ a database instance for $\mathcal{RS}$, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. The problem of establishing whether $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ is in coNP with respect to data complexity.*

Proof. Membership in coNP follows from the algorithm $\mathsf{Answer}_{\mathsf{LS}}(\mathcal{RS}, \mathcal{D}, q, t)$ and from Theorem 5.2.17. Indeed, in the algorithm the problem of establishing whether $t \notin ans_{ls}(q, \mathcal{RS}, \mathcal{D})$, that is the complement of our problem, is carried out by guessing a database and checking conditions (1), (2), and (3) that can be verified in polynomial time. □

It is easy to see that the above complexity characterization holds also in the absence of IDs on the relational schema $\mathcal{RS}$, i.e., when $\Sigma_I = \emptyset$, and in all the cases in which at least one between Σ_K or Σ_E is not empty. It is also worth noticing that query answering in the presence of only IDs over $\mathcal{RS}$ under the loosely-sound semantics is analogous to the same problem under the strictly-sound semantics, which has been addressed in the previous section.

Let us now turn our attention to lower bounds. We first consider the case of only KDs.

Theorem 5.3.5 *Let $\mathcal{RS} = \langle \Psi, \Sigma_K \rangle$ be a relational schema, $\mathcal{D}$ a database instance for $\mathcal{RS}$, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. The problem of establishing whether $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ is coNP-hard with respect to data complexity.*

Proof. In order to prove the hardness, we reduce the 3-colorability problem to the complement of our problem.

Consider a graph $G = \langle V, E \rangle$ with a set of vertices V and edges E. We define a relational schema $\mathcal{RS} = \langle \Psi, \Sigma_K \rangle$ where Ψ consists of the two binary relations *edge* and *col*, and Σ_K contains the dependency $key(col) = \{1\}$. The instance $\mathcal{D}$ is defined as follows:

$$\begin{aligned} \mathcal{D} \quad = \quad & \{col(n, i) | i \in \{1, 2, 3\} \text{ and } n \in V\} \cup \\ & \{edge(x, y) | \langle x, y \rangle \in E\} \end{aligned}$$

Finally, we define the query

$$q \leftarrow edge(X, Y), col(X, Z), col(Y, Z)$$

We prove that G is 3-colorable (i.e., for each pair of adjacent vertices, the vertices are associated with different colors) if and only if $\langle\rangle \notin ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ (i.e., the boolean query q has a negative answer)[6]. In fact, it is immediate to verify that, for each possible coloring C of the graph (i.e., a set of pairs of vertices and colors, where the three colors are represented by the values 1,2,3) there exists $\mathcal{B} \in sem_{ls}(\mathcal{RS}, \mathcal{D})$ that exactly corresponds to C, i.e., $col^\mathcal{B}$ is exactly the set of pairs in the coloring C. Therefore, if there exists a coloring that is a 3-coloring, then $\langle\rangle \notin q^\mathcal{B}$ for some $\mathcal{B} \in sem_{ls}(\mathcal{RS}, \mathcal{D})$, consequently $\langle\rangle \notin ans_{ls}(q, \mathcal{RS}, \mathcal{D})$. Conversely, it is immediate to verify that, for each $\mathcal{B} \in sem_{ls}(\mathcal{RS}, \mathcal{D})$, $col^{\mathcal{B} \cap \mathcal{D}}$ corresponds to a possible coloring of the graph. Hence, if each possible coloring is not a 3-coloring, then $\langle\rangle \in q^\mathcal{B}$, therefore $\langle\rangle \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$. □

An analogous result holds also for schemas containing only EDs.

Theorem 5.3.6 *Let $\mathcal{RS} = \langle \Psi, \Sigma_E \rangle$ be a relational schema, $\mathcal{D}$ a database instance for $\mathcal{RS}$, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. The problem of establishing whether $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ is coNP-hard with respect to data complexity.*

[6] $\langle\rangle$ represents the empty tuple. According to a common notation in relational databases [2], given a boolean query q, $\langle\rangle \in q^\mathcal{B}$ iff q evaluates to true over a database instance $\mathcal{B}$.

Proof. Also in this case we prove coNP hardness by reducing the 3-colorability problem to the complement our problem. Consider a graph $G = \langle V, E\rangle$ with a set of vertices V and edges E. We define a relational schema $\mathcal{RS} = \langle \Psi, \Sigma_E\rangle$ where Ψ consists of the binary relation *edge* and the three unary relations *col1*, *col2* and *col3*. Then, Σ_E contains the dependencies

$$\begin{aligned} col1[1] \quad &\cap \quad col2[1] = \emptyset; \\ col1[1] \quad &\cap \quad col3[1] = \emptyset; \\ col2[1] \quad &\cap \quad col3[1] = \emptyset. \end{aligned}$$

The instance $\mathcal{D}$ is defined as follows:

$$\begin{aligned} \mathcal{D} \quad = \quad & \{col1(n), col2(n), col3(n) | n \in V\} \cup \\ & \{edge(x, y) | \langle x, y\rangle \in E\} \end{aligned}$$

Finally, we define the query

$$\begin{aligned} q \quad \leftarrow \quad & edge(X, Y), col1(X), col1(Y) \ \vee \\ & edge(X, Y), col2(X), col2(Y) \ \vee \\ & edge(X, Y), col3(X), col3(Y). \end{aligned}$$

It is easy to see that, also in this case, G is 3-colorable if and only if $\langle\rangle \notin ans_{ls}(q, \mathcal{RS}, \mathcal{D})$. Indeed, for each possible coloring C of the graph there exists $\mathcal{B} \in sem_{ls}(\mathcal{RS}, \mathcal{D})$ that exactly corresponds to C, i.e., $col1^{\mathcal{B}}$, $col2^{\mathcal{B}}$ and $col3^{\mathcal{B}}$ store exactly the set of nodes associated in the coloring C with color 1, 2 and 3, respectively. Therefore, if there exists a coloring that is a 3-coloring, then $\langle\rangle \notin q^{\mathcal{B}}$ for some $\mathcal{B} \in sem_{ls}(\mathcal{RS}, \mathcal{D})$, consequently $\langle\rangle \notin ans_{ls}(q, \mathcal{RS}, \mathcal{D})$. Conversely, it is immediate to verify that, for each $\mathcal{B} \in sem_{ls}(\mathcal{RS}, \mathcal{D})$, $col1^{\mathcal{B}\cap\mathcal{D}}$, $col2^{\mathcal{B}\cap\mathcal{D}}$ and $col3^{\mathcal{B}\cap\mathcal{D}}$ correspond to a possible coloring of the graph. Hence, if each possible coloring is not a 3-coloring, then $\langle\rangle \in q^{\mathcal{B}}$, therefore $\langle\rangle \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$. □

It should be easy to see that whenever KDs or EDs are specified on the relational schema, the problem of query answering is coNP-complete with respect data complexity. In particular, we can provide the following theorem.

Theorem 5.3.7 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E\rangle$ be a NKC relational schema, $\mathcal{D}$ a database instance for $\mathcal{RS}$, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_{\mathcal{D}}$. The problem of establishing whether $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ is coNP-complete with respect to data complexity.*

Proof. The claim follows from Lemma 5.3.4, and from Theorem 5.3.5 or Theorem 5.3.6 □

Let us now deal with combined complexity. We first consider query answering in the absence of IDs, and then address the general setting.

Theorem 5.3.8 *Let $\mathcal{RS} = \langle \Psi, \Sigma\rangle$ be a relational schema, where either $\Sigma = \Sigma_K$ or $\Sigma = \Sigma_E$ (or $\Sigma = \Sigma_K \cup \Sigma_E$). Moreover, let $\mathcal{D}$ be a database instance for $\mathcal{RS}$, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_{\mathcal{D}}$. The problem of establishing whether $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ is Π_2^p-complete with respect to combined complexity.*

Proof. To prove membership in Π_2^p, consider the algorithm $\mathsf{Answer_{LS}}(\mathcal{RS}, \mathcal{D}, q, t)$. Since there are no IDs specified over $\mathcal{RS}$, the algorithm $\mathsf{Answer_S}(\mathcal{RS}, \mathcal{D}_1, q, t)$ in condition (3) simply verifies if $t \notin q^{\mathcal{D}_1}$, i.e., it performs classical query answering in the absence of integrity constraints. Since classical query processing over relational databases is combined complete for NP, the claim easily follows. Notice that the above proof holds either if $\Sigma = \Sigma_K$ or if $\Sigma = \Sigma_E$ (or if $\Sigma = \Sigma_K \cup \Sigma_E$).

Hardness can be proved by a reduction to our problem of the 2-QBF validity problem, i.e., the validity problem for quantified boolean formulae. Two different reductions can be given in the two cases in which $\Sigma = \Sigma_K$ or $\Sigma = \Sigma_E$.

Consider first the case in which $\Sigma = \Sigma_K$. We prove hardness with respect to Π_2^p, by reducing to our problem the 2-QBF validity problem, i.e., the validity problem for quantified boolean formulae.

Let g be the 2-QBF formula $\forall x_1 \ldots x_n \exists y_1 \ldots y_m f(x_1, \ldots, x_n, y_1, \ldots, y_m)$ where f is a 3-CNF, i.e., a propositional formula of the form

$$(l_{1,1} \vee l_{1,2} \vee l_{1,3}) \wedge \ldots \wedge (l_{k,1} \vee l_{k,2} \vee l_{k,3}). \tag{5.1}$$

where each $l_{i,j}$ is a literal of the form a or $\overline{a}$, with $a \in \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$.

We construct a relational schema $\mathcal{RS} = \langle \Psi, \Sigma_K \rangle$ as follows:

- Ψ consists of the relations (with associated arities): $x_i/2$ for $1 \leq i \leq n$, $y_i/1$ for $1 \leq i \leq m$, and $c_i/3$ for $1 \leq i \leq k$;
- Σ_K consists of the key dependencies $key(x_i) = \{1\}$ for $1 \leq i \leq n$.

Moreover, we build an instance $\mathcal{D}$ for $\mathcal{RS}$, containing the following facts:

1. for each i such that $1 \leq i \leq n$, the facts $x_i(c_0, 0), x_i(c_0, 1)$;
2. for each i such that $1 \leq i \leq m$, the facts $y_i(0), y_i(1)$;
3. for each i such that $1 \leq i \leq k$ the facts of the form $c_i(b_{i,1}, b_{i,2}, b_{i,3})$ such that $b_{i,j} \in \{0, 1\}$ for $1 \leq j \leq 3$, and when applying the substitution $\sigma\{l_{i,1} \to b_{i,1}, l_{i,2} \to b_{i,2}, l_{i,3} \to b_{i,3}\}$ to f, the conjunct $l_{i,1} \vee l_{i,2} \vee l_{i,3}$ evaluates to 1, i.e., to *true*.

Finally, let q be the query

$$q \leftarrow x_1(C_1, X_1), \ldots x_n(C_n, X_n), y_1(Y_1) \ldots y_m(Y_m), c_1(Z_{1,1}, Z_{1,2}, Z_{1,3}), \ldots c_k(Z_{k,1}, Z_{k,2}, Z_{k,3})$$

where either

(i) $Z_{i,j} = X_r$ if $l_{i,j} = X_r$ or $l_{i,j} = \overline{X}_r$ for $r \in \{1, \ldots n\}$, or

(ii) $Z_{i,j} = Y_r$ if $l_{i,j} = Y_r$ or $l_{i,j} = \overline{Y}_r$ for $r \in \{1, \ldots m\}$.

We prove that g is valid if and only if $\langle\rangle \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$.

First, suppose that g is valid. Then, for each propositional evaluation μ of $x_1, \ldots, x_n$ there exists a propositional evaluation μ' of $y_1, \ldots, y_m$ such that the interpretation of f with respect to μ and μ' is true. Let $\mathcal{B}_\mu$ be the database obtained from $\mathcal{D}$ by removing each fact of the form $x_i(c_0, 1)$

if $\mu(x_i) = \mathit{false}$ and $x_i(c_0, 0)$ if $\mu(x_i) = \mathit{true}$. It is immediate to see that $\mathcal{B}_\mu \in \mathit{sem}_{ls}(\mathcal{RS}, \mathcal{D})$. Now, since the interpretation of f with respect to μ and μ' is true, it follows that each conjunct $l_{i,1} \vee l_{i,2} \vee l_{i,2}$ in f evaluates to true. Then, according to the construction of $\mathcal{D}$ (point 3), for $1 \leq i \leq n$, $\mathcal{B}_\mu$ contains the facts $c_i(b_{i,1}, b_{i,2}, b_{i,3})$, where $b_{i,j} = 1$ if $\nu(l_{i,j}) = \mathit{true}$, or $b_{i,j} = 0$ if $\nu(l_{i,j}) = \mathit{false}$, where $\nu = \mu \cup \mu'$. It should be easy to see that the query q evaluates to true over $\mathcal{B}_\mu$, i.e., $\langle\rangle \in q^{\mathcal{B}_\mu}$. Consider now the set $\{\mu_1, \ldots, \mu_h\}$ of possible propositional evaluations of $x_1, \ldots, x_n$ in g, then each $\mathcal{B} \in \mathit{sem}_{ls}(\mathcal{RS}, \mathcal{D})$ is such that $\mathcal{B} \supseteq \mathcal{B}_{\mu_i}$, for some $i \in \{1, \ldots, h\}$. It follows that $\langle\rangle \in \mathit{ans}_{ls}(q, \mathcal{RS}, \mathcal{D})$.

Suppose now that g is not valid. Therefore, there exists an evaluation μ of $x_1, \ldots, x_n$ such that for each evaluation μ' of $y_1, \ldots, y_m$, the interpretation of f with respect to μ and μ' is false. Let $\mathcal{B}$ be the database obtained from $\mathcal{D}$ by removing each fact of the form $x_i(c_0, 1)$ if $\mu(x_i) = \mathit{false}$ and $x_i(c_0, 0)$ if $\mu(x_i) = \mathit{true}$ It is immediate to see that $\mathcal{B} \in \mathit{sem}_{ls}(\mathcal{RS}, \mathcal{D})$, since it corresponds to a maximal subset of $\mathcal{D}$ consistent with Σ_K. Since g is not valid, then at least one conjunct in f is not satisfied for each evaluation μ' of $y_1, \ldots, y_m$. Hence, in the evaluation of the query q over $\mathcal{B}$, at least one atom of the form $c_i(Z_{i,1}, Z_{i,2}, Z_{i,3})$ does not unify with the facts in $\mathcal{B}$. Therefore, $\langle\rangle \notin q^{\mathcal{B}}$, which implies that $\langle\rangle \notin \mathit{ans}_{le}(q, \mathcal{RS}, \mathcal{D})$.

Consider now the case in which $\Sigma = \Sigma_E$. To prove Π_2^p-hardness, we use a reduction from 2-QBF validity problem similar to the one described above. Indeed, in this case we construct a relational schema $\mathcal{RS} = \langle \Psi, \Sigma_E \rangle$ as follows:

- Ψ is as in the above reduction;
- Σ_E consists of the exclusion dependencies $x_i[1] \cap x_i[2] = \emptyset$ for $1 \leq i \leq n$.

The instance $\mathcal{D}$ for $\mathcal{RS}$, contains the following facts:

1. for each i such that $1 \leq i \leq n$, the facts $x_i(1, 0), x_i(0, 1)$;
2. the same facts indicated at point 2 and 3 in the above reduction.

Finally, the query q is the same as above.

Proceeding as in the previous reduction, we can analogously prove that g is valid if and only if $\langle\rangle \in \mathit{ans}_{ls}(q, \mathcal{RS}, \mathcal{D})$.

□

As for the general case, we prove in the following that the presence of IDs significantly complicate the problem.

Theorem 5.3.9 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ be a NKC relational schema, $\mathcal{D}$ a database instance for $\mathcal{RS}$, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. The problem of establishing whether $t \in \mathit{ans}_{ls}(q, \mathcal{RS}, \mathcal{D})$ is PSPACE-complete with respect to combined complexity.*

Proof. Hardness follows from Corollary 5.2.18 and from the fact that, for NKC schemas when $\mathcal{D}$ is consistent with $\Sigma_K \cup \Sigma_E^*$, $t \in \mathit{ans}_{ls}(q, \mathcal{RS}, \mathcal{D})$ if and only if $t \in \mathit{ans}_s(q, \mathcal{RS}, \mathcal{D})$.

Membership in PSPACE follows from algorithm $\mathsf{Answer}_{\mathsf{LS}}(\mathcal{RS}, \mathcal{D}, q, t)$ and Theorem 5.2.17. Indeed, it is easy to see that conditions (1), (2), and (3) can be verified in polynomial space.

Then, by Savitch's theorem [146] $\mathsf{Answer}_{\mathsf{LS}}$ can be transformed into a deterministic algorithm that runs in polynomial space. □

Notice that, since PSPACE-completeness of query answering under the strictly-sound semantics also holds in the presence of only IDs and EDs, and only IDs and NKCIDs on the relational schema, as said at end of Section 5.2.1, an analogous result holds for NKC schemas of the form $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K \rangle$ and of the form $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_E \rangle$.

5.3.2 Query Answering under the Loosely-exact Semantics

We now study the query answering problem under the loosely-exact semantics. We recall that, differently from the loosely-sound semantics, in this case IDs can be always satisfied by either adding or deleting facts. Hence, $sem_{le}(\mathcal{RS}, \mathcal{D})$ accounts for databases that minimize both elimination and insertion of facts, i.e., that are "as exact as possible".

We first prove that query answering under the loosely-exact semantics is undecidable in the general case, i.e., when no restriction is imposed on the form of IDs, KDs and EDs.

Theorem 5.3.10 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ be a relational schema, $\mathcal{D}$ a database instance for $\mathcal{RS}$, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. The problem of establishing whether $t \in ans_{le}(q, \mathcal{RS}, \mathcal{D})$ is undecidable.*

Proof. We reduce query answering in the loosely-sound semantics to query answering in the loosely-exact semantics. We can restrict our attention to schemas without EDs. Furthermore, we can consider instances $\mathcal{D}$ consistent with Σ_K, since for this class of instances the problem of establishing whether $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ is undecidable (see end of Section 5.2.1). Starting from such a problem instance $(\mathcal{RS}, \mathcal{D}, q, t)$, we define a new problem instance $(\mathcal{RS}', \mathcal{D}', q', t')$ such that $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ iff $t' \in ans_{le}(q', \mathcal{RS}', \mathcal{D}')$. Precisely:

- $\mathcal{RS}' = \langle \Psi', \Sigma'_I, \Sigma'_K \rangle$ is obtained from $\mathcal{RS}$ by:
  - defining Ψ' as the schema obtained from Ψ by adding an attribute to each relation in Ψ (in the last position);
  - changing each inclusion in order to propagate such a new attribute from r to s, i.e., Σ'_I is obtained from Σ_I by replacing each $I = r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k]$ with $I' = r[i_1, \ldots, i_k, n] \subseteq s[j_1, \ldots, j_k, m]$, where n is the arity of r in Ψ' and m is the arity of s in Ψ';
- $\mathcal{D}'$ is the set $\mathcal{D}'_1 \cup \mathcal{D}'_2$, where $\mathcal{D}'_1 = \{\ r(\overline{u}, t_0) | r(\overline{u}) \in \mathcal{D}\ \}$ and

$$\mathcal{D}'_2 = \{\ r(\overline{u}, t_1) \mid r \in \Psi \text{ and } \overline{u} \text{ is a tuple of constants of } \mathcal{U}_\mathcal{D} \cup \{t_1\}\ \}$$

  where t_0, t_1 are constants of $\mathcal{U}$ not belonging to $\mathcal{U}_\mathcal{D}$. Notice that the set $\mathcal{D}'$ is finite;
- if the query q has the form

$$q(\vec{\mathbf{x}}) \leftarrow conj_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1) \vee \cdots \vee conj_k(\vec{\mathbf{x}}, \vec{\mathbf{y}}_k)$$

the query q' is as follows:

$$q'(\vec{\mathbf{x}}, Y) \leftarrow conj_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1, t_0) \vee \cdots \vee conj_k(\vec{\mathbf{x}}, \vec{\mathbf{y}}_k, t_0) \vee body'$$

where $body'$ is the disjunction

$$\bigvee \{r(\overline{u}, t_1) \mid r(\overline{u}) \in \mathcal{D} \text{ and there is a KD for } r \text{ in } \Sigma_K\}$$

- t' is obtained from t by adding the value t_0 at the end of the tuple t.

It can be shown that $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ iff $t' \in ans_{le}(q', \mathcal{RS}', \mathcal{D}')$, since for each database $\mathcal{B}$ in $sem_{le}(\mathcal{RS}, \mathcal{D})$, there are two possible cases:

1. $\mathcal{B} \cap \mathcal{D} = \mathcal{D}$. In this case, due to the key dependencies Σ_K, $\mathcal{B}$ does not contain any tuple of the form $r(\overline{u}, t_1)$ such that $r(\overline{u}) \in \mathcal{D}$ and a key dependency for r is defined in Σ_K. Consequently, $t' \in q'^{\mathcal{B}}$ iff $t \in q^{\mathcal{B}}$. Moreover, it is immediate to verify that there exists at least one such $\mathcal{B}$ in $sem_{le}(\mathcal{RS}', \mathcal{D}')$;

2. $\mathcal{B} \cap \mathcal{D} \subset \mathcal{D}$. In this case, there exists at least one tuple in $\mathcal{B}$ of the form $r(\overline{u}, t_1)$ such that $r(\overline{u}) \in \mathcal{D}$ and a key dependency for r is defined in Σ_K, consequently $t' \in q'^{\mathcal{B}}$ for each such $\mathcal{B}$. In other words, this kind of databases does not affect $ans_{le}(q', \mathcal{RS}', \mathcal{D}')$, since in $\mathcal{B}$ every possible tuple is in the answer of q'.

Therefore, $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ iff $t' \in ans_{le}(q', \mathcal{RS}', \mathcal{D}')$.

Finally, since the above reduction is effectively computable and since, by Theorem 5.3.1, establishing whether $t \in ans_{ls}(q, \mathcal{RS}, \mathcal{D})$ is undecidable, the thesis follows. □

Differently from the previous semantics, in the case when the instance $\mathcal{D}$ is consistent with $\Sigma_K \cup \Sigma_E^*$, we obtain a surprising result: query answering is decidable under the loosely-exact semantics even without any restriction on the form of KDs and IDs.

Theorem 5.3.11 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ be a relational schema, $\mathcal{D}$ a database instance consistent with $\Sigma_K \cup \Sigma_E^*$, q a query of arity n over $\mathcal{RS}$, and t be an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. The problem of establishing whether $t \in ans_{le}(q, \mathcal{RS}, \mathcal{D})$ can be decided in polynomial time with respect to data complexity and is NP-complete with respect to combined complexity.*

Proof. To prove the thesis, we define the following algorithm:

Algorithm $\mathsf{AnswerCons_{LE}}(\mathcal{RS}, \mathcal{D}, q, t)$
Input: relational schema $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$,
database instance $\mathcal{D}$ consistent with $\Sigma_K \cup \Sigma_E^*$,
conjunctive query q of arity n, n-tuple t of constants of $\mathcal{U}_\mathcal{D}$
Output: *true* if $t \in ans_{le}(q, \mathcal{RS}, \mathcal{D})$, *false* otherwise
$\mathcal{D}_1 = \mathcal{D}$;
repeat
  $\mathcal{D}_0 = \mathcal{D}_1$;
  for each $r(t') \in \mathcal{D}_1$

if there exists $r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k] \in \Sigma_I$
such that
for each $s(t'') \in \mathcal{D}_1$, $t''[j_1, \ldots, j_k] \neq t'[i_1, \ldots, i_k]$
then $\mathcal{D}_1 = \mathcal{D}_1 - \{r(t')\}$
until $\mathcal{D}_1 = \mathcal{D}_0$;
if $t \in q^{\mathcal{D}_1}$
then return *true*
else return *false*

Correctness of the algorithm $\mathsf{AnswerCons_{LE}}$ follows from the fact that the database $\mathcal{D}_1$ computed by the algorithm is such that *(i)* $\mathcal{D}_1 \in sem_{le}(\mathcal{RS}, \mathcal{D})$; *(ii)* for each $\mathcal{B} \in sem_{le}(\mathcal{RS}, \mathcal{D})$, $\mathcal{B} \supseteq \mathcal{D}_1$. Therefore, $t \in ans_{le}(q, \mathcal{RS}, \mathcal{D})$ if and only if $t \in q^{\mathcal{D}_1}$. It is well-known that this last condition (corresponding to standard query answering over a relational database) can be computed in polynomial time with respect to data complexity and in nondeterministic polynomial time with respect to combined complexity. □

Let us turn our attention on query answering under the loosely-exact semantics in the case of NKC relational schemas. To this aim, we first define a particular query $Q(\delta_i, t)$ associated with a tuple t and an inclusion dependency δ_i.

Definition 5.3.12 Let δ_i be an inclusion dependency of the form $r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k]$, where r has arity n and s has arity m, and let t be an n-tuple. We denote as $Q(\delta_i, t)$ the boolean conjunctive query $q \leftarrow s(z_1, \ldots, z_m)$, where, for each ℓ such that $1 \leq \ell \leq m$, each z_ℓ is as follows: if there exists h such that $\ell = j_h$ then $z_\ell = t[i_h]$, otherwise $z_\ell = X_\ell$.

In the following, the query $Q(\delta_i, t)$ is used in order to verify whether a relational schema $\mathcal{RS}$ and an instance $\mathcal{D}$ imply the existence in all databases $\mathcal{B} \in sem_s(\mathcal{RS}, \mathcal{D})$ of a fact of the form $s(t')$ such that $t[i_1, \ldots, i_k] = t'[j_1, \ldots, j_k]$.

Below we define the algorithm $\mathsf{Answer_{LE}}$ for query answering under the loosely-exact semantics.

Algorithm $\mathsf{Answer_{LE}}(\mathcal{RS}, \mathcal{D}, q, t)$
Input: NKC relational schema $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$, database instance $\mathcal{D}$,
query q of arity n over $\mathcal{RS}$, n-tuple t of constants of $\mathcal{U}_\mathcal{D}$
Output: *true* if $t \in ans_{le}(q, \mathcal{RS}, \mathcal{D})$, *false* otherwise
if there exists $\mathcal{D}' \subseteq \mathcal{D}$ **such that**
(a) $\mathcal{D}'$ is consistent with $\Sigma_K \cup \Sigma_E^*$ **and**
(b) $\mathsf{Answer_S}(\langle \Psi, \Sigma_I \rangle, \mathcal{D}', q, t)$ returns *false* **and**
(c) **for each** $\mathcal{D}''$ such that $\mathcal{D}' \subset \mathcal{D}'' \subseteq \mathcal{D}$
(c1) $\mathcal{D}''$ is not consistent with $\Sigma_K \cup \Sigma_E^*$ **or**
(c2) **there exists** $\delta_i \in \Sigma_I$ **and** $r(t_1) \in \mathcal{D}''$
such that
$\mathsf{Answer_S}(\langle \Psi, \Sigma_I \rangle, \mathcal{D}', Q(\delta_i, t_1), \langle\rangle)$ returns *false* **and**
$\mathsf{Answer_S}(\langle \Psi, \emptyset \rangle, \mathcal{D}'', Q(\delta_i, t_1), \langle\rangle)$ returns *false*

then return *false*
else return *true*

Intuitively, to return *false* the algorithm looks for the existence of a database $\mathcal{B}'$ in $sem_{le}(\mathcal{RS}, \mathcal{D})$ such that $t \notin q^{\mathcal{B}'}$. As in the algorithm $\mathsf{Answer_{LS}}$, the database $\mathcal{B}'$ is represented by its intersection with the initial instance $\mathcal{D}$ (denoted as $\mathcal{D}'$ in the algorithm): the fact that $t \notin q^{\mathcal{B}'}$ is verified by condition (b), while the fact that $\mathcal{B}' \in sem_{le}(\mathcal{RS}, \mathcal{D})$ is verified by conditions (a) and (c) of the algorithm. In particular, condition (c) verifies that, for each database $\mathcal{B}''$ (represented by its intersection with $\mathcal{D}$ denoted as $\mathcal{D}''$), it is not the case that $\mathcal{B}'' <_{\mathcal{D}} \mathcal{B}'$. In conditions (c1) and (c2), the symbol $\langle\rangle$ denotes the empty tuple.

Soundness and completeness of the algorithm is established by the following theorem.

Theorem 5.3.13 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ be a NKC relational schema, $\mathcal{D}$ a database instance, q a query of arity n over Ψ, and t an n-tuple of constants of $\mathcal{U}_{\mathcal{D}}$. Then, $t \in ans_{le}(q, \mathcal{RS}, \mathcal{D})$ iff $\mathsf{Answer_{LE}}(\mathcal{RS}, \mathcal{D}, q, t)$ returns true.*

Proof. In order to prove correctness of the above algorithm, we need a preliminary lemma. In the following, given an instance $\mathcal{D}$ of a relational schema $\mathcal{RS} = \langle \Psi, \Sigma_I \rangle$, we denote as $chase_1(\mathcal{RS}, \mathcal{D})$ the set of new facts obtained by applying the chase rule to the facts in $\mathcal{D}$, i.e., the set of facts of the form $s(t_2)$ such that there exist $\delta_i = r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k] \in \Sigma_I$ and $r(t_1) \in \mathcal{D}$ such that $t_1[i_1, \ldots, i_k] = t_2[j_1, \ldots, j_k]$ and there exists no $s(t_3) \in \mathcal{D}$ such that $t_1[i_1, \ldots, i_k] = t_3[j_1, \ldots, j_k]$.

Lemma 5.3.14 *Let $\mathcal{D}', \mathcal{D}''$ be instances of a relational schema $\mathcal{RS} = \langle \Psi, \Sigma_I \rangle$ such that $\mathcal{D}' \subset \mathcal{D}''$ and, for each $\delta_i \in \Sigma_I$ of the form $\delta_i = r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k]$ and for each $r(t_1) \in \mathcal{D}''$, either $\mathsf{Answer_S}(\langle \Psi, \Sigma_I \rangle, \mathcal{D}', Q(\delta_i, t_1), \langle\rangle)$ returns true or $\mathsf{Answer_S}(\langle \Psi, \emptyset \rangle, \mathcal{D}'', Q(\delta_i, t_1), \langle\rangle)$ returns true. Then, $chase(\mathcal{RS}, \mathcal{D}'') - \mathcal{D}'' \subseteq chase(\mathcal{RS}, \mathcal{D}') - \mathcal{D}'$.*

Proof. It is straightforward to verify that the hypothesis implies that $chase_1(\mathcal{RS}, \mathcal{D}'') \subseteq chase_1(\mathcal{RS}, \mathcal{D}')$; this in turn implies that each new fact added in $chase(\mathcal{RS}, \mathcal{D}'')$ by an application of the chase rule in $chase(\mathcal{RS}, \mathcal{D}'')$ is also added by the chase rule in $chase(\mathcal{RS}, \mathcal{D}')$. Consequently, the thesis follows. □

We now prove the theorem.

"⇒"Suppose $\mathsf{Answer_{LE}}(\mathcal{RS}, \mathcal{D}, q, t)$ returns *false*. Then, there exists $\mathcal{D}' \subseteq \mathcal{D}$ such that conditions (a), (b) and (c) of the algorithm hold for $\mathcal{D}'$. Let $\mathcal{B}' = chase(\mathcal{RS}, \mathcal{D}')$. Now, suppose $\mathcal{B}' \notin sem_{le}(\mathcal{RS}, \mathcal{D})$: hence, there exists a database instance $\mathcal{B}''$ such that $\mathcal{B}''$ is consistent with Σ_K and $\mathcal{B}'' <_{\mathcal{D}} \mathcal{B}'$, which implies that $\mathcal{B}'' - \mathcal{D} \subseteq \mathcal{B}' - \mathcal{D}$. Since by hypothesis condition (c) holds for $\mathcal{D}'$, it follows that condition (c2) holds for $\mathcal{D}''$, i.e., there exists a fact $r(t_1) \in \mathcal{D}''$ and an inclusion $\delta_i = r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k] \in \Sigma_I$ such that:

1. $\mathsf{Answer_S}(\langle \Psi, \Sigma_I \rangle, \mathcal{D}', Q(\delta_i, t_1), \langle\rangle)$ returns *false*, which implies that there is no fact in $\mathcal{B}'$ of the form $s(t_2)$ such that $t_1[i_1, \ldots, i_k] = t_2[j_1, \ldots, j_k]$;
2. $\mathsf{Answer_S}(\langle \Psi, \emptyset \rangle, \mathcal{D}'', Q(\delta_i, t_1), \langle\rangle)$ returns *false*, which implies that there is no fact in $\mathcal{D}''$ of the form $s(t_2)$ such that $t_1[i_1, \ldots, i_k] = t_2[j_1, \ldots, j_k]$. On the other hand, a fact of the form $s(t_2)$ such that $t_1[i_1, \ldots, i_k] = t_2[j_1, \ldots, j_k]$ must be present in $\mathcal{B}''$, due to the presence of $r(t_1)$ in $\mathcal{D}''$ and to the inclusion δ_i.

The two above conditions imply that there exists a fact of the form $s(t_2)$ in $\mathcal{B}'' - \mathcal{D}$ which does not belong to $\mathcal{B}'$. Consequently, $\mathcal{B}'' - \mathcal{D} \subseteq \mathcal{B}' - \mathcal{D}$ does not hold, thus contradicting the hypothesis that $\mathcal{B}'' <_{\mathcal{D}} \mathcal{B}'$. Therefore, $\mathcal{B}' \in sem_{le}(\mathcal{RS}, \mathcal{D})$, and since conditions (a) and (b) hold for $\mathcal{D}'$, it follows that $t \notin q^{\mathcal{B}'}$, hence $t \notin ans_{le}(q, \mathcal{RS}, \mathcal{D})$.

"$\Leftarrow$"Suppose $t \notin ans_{le}(q, \mathcal{RS}, \mathcal{D})$. Therefore, there exists $\mathcal{B}' \in sem_{le}(\mathcal{RS}, \mathcal{D})$ such that $t \notin q^{\mathcal{B}'}$. Let $\mathcal{D}' = \mathcal{D} \cap \mathcal{B}'$. Since $\mathcal{B}' \in sem_{le}(\mathcal{RS}, \mathcal{D})$, then $\mathcal{B}'$ satisfies $\Sigma_I \cup \Sigma_E^*$, and since $\mathcal{D}' \subseteq \mathcal{B}'$, condition (a) of the algorithm holds. From $t \notin q^{\mathcal{B}'}$ and from soundness and completeness of the algorithm $\mathsf{Answer_S}$ it follows that condition (b) holds for $\mathcal{D}'$. Now, suppose condition (c) does not hold for $\mathcal{D}'$: then, there exists $\mathcal{D}''$ such that conditions (c1) and (c2) do not hold for $\mathcal{D}'$ and $\mathcal{D}''$, i.e., $\mathcal{D}''$ is consistent with $\Sigma_K \cup \Sigma_E^*$ and, for each $\delta_i \in \Sigma_I$ of the form $\delta_i = r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k]$ and for each $r(t_1) \in \mathcal{D}''$, either $\mathsf{Answer_S}(\langle \Psi, \Sigma_I \rangle, \mathcal{D}', Q(\delta_i, t_1), \langle\rangle)$ returns *true* or $\mathsf{Answer_S}(\langle \Psi, \emptyset \rangle, \mathcal{D}'', Q(\delta_i, t_1), \langle\rangle)$ returns *true*. By Lemma 5.3.14, it follows that $chase(\mathcal{RS}, \mathcal{D}'') - \mathcal{D}'' \subseteq chase(\mathcal{RS}, \mathcal{D}') - \mathcal{D}'$. Now let $\mathcal{B}'' = chase(\mathcal{RS}, \mathcal{D}'')$: since $\mathcal{B}' \supseteq chase(\mathcal{RS}, \mathcal{D}')$, it follows that $\mathcal{B}'' - \mathcal{D} \subseteq \mathcal{B}' - \mathcal{D}$, and by hypothesis $\mathcal{D}'' \supset \mathcal{D}'$, therefore $\mathcal{B}'' \cap \mathcal{D} \supset \mathcal{B}' \cap \mathcal{D}$, hence $\mathcal{B}'' <_{\mathcal{D}} \mathcal{B}'$. Moreover, since $\mathcal{D}''$ is consistent with $\Sigma_K \cup \Sigma_E^*$, $\mathcal{B}''$ is consistent with Σ_K, Σ_E and Σ_I, consequently $\mathcal{B}' \notin sem_{le}(\mathcal{RS}, \mathcal{D})$, thus contradicting the hypothesis. Therefore, condition (c) holds for $\mathcal{D}'$, which implies that $\mathsf{Answer_{LE}}(\mathcal{RS}, \mathcal{D}, q, t)$ returns *false*. □

Finally, based on the above algorithm, we analyze the computational complexity of query answering under the loosely-exact semantics for NKC relational schemas.

Theorem 5.3.15 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ be a NKC relational schema, $\mathcal{D}$ a database instance, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_{\mathcal{D}}$. The problem of establishing whether $t \in ans_{le}(q, \mathcal{RS}, \mathcal{D})$ is Π_2^p-complete with respect to data complexity and PSPACE-complete with respect to combined complexity.*

Proof. The analysis of the algorithm $\mathsf{Answer_{LE}}$ shows that the problem is in Π_2^p w.r.t. data complexity. Indeed, it is immediate to verify that:

- condition (a) can be verified in polynomial time;
- condition (b) can be verified in polynomial time, as shown in Section 5.2.1;
- conditions (c1) and (c2) can be verified in polynomial time: therefore, condition (c) can be verified in nondeterministic polynomial time.

Consequently, if considered as a nondeterministic procedure, the algorithm runs in Σ_2^p w.r.t. data complexity, hence our problem (that is the complement of the problem solved by $\mathsf{Answer_{LE}}$) is in Π_2^p w.r.t. data complexity.

Hardness with respect to Π_2^p can be proved by a reduction from 2-QBF validity. In this respect we consider here an instance of our problem in which no ED is specified on the relational schema $\mathcal{RS}$. Let g be the 2-QBF formula $\forall x_1 \ldots x_n \exists y_1 \ldots y_m f(x_1, \ldots, x_n, y_1, \ldots, y_m)$ where f is a 3-CNF, i.e., a propositional formula of the form $c_1 \wedge \ldots \wedge c_k$, in which each conjunct c_i is of the form $l_1 \vee l_2 \vee l_3$ and each l_i is a literal of the form a or $\overline{a}$, with $a \in \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$. We construct a relational schema $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K \rangle$ as follows:

1. Ψ consists of four relations (with associated arities): $r_1/3, r_2/2, r_3/2, s/1$;
2. Σ_I is constituted of four unary IDs:

$$\begin{array}{lcll} r_2[2] & \subseteq & r_1[2] & \text{(ID1)} \\ r_1[3] & \subseteq & r_3[1] & \text{(ID2)} \\ r_2[1] & \subseteq & r_3[1] & \text{(ID3)} \\ r_3[2] & \subseteq & r_2[1] & \text{(ID4)} \end{array}$$

3. Σ_K consists of the KD $key(r_1) = 1$.

Moreover, we build an instance $\mathcal{D}$ for $\mathcal{RS}$, containing the following facts:

1. for each i such that $1 \leq i \leq n$, the facts $r_1(x_i, x_i, c_0), r_1(x_i, \overline{x}_i, c_0)$;
2. for each i such that $1 \leq i \leq m$, the facts $r_1(y_i, y_i, c_1), r_1(y_i, \overline{y}_i, c_1)$;
3. for each i such that $1 \leq i \leq k$ and for each a_j occurring in c_i, the fact $r_2(c_i, a_j)$;
4. for each i such that $1 \leq i < k$, the fact $r_3(c_i, c_{i+1})$, plus the fact $r_3(c_k, c_1)$;
5. the facts $r_1(c_0, c_0, c_0), r_2(c_0, c_0), r_3(c_0, c_0)$;
6. the facts $s(c_i)$ for each i such that $1 \leq i \leq k$.

Finally, let q be the query $q \leftarrow s(X), r_3(X, Y)$. We prove that g is valid if and only if $\langle\rangle \in ans_{le}(q, \mathcal{RS}, \mathcal{D})$.

First, suppose that g is valid. Then, for each propositional evaluation μ of $x_1, \ldots, x_n$ there exists a propositional evaluation μ' of $y_1, \ldots, y_m$ such that the interpretation of f with respect to μ and μ' is true. Let $\mathcal{B}_\mu$ be the database obtained from $\mathcal{D}$ by:

(i) removing each fact of the form $r_1(x_i, x_i, c_0)$ if $\mu(x_i) = \mathit{false}$ and $r_1(x_i, \overline{x}_i, c_0)$ if $\mu(x_i) = \mathit{true}$;

(ii) removing each fact of the form $r_1(y_i, y_i, c_1)$ if $\mu'(y_i) = \mathit{false}$ and $r_1(y_i, \overline{y}_i, c_1)$ if $\mu'(y_i) = \mathit{true}$;

(iii) removing each fact of the form $r_2(c_i, x_j)$ (respectively, $r_2(c_i, \overline{x}_j)$) if $\mu(x_j) = \mathit{false}$ (respectively, $\mu(x_j) = \mathit{true}$) and removing all facts of the form $r_2(c_i, y_j)$ (respectively, $r_2(c_i, \overline{y}_j)$) if $\mu'(y_j) = \mathit{false}$ (respectively, $\mu'(y_j) = \mathit{true}$).

Now, since the interpretation of f with respect to μ and μ' is true, it follows that for each i such that $1 \leq i \leq k$ there is a fact of the form $r_2(c_i, V)$ in $\mathcal{B}_\mu$ (from now on the symbols V, V' represent generic values), which implies that $\mathcal{B}_\mu$ satisfies all the IDs in Σ_I; moreover, it is immediate to see that $\mathcal{B}_\mu$ is consistent with Σ_K. Then consider any $\mathcal{B}'$ such that $\mathcal{B}' \in sem_{le}(\mathcal{RS}, \mathcal{D})$. It can be immediately verified that, due to Σ_K and Σ_I, $\mathcal{B}'$ must be such that there exists an evaluation μ of $x_1, \ldots, x_n$ such that the set of facts in $\mathcal{B}'$ of the form $r_1(x_i, V, V')$ coincides with the set of facts in $\mathcal{B}_\mu$ of the form $r_1(x_i, V, V')$. Now suppose that in $\mathcal{B}'$ there is no fact of the form $r_3(c_i, V)$ such that $1 \leq i \leq k$. Due to (ID3) and to the fact that $\mathcal{B}'$ is consistent with Σ_I, it follows that in $\mathcal{B}'$ there is no fact of the form $r_2(c_i, V)$ such that $1 \leq i \leq k$. This in turn implies that, due

to (ID4), in $\mathcal{B}'$ there is no fact of the form $r_1(V, V', c_1)$, consequently no fact in $\mathcal{D}$ of the form $r_1(y_i, V, V')$ is in $\mathcal{B}'$. Therefore, $\mathcal{B}' \cap \mathcal{D} \subset \mathcal{B}_\mu \cap \mathcal{D}$, and since $\mathcal{B}_\mu - \mathcal{D} = \emptyset$ and $\mathcal{B}_\mu$ is consistent with Σ_I and Σ_K, it follows that $\mathcal{B}' \notin sem_{le}(\mathcal{RS}, \mathcal{D})$, thus contradicting the hypothesis. Hence, for each $\mathcal{B}'$ such that $\mathcal{B}' \in sem_{le}(\mathcal{RS}, \mathcal{D})$, there is at least one fact of the form $r_3(c_i, V)$ such that $1 \leq i \leq k$, and since each such $\mathcal{B}'$ contains the fact $s(c_i)$ for each i such that $1 \leq i \leq k$, it follows that $\langle\rangle \in ans_{le}(\mathcal{RS}, \mathcal{D}, q)$.

Suppose now that g is not valid. Therefore, there exists an evaluation μ of $x_1, \ldots, x_n$ such that for each evaluation μ' of $y_1, \ldots, y_m$, the interpretation of f with respect to μ and μ' is false. Let $\mathcal{B}$ be the database obtained from $\mathcal{D}$ by:

(i) removing each fact of the form $r_1(x_i, x_i, c_0)$ if $\mu(x_i) = \mathit{false}$ and $r_1(x_i, \overline{x}_i, c_0)$ if $\mu(x_i) = \mathit{true}$;

(ii) removing all facts of the form $r_1(y_i, V, V')$;

(iii) removing all facts of the form $r_2(V, V')$ and each fact of the form $r_3(c_i, V)$ such that $1 \leq i \leq k$.

It is immediate to see that $\mathcal{B} \in sem_{le}(\mathcal{RS}, \mathcal{D})$, since it corresponds to a maximal subset of $\mathcal{D}$ consistent with Σ_I and Σ_K. Hence, in $\mathcal{B}$ there is no fact of the form $r_3(c_i, V)$ such that $1 \leq i \leq k$. Therefore, $\langle\rangle \notin q^{\mathcal{B}}$, which implies that $\langle\rangle \notin ans_{le}(\mathcal{RS}, \mathcal{D}, q)$.

As concerns combined complexity, it is immediate to verify that each of the conditions of the algorithm is computed in nondeterministic polynomial space, therefore the algorithm runs in nondeterministic polynomial space w.r.t. combined complexity, which proves membership in PSPACE of the problem. PSPACE-hardness can be proved by reducing query answering under loosely-sound semantics for databases without key dependencies to this problem. The reduction is obtained by a slight modification of the reduction from query answering under loosely-sound semantics exhibited in the proof of Theorem 5.3.10, and observing that, if the original problem instance is such that, for each $I = r[\vec{\mathbf{A}}] \subseteq s[\vec{\mathbf{B}}] \in \Sigma_I$, $\vec{\mathbf{B}}$ does not cover the set of all the attributes of s, then the derived database schema $\mathcal{RS}'$ is a NKC schema. Moreover, it is immediate to verify that restricting to such a kind of problem instances does not affect PSPACE-hardness of the query answering problem under the loosely-sound semantics. Finally, the reduction is modified in a way such that the database instance $\mathcal{D}'$ obtained from the original instance $\mathcal{D}$ has size polynomial w.r.t. combined complexity. □

From the proof of the above theorem, it is easy to see that the complexity of the query answering problem in the absence of EDs, i.e., when $\mathcal{RS} = \langle\Psi, \Sigma_I, \Sigma_K\rangle$, actually coincides with the complexity of the general case. Then, the question arises whether the complexity is the same also in the presence of only IDs and EDs. The following theorem affirmatively replies to this question.

Theorem 5.3.16 *Let $\mathcal{RS} = \langle\Psi, \Sigma_I, \Sigma_E\rangle$ be a relational schema, $\mathcal{D}$ a database instance, q a query of arity n over $\mathcal{RS}$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. The problem of establishing whether $t \in ans_{le}(q, \mathcal{RS}, \mathcal{D})$ is Π_2^p-complete with respect to data complexity and PSPACE-complete with respect to combined complexity.*

Proof. Membership for both data and combined complexity is as in Theorem 5.3.15.

Hardness with respect to Π_2^p can be proved by a reduction from 2-QBF validity. To this aim, we consider here an instance of our problem in which no ED is specified on the relational schema $\mathcal{RS}$. Let g be the 2-QBF formula $\forall x_1 \ldots x_n \exists y_1 \ldots y_m f(x_1, \ldots, x_n, y_1, \ldots, y_m)$ where f is a 3-CNF, i.e., a propositional formula of the form $c_1 \wedge \ldots \wedge c_k$, in which each conjunct c_i is of the form $l_1 \vee l_2 \vee l_3$ and each l_i is a literal of the form a or $\overline{a}$, with $a \in \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$. We construct a relational schema $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_K \rangle$ as follows:

1. Ψ consists of four relations (with associated arities): $r_1/3, r_2/2, r_3/2, s/1$;
2. Σ_I is constituted of four unary IDs:

$$\begin{array}{lcll} r_2[2] & \subseteq & r_1[2] & \text{(ID1)} \\ r_1[3] & \subseteq & r_3[1] & \text{(ID2)} \\ r_2[1] & \subseteq & r_3[1] & \text{(ID3)} \\ r_3[2] & \subseteq & r_2[1] & \text{(ID4)} \end{array}$$

3. Σ_K consists of the KD $key(r_1) = 1$.

Moreover, we build an instance $\mathcal{D}$ for $\mathcal{RS}$, containing the following facts:

1. for each i such that $1 \leq i \leq n$, the facts $r_1(x_i, x_i, c_0), r_1(x_i, \overline{x}_i, c_0)$;
2. for each i such that $1 \leq i \leq m$, the facts $r_1(y_i, y_i, c_1), r_1(y_i, \overline{y}_i, c_1)$;
3. for each i such that $1 \leq i \leq k$ and for each a_j occurring in c_i, the fact $r_2(c_i, a_j)$;
4. for each i such that $1 \leq i < k$, the fact $r_3(c_i, c_{i+1})$, plus the fact $r_3(c_k, c_1)$;
5. the facts $r_1(c_0, c_0, c_0), r_2(c_0, c_0), r_3(c_0, c_0)$;
6. the facts $s(c_i)$ for each i such that $1 \leq i \leq k$.

Finally, let q be the query $q \leftarrow s(X), r_3(X, Y)$. We prove that g is valid if and only if $\langle\rangle \in ans_{le}(q, \mathcal{RS}, \mathcal{D})$.

First, suppose that g is valid. Then, for each propositional evaluation μ of $x_1, \ldots, x_n$ there exists a propositional evaluation μ' of $y_1, \ldots, y_m$ such that the interpretation of f with respect to μ and μ' is true. Let $\mathcal{B}_\mu$ be the database obtained from $\mathcal{D}$ by:

(i) removing each fact of the form $r_1(x_i, x_i, c_0)$ if $\mu(x_i) = \mathit{false}$ and $r_1(x_i, \overline{x}_i, c_0)$ if $\mu(x_i) = \mathit{true}$;

(ii) removing each fact of the form $r_1(y_i, y_i, c_1)$ if $\mu'(y_i) = \mathit{false}$ and $r_1(y_i, \overline{y}_i, c_1)$ if $\mu'(y_i) = \mathit{true}$;

(iii) removing each fact of the form $r_2(c_i, x_j)$ (respectively, $r_2(c_i, \overline{x}_j)$) if $\mu(x_j) = \mathit{false}$ (respectively, $\mu(x_j) = \mathit{true}$) and removing all facts of the form $r_2(c_i, y_j)$ (respectively, $r_2(c_i, \overline{y}_j)$) if $\mu'(y_j) = \mathit{false}$ (respectively, $\mu'(y_j) = \mathit{true}$).

Now, since the interpretation of f with respect to μ and μ' is true, it follows that for each i such that $1 \leq i \leq k$ there is a fact of the form $r_2(c_i, V)$ in $\mathcal{B}_\mu$ (from now on the symbols V, V' represent generic values), which implies that $\mathcal{B}_\mu$ satisfies all the IDs in Σ_I; moreover, it is immediate to see that $\mathcal{B}_\mu$ is consistent with Σ_K. Then consider any $\mathcal{B}'$ such that $\mathcal{B}' \in \mathit{sem}_{le}(\mathcal{RS}, \mathcal{D})$. It can be immediately verified that, due to Σ_K and Σ_I, $\mathcal{B}'$ must be such that there exists an evaluation μ of $x_1, \ldots, x_n$ such that the set of facts in $\mathcal{B}'$ of the form $r_1(x_i, V, V')$ coincides with the set of facts in $\mathcal{B}_\mu$ of the form $r_1(x_i, V, V')$. Now suppose that in $\mathcal{B}'$ there is no fact of the form $r_3(c_i, V)$ such that $1 \leq i \leq k$. Due to (ID3) and to the fact that $\mathcal{B}'$ is consistent with Σ_I, it follows that in $\mathcal{B}'$ there is no fact of the form $r_2(c_i, V)$ such that $1 \leq i \leq k$. This in turn implies that, due to (ID4), in $\mathcal{B}'$ there is no fact of the form $r_1(V, V', c_1)$, consequently no fact in $\mathcal{D}$ of the form $r_1(y_i, V, V')$ is in $\mathcal{B}'$. Therefore, $\mathcal{B}' \cap \mathcal{D} \subset \mathcal{B}_\mu \cap \mathcal{D}$, and since $\mathcal{B}_\mu - \mathcal{D} = \emptyset$ and $\mathcal{B}_\mu$ is consistent with Σ_I and Σ_K, it follows that $\mathcal{B}' \notin \mathit{sem}_{le}(\mathcal{RS}, \mathcal{D})$, thus contradicting the hypothesis. Hence, for each $\mathcal{B}'$ such that $\mathcal{B}' \in \mathit{sem}_{le}(\mathcal{RS}, \mathcal{D})$, there is at least one fact of the form $r_3(c_i, V)$ such that $1 \leq i \leq k$, and since each such $\mathcal{B}'$ contains the fact $s(c_i)$ for each i such that $1 \leq i \leq k$, it follows that $\langle\rangle \in \mathit{ans}_{le}(\mathcal{RS}, \mathcal{D}, q)$.

Suppose now that g is not valid. Therefore, there exists an evaluation μ of $x_1, \ldots, x_n$ such that for each evaluation μ' of $y_1, \ldots, y_m$, the interpretation of f with respect to μ and μ' is false. Let $\mathcal{B}$ be the database obtained from $\mathcal{D}$ by:

(i) removing each fact of the form $r_1(x_i, x_i, c_0)$ if $\mu(x_i) = \mathit{false}$ and $r_1(x_i, \overline{x}_i, c_0)$ if $\mu(x_i) = \mathit{true}$;

(ii) removing all facts of the form $r_1(y_i, V, V')$;

(iii) removing all facts of the form $r_2(V, V')$ and each fact of the form $r_3(c_i, V)$ such that $1 \leq i \leq k$.

It is immediate to see that $\mathcal{B} \in \mathit{sem}_{le}(\mathcal{RS}, \mathcal{D})$, since it corresponds to a maximal subset of $\mathcal{D}$ consistent with Σ_I and Σ_K. Hence, in $\mathcal{B}$ there is no fact of the form $r_3(c_i, V)$ such that $1 \leq i \leq k$. Therefore, $\langle\rangle \notin q^{\mathcal{B}}$, which implies that $\langle\rangle \notin \mathit{ans}_{le}(\mathcal{RS}, \mathcal{D}, q)$.

As concerns combined complexity, PSPACE-hardness can be proved by reducing query answering under loosely-sound semantics for databases without key dependencies to this problem. The reduction is obtained by a slight modification of the reduction from query answering under loosely-sound semantics exhibited in the proof of Theorem 5.3.10, and observing that, if the original problem instance is such that, for each $I = r[\vec{\mathbf{A}}] \subseteq s[\vec{\mathbf{B}}] \in \Sigma_I$, $\vec{\mathbf{B}}$ does not cover the set of all the attributes of s, then the derived database schema $\mathcal{RS}'$ is a NKC schema. Moreover, it is immediate to verify that restricting to such a kind of problem instances does not affect PSPACE-hardness of the query answering problem under the loosely-sound semantics. Finally, the reduction is modified in a way such that the database instance $\mathcal{D}'$ obtained from the original instance $\mathcal{D}$ has size polynomial w.r.t. combined complexity.

□

We finally point out that query answering under the loosely-exact semantics in the absence of IDs coincides with the same problem under the loosely-sound semantics, hence algorithms and complexity results are the same in the two cases.

Data/Combined complexity for *general* database instances:

EDs	KDs	IDs	strictly-sound	loosely-sound	loosely-exact
no	no	GEN	PTIME♠/PSPACE♠	PTIME/PSPACE	PTIME/NP
yes-no	yes	no	PTIME♠/NP♠	coNP♠/Π_2^p	coNP♠/Π_2^p
yes	yes-no	no	PTIME♠/NP♠	coNP♠/Π_2^p	coNP♠/Π_2^p
yes-no	yes	NKC	PTIME/PSPACE	coNP/PSPACE	Π_2^p/PSPACE
yes	no	GEN	PTIME/PSPACE	coNP/PSPACE	Π_2^p/PSPACE
yes-no	yes	1KC	undecidable	undecidable	undecidable
yes-no	yes	GEN	undecidable♠	undecidable	undecidable

Data/Combined complexity for databases consistent with KDs and EDs:

EDs	KDs	IDs	strictly-sound	loosely-sound	loosely-exact
no	no	GEN	PTIME♠/PSPACE♠	PTIME/PSPACE	PTIME/NP
yes-no	yes	no	PTIME♠/NP♠	PTIME♠/NP♠	PTIME♠/NP♠
yes	yes-no	no	PTIME♠/NP♠	PTIME♠/NP♠	PTIME♠/NP♠
yes-no	yes	NKC	PTIME/PSPACE	PTIME/PSPACE	PTIME/NP
yes	no	GEN	PTIME/PSPACE	PTIME/PSPACE	PTIME/NP
yes-no	yes	1KC	undecidable	undecidable	PTIME/NP
yes-no	yes	GEN	undecidable♠	undecidable	PTIME/NP

Legenda: GEN = general IDs, NKC = non-key-conflicting IDs, 1KC = 1-key-conflicting IDs, ♠ = already known results.

Figure 5.1: Complexity of query answering under EDs, KDs and IDs (decision problem)

5.4 Summary of Results

The summary of the results we have obtained is reported in Figure 5.1, in which we have two distinct tables, that present, respectively, the complexity of query answering for the class of general database instances and for instances consistent with KDs and EDs[7]. Each column (with the exception of the first three) corresponds to a different semantics, while each row corresponds to a different combination of dependencies (specified in the first three columns, where *yes* indicates the presence of the dependencies, *no* the absence, and *yes-no* that the corresponding complexity is independent by these dependencies). Each cell of the tables reports data complexity and combined complexity of query answering: for each decidable case, the complexity of the problem is complete with respect to the class reported (with the exception of the cases in which the complexity is PTIME). We have marked with the symbol ♠ the cells corresponding either to already known results or to results straightforwardly implied by known results.

In the first table, complexity under the strictly-sound semantics for the case in which no IDs are specified on the schema coincide with the complexity of standard query answering over

[7]We assume here that the EDs are closed w.r.t. logical implication of IDs and EDs.

relational databases [2]. Notice also that in the presence of only IDs the strictly-sound and the loosely-sound semantics coincide, whereas complexity under the loosely-exact semantics in this case has been proved in Theorem 5.3.11 (in the absence of IDs and EDs a generic database instance satisfies the assumptions of such theorem).

We underline also that, in the second table, the strictly-sound and the loosely-sound semantics coincide. Furthermore, in this table, all results for the loosely-exact semantics can be proved as done for Theorem 5.3.11.

We point out that, due to the correspondence between query answering and query containment illustrated in Section 5.2.1, all the complexity results established for the problem of query answering also hold for the conjunctive query containment problem.

5.5 Related Work

The problem of reasoning with inconsistent databases is closely related to the studies in *belief revision and update* [6]. This area of Artificial Intelligence studies the problem of integrating new information with previous knowledge. In general, the problem is studied in a logical framework, in which the new information is a logical formula f and the previous knowledge is a logical theory (also called knowledge base) T. Of course, f may in general be inconsistent with T. The *revised* (or *updated*) knowledge base is denoted as $T \circ f$, and several semantics have been proposed for the operator $\circ$. The semantics for belief revision can be divided into *revision* semantics, when the new information f is interpreted as a modification of the knowledge about the world, and *update* semantics, when f reflects a change in the world.

The problem of reasoning with inconsistent databases can be actually seen as a problem of belief revision. In fact, with respect to the above illustrated knowledge base revision framework, if we consider the database instance $\mathcal{D}$ as the initial knowledge base T, and the set of integrity constraints Σ as the new information f, then the problem of deciding whether a tuple t is in the answer set of a query q with respect to the database schema $\mathcal{RS} = \langle \Psi, \Sigma \rangle$ and the instance $\mathcal{D}$ corresponds to the belief revision problem $\mathcal{D} \circ \Sigma \models q(t)$. Based on such a correspondence, the studies in belief revision appear very relevant for the field of inconsistent databases: indeed, almost all the approaches to inconsistent databases that we have considered in this section can be reconstructed in terms of direct applications of well-known semantics for belief revision/update in a particular class of theories.

On the other hand, from a computational perspective, there are no results concerning the particular kind of belief revision/update that is of interest for database applications: in particular, the class of relational integrity constraints as revision/update knowledge has not been taken into account in the belief revision literature, where the computational results mostly concern a setting in which knowledge is specified in terms of propositional formulae of classical logic [64, 65]. Instead, the typical database setting is considered by the literature on *inconsistent databases*, which we briefly survey in the following.

The notion of consistent query answers over inconsistent databases was originally given in [17]. However, the approach in [17] is completely proof-theoretic, and no computational technique for obtaining consistent answers from inconsistent database is provided.

In [123] the authors describe an operator for *merging databases* under constraints which allows for obtaining a maximal amount of information from each database by means of a majority criterion used in case of conflict. Even if a large set of constraints is considered, namely the constraints that can be expressed as first-order formulae, the computational complexity of the merging procedure is not explored, and no algorithm to compute consistent query answers is provided. Furthermore, the problem of dealing with incomplete databases is not taken into account. Notice also that, different from all the other studies mentioned in the following, this approach relies on a cardinality-based ordering between databases (rather than a set-containment-based ordering).

In [71] the authors propose a framework for updating theories and logical databases (i.e., databases obtained by giving priorities to sentences in the databases) that can be extended also to the case of updating views. The semantics proposed in such a paper is based on a particular set-containment based ordering between theories that "accomplish" an update to an original theory. A theory T_1 accomplishes an insertion of σ into T if $\sigma \in T_1$, and accomplishes a deletion of σ if σ is not a logical consequence of T_1. Then, a theory T_1 accomplishes an update u to T with a *smaller change than* T_2, and thus is preferred to T_2, if both T_1 and T_2 accomplish u, but T_1 has fewer deletions than T_2, or they have the same deletions but T_1 has fewer insertions than T_2. The basic idea is to maintain as many as possible of the facts that are known to be true. This semantics is similar to the loosely-sound semantics presented in Section 5.1.

In [7] the authors define an algorithm for consistent query answers in inconsistent databases based on the notion of *residues*, originally defined in the context of semantic query optimization. The method is proved to be sound and complete only for the class of universally quantified binary constraints, i.e., constraints that involve two database relations. In [8] the same authors propose a new method that can handle arbitrary universally quantified constraints by specifying the database repairs into *logic rules with exceptions* (LPe). The semantics underlying the notion of consistent query answers both in [7] and in [8] is defined on a set-containment ordering between databases, which corresponds to the loosely-exact semantics of our framework.

Moreover, a different semantics for database repairing has been considered in [52, 51]. Specifically, in such works a semantics is defined in which only tuple elimination is allowed; therefore, the problem of dealing with infinite models is not addressed. Then, a preference order over the database repairs is defined, in such a way that only minimal repairs (in terms of set containment) are considered. Hence, the semantics is a "maximal complete" one, in the sense that only maximal consistent subsets of the database instance are considered as repairs of such an instance. In [52] the authors establish complexity results for query answering under such a semantics in the presence of *denial constraints*, a generalization of key dependencies and exclusion dependencies, while in [51] also inclusion dependencies are considered. Such a "maximal complete" semantics is different from the complete semantics considered in the present paper.

Finally, [86] proposes a technique to deal with inconsistencies that is based on the reformulation of integrity constraints into a disjunctive datalog program with two different forms of negation: negation as failure and classical negation. Such a program can be used both to repair databases, i.e., modify the data in the databases in order to satisfy integrity constraints, and to compute consistent query answers. The technique is proved to be sound and complete for

universally quantified constraints. The semantics adopted to support this method corresponds to our loosely-exact semantics.

Chapter 6

Query Answering and Rewriting in GAV Data Integration Systems

In this chapter, we turn our attention to a data integration context and restrict our analysis to the GAV approach. We recall that in our integration framework, the integrity constraints allowed for the global schema are inclusion, key and exclusion dependencies, the language adopted for queries in the GAV mapping is non-recursive Datalog$^\neg$, while the language allowed for user queries is that of union of conjunctive queries.

In the following, we first provide decidability and complexity results on query answering in GAV for all the semantics introduced in Section 3, then we focus on strictly-sound and loosely-sound mappings.

6.1 Contributions

The main contributions of this chapter are the following:

1. we extend to GAV data integration systems the decidability and complexity results for query answering given in Chapter 5 in the setting of a single database;

2. we provide a sound and complete query rewriting technique under the strictly-sound semantics, first for the case of IDs alone, and then for the case of EDs, KDs and NKCIDs;

3. we present a sound and complete query rewriting technique in the presence of the same constraints under the loosely-sound semantics, thus allowing for consistent query answering when data retrieved from the sources may violate EDs and KDs.

Results in this chapter advance the state of the art in data integration in terms of: (i) decidability and complexity results; (ii) new algorithms: to the best of our knowledge, our methods are the first that solve query answering in the presence of EDs, KDs and cyclic IDs, for both incomplete and inconsistent data; (iii) *effective* techniques for data integration which exploit intensional processing of queries. Indeed, our algorithms are able to intensionally carry

out query processing, since treatment of integrity constraints is essentially done at the query and schema level. As we will see in Chapters 8 and 9, this strongly augments the feasibility of the approach, and allows for interesting optimizations.

Before proceeding we introduce an example that we will use throughout this chapter.

Example 6.1.1 Let $\mathcal{I}_0 = \langle \mathcal{G}_0, \mathcal{S}_0, \mathcal{M}_0 \rangle$ be a data integration system referring to the context of football teams. The global schema $\mathcal{G}_0 = \langle \Psi_0, \Sigma_0 \rangle$ consists of the three relations

$$\begin{array}{c} player(Pcode, Pname, Pteam) \\ team(Tcode, Tname, Tleader) \\ coach(Ccode, Cname, Cteam) \end{array}$$

and the following constraints:

$$\begin{array}{rcl} key(player) & = & \{Pcode\} \\ key(team) & = & \{Tcode\} \\ key(coach) & = & \{Ccode\} \\ team[Tleader] & \subseteq & player[Pcode] \\ player[Pcode] & \cap & coach[Ccode]) = \emptyset \end{array}$$

where the last two constraints state that a team leader has to be a player, and that a coach can neither be a player nor a team leader.

The source schema $\mathcal{S}_0$ comprises the relation s_1 of arity 4, and the relations s_2, s_3 and s_4, of arity 3. The mapping $\mathcal{M}_0$ is defined by the three assertions

$$\begin{array}{rl} \langle player, & player(X, Y, Z) \leftarrow s_1(X, Y, Z, W) \rangle \\ \langle team, & team(X, Y, Z) \leftarrow s_2(X, Y, Z) \\ & team(X, Y, Z) \leftarrow s_3(X, Y, Z) \rangle \\ \langle coach, & coach(X, Y, Z) \leftarrow s_4(X, Y, Z) \rangle \end{array}$$

Finally, let Q_0 be the user query of the form $q(X) \leftarrow player(X, Y, Z)$ that asks for the codes of all players. ∎

6.2 Decidability and Complexity of Query Answering

In Chapter 5, we have abstracted from the integration context and have focused on the setting of a single relational schema in which (strict and loose) sound, complete or exact interpretations on the stored data could be adopted. In order to clarify the relationship between such interpretations and the ones that can be assumed on the mapping of a data integration system, we have described a relational schema $\mathcal{RS}$ in terms of the global schema of a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ (that can be either LAV or GAV) where there is a one-to-one correspondence between global and source relations (see Section 5.1). Then we have provided decidability and complexity results for query answering in the framework of a single relational schema.

We now go back to the GAV integration framework and generalize to such a setting the results provided in Chapter 5. To this aim, we first exploit the correspondence between $\mathcal{RS}$ and $\mathcal{I}$ described in Section 5.1. More precisely, assuming that $\mathcal{RS}$ contains the relations $g_1/h_1, \ldots, g_n/h_n$,

we consider a database $\mathcal{D}'$ for $\mathcal{RS}$, a query Q over $\mathcal{RS}$ and a tuple t of constants of the active domain $\mathcal{U}_{\mathcal{D}}$[1]. Then, we can construct $\mathcal{I}$, in time that is polynomial in the size of the schema and the query, as follows:

- $\mathcal{G} = \mathcal{RS}$;
- the source schema $\mathcal{S}$ contains the relations $s_1/h_1, \ldots, s_n/h_n$;
- the mapping $\mathcal{M}$ is given by the n assertions of the form $\langle g_i, g_i(X_1, \ldots, X_k) \leftarrow s_i(X_1, \ldots, X_k)\rangle$, for each i, $1 \leq i \leq n$;
- a source database $\mathcal{D}$ for $\mathcal{I}$ is obtained by assigning to each source relation s_i the tuples that $\mathcal{D}'$ assigns to the corresponding global relation g_i, i.e., $s_i^{\mathcal{D}} = g_i^{\mathcal{D}'}$;
- Q and t remain unchanged.

It should be easy to see that $sem_{as(\mathcal{D}')}(\mathcal{RS}, \mathcal{D}') = sem_{as(\mathcal{M})}(\mathcal{I}, \mathcal{D})$[2], hence, $t \in ans_{as(\mathcal{D}')}(Q, \mathcal{RS}, \mathcal{D}')$ iff $t \in ans_{as(\mathcal{M})}(Q, \mathcal{I}, \mathcal{D})$. The above reduction actually proves the following theorem.

Theorem 6.2.1 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M}\rangle$ be a GAV data integration system, where $\mathcal{G} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E\rangle$, in which Σ_K is a set of KDs, Σ_E is a set of EDs and Σ_I is a set of 1KCIDs w.r.t. Σ_K. Let $\mathcal{D}$ be a source database for $\mathcal{I}$, Q a user query of arity n over $\mathcal{G}$, and t an n-tuple of constants of $\mathcal{U}_{\mathcal{D}}$. Then, the problem of calculating $ans_{as(\mathcal{M})}(Q, \mathcal{I}, \mathcal{D})$ is undecidable for the sound, the loosely-sound and the loosely-exact assumption on the mapping $\mathcal{M}$, i.e., for $as(\mathcal{M}) = s$, ls or le, respectively.*

In the following we call 1-key-conflicting integration systems the systems where the global schema $\mathcal{G}$ is a 1KC relational schema, and non-key-conflicting integration systems the ones in which $\mathcal{G}$ is a NKC schema.

Notice that the above reduction allows us also to extend to the GAV integration framework all the complexity lower bounds provided in Chapter 5. To conclude that also the same upper bounds hold in this setting, we need to establish membership results for all the combinations of IDs, KDs and EDs under the different semantics considered in Chapter 5. To this aim, we provide the following definition.

Definition 6.2.2 Given a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M}\rangle$ and a source database $\mathcal{D}$ for $\mathcal{I}$, we call *retrieved global database*, denoted $ret(\mathcal{I}, \mathcal{D})$, the global database obtained by evaluating each query of the mapping $\mathcal{M}$ over $\mathcal{D}$, i.e., $ret(\mathcal{I}, \mathcal{D}) = \{r_{\mathcal{G}}(t) \mid t \in q_{\mathcal{S}}^{\mathcal{D}}$ for each $\langle r_{\mathcal{G}}, q_{\mathcal{S}}\rangle \in \mathcal{M}\}$.

Since queries in $\mathcal{M}$ are non-recursive Datalog$^\neg$ queries, then $ret(\mathcal{I}, \mathcal{D})$ is unique.

A significant property of the retrieved global database is that it satisfies the mapping, whatever assumption on $\mathcal{M}$ is adopted. In particular, under the exact assumption $ret(\mathcal{I}, \mathcal{D})$ is the

[1] We recall that the active domain for $\mathcal{D}$ is the subset of the domain $\mathcal{U}$ that contains the constants occurring in the source database $\mathcal{D}$

[2] Notice that in the left-hand side *sem* refers to a data integration system, whereas in the right hand side it refers to a database schema.

only database that satisfies the mapping, whereas under the sound or the complete assumption, it is in general one of several databases that satisfy the mapping, viz. all the databases that contain $ret(\mathcal{I}, \mathcal{D})$, or that are contained in $ret(\mathcal{I}, \mathcal{D})$, in the two cases, respectively. It should be easy now to generalize to each GAV data integration system the property holding for the particular case in which in $\mathcal{I}$ there is a one-to-one correspondence between relations in $\mathcal{G}$ and relations in $\mathcal{S}$. Indeed, the semantics of $\mathcal{I}$ with respect of a source database $\mathcal{D}$ and an assumption $as(\mathcal{M})$ on $\mathcal{M}$, coincides with the semantics of the global schema $\mathcal{G}$ with respect to the database instance $ret(\mathcal{I}, \mathcal{D})$ and the same assumption $as(ret(\mathcal{I}, \mathcal{D}))$ on $ret(\mathcal{I}, \mathcal{D})$, i.e., $as(\mathcal{M}) = as(ret(\mathcal{I}, \mathcal{D}))$. Hence, answering a user query Q in such a setting can be tackled in two steps, where the first one aims at computing the retrieved global database, and the second one at calculating the certain answers to Q over $ret(\mathcal{I}, \mathcal{D})$, taking into account the assumption on $ret(\mathcal{I}, \mathcal{D})$, i.e., $ans_{as(\mathcal{M})}(Q, \mathcal{I}, \mathcal{D}) = ans_{as(ret(\mathcal{I},\mathcal{D}))}(Q, \mathcal{G}, ret(\mathcal{I}, \mathcal{D}))$.

We are now able to provide the separation theorem for GAV integration systems thus extending to this setting the result established in Theorem 5.2.16.

Theorem 6.2.3 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a NKC GAV system, where $\mathcal{G} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$, let $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G}' = \langle \Psi, \Sigma_I \rangle$, be the system obtained by $\mathcal{I}$ by eliminating the KDs and the EDs of $\mathcal{G}$; let $\mathcal{D}$ be a source database for $\mathcal{I}$ and $\mathcal{I}'$. Moreover, let Q be a query of arity n over $\mathcal{G}$ and $\mathcal{G}'$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. We have that $t \notin ans_s(Q, \mathcal{I}, \mathcal{D})$ iff $ret(\mathcal{I}, \mathcal{D})$ is consistent with $\Sigma_K \cup \Sigma_E^*$ and $t \notin ans_s(Q, \mathcal{I}', \mathcal{D})$, where Σ_E^* indicates the set EDs that are logically implied by the dependencies in $\Sigma_E \cup \Sigma_I$.*

Proof. The thesis is an immediate consequence of Theorem 5.2.16, and of $ans_{as(\mathcal{M})}(Q, \mathcal{I}, \mathcal{D}) = ans_{as(ret(\mathcal{I},\mathcal{D}))}(Q, \mathcal{G}, ret(\mathcal{I}, \mathcal{D}))$. □

Finally, we can conclude that all membership results given in Chapter 5 still hold in our framework for NKC GAV systems. Indeed, the computation of the retrieved global database is in PTIME in data complexity and is it PSPACE-complete in combined complexity, since evaluating non-recursive Datalog$^\neg$ is PTIME-complete in data complexity and PSPACE-complete in combined complexity[3] [54]. Obviously, this generalizes to GAV systems all complexity results reported in Figure 5.1.

6.3 Query Rewriting under the Strictly-sound Semantics

In this section we present algorithms for computing the perfect rewriting of a UCQ query in GAV integration systems with NKCIDs, KDs, and EDs, under the strictly-sound semantics. We recall that the perfect rewriting of a user query Q over a data integration systems $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, is a reformulation Q_r of Q that can be evaluated over the source schema $\mathcal{S}$, and that provides the certain answers to Q for each source database for $\mathcal{I}$, i.e., $Q_r^\mathcal{D} = ans_s(Q, \mathcal{I}, \mathcal{D})$ for each $\mathcal{D}$. We first study the case in which only IDs are expressed on the global schema, then we deal with the simultaneous presence of NKIDs, KDs and EDs.

[3] Notice that allowing recursion in the mapping, both with or without stratified negation, would lead to different combined complexity results, since Datalog and Datalog$^{\neg s}$ are combined complete for EXPTIME [54].

6.3.1 Query Rewriting in the presence of IDs

We start by studying query rewriting when only IDs are expressed on the global schema. To this aim, we present an algorithm that computes the perfect rewriting of a UCQ in a single database with inclusion dependencies under the strictly-sound semantics[4] (see Section 5.1 for the definition of strictly-sound semantics in a single database setting). Before proceeding, we need some preliminary definitions.

Preliminary definitions

Given a conjunctive query q over a relational schema $\mathcal{RS} = \langle \Psi, \Sigma_I \rangle$, we say that a variable X is *unbound* in q if it occurs only once in q, otherwise we say that X is *bound* in q. Notice that variables occurring in the head of the query are necessarily bound, since each of them must also occur in the query body. A *bound term* is either a bound variable or a constant.

Following the standard notation used in deductive databases, we adopt a special symbol for all unbound variables in the query q: specifically, we use the special term "ξ" for such variables (deductive database systems use the symbol "_" to this purpose).

Definition 6.3.1 Given an atom $g = s(X_1, \ldots, X_n)$ and an inclusion dependency $I \in \Sigma_I$ of the form $r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k]$, we say that I *is applicable to* g if the following conditions hold:

1. for each ℓ such that $1 \leq \ell \leq n$, if $X_\ell \neq \xi$ then there exists h such that $j_h = \ell$;
2. there exists a variable substitution σ such that $\sigma(s[j_h]) = \sigma(s[j_{h'}])$ for each h, h' such that $i_h = i_{h'}$. If such a σ exists, we denote with $\sigma_{g,I}$ the most general substitution that verifies this condition (we recall that σ is a most general substitution if for every other substitution σ' that verifies this condition there exists a substitution γ such that $\sigma' = \gamma\sigma$[5]).

Moreover, if I is applicable to g, we denote with $gr(g, I)$ the atom $r(Y_1, \ldots, Y_m)$ (m is the arity of r in Ψ) where for each ℓ such that $1 \leq \ell \leq m$, $Y_\ell = \sigma_{g,I}(X_{j_h})$ if there exists h such that $i_h = \ell$, otherwise $Y_\ell = \xi$.

Roughly speaking, an inclusion I is applicable to an atom g if the relation symbol of g corresponds to the symbol in the right-hand side of I and if all the attributes for which bound terms appear in g are propagated by the inclusion I (condition 1 of the above definition). Moreover, if the left-hand side of the inclusion contains repeated attributes, then the inclusion is applicable only if the corresponding terms in g unify (condition 2 of the above definition). When I is applicable to g, $gr(g, I)$ denotes the atom obtained from g by using I as a rewriting rule, whose direction is right-to-left.

For instance, let $I = r[1, 2] \subseteq s[1, 3]$, $g = s(X, \xi, c)$ and let r be of arity 4. I is applicable to g, since the attributes of s which contain bound terms (i.e., attributes 1 and 3) are propagated by the inclusion, and $gr(g, I) = r(X, c, \xi, \xi)$. As another example, consider now $I = r[1, 1] \subseteq s[1, 3]$, $g = s(X, \xi, c)$ and let r be of arity 4. I is applicable to g since the terms X and c unify and $\sigma_{g,I} = \{X \rightarrow c\}$. Therefore, $gr(g, I) = r(c, c, \xi, \xi)$.

[4] In this setting, the perfect rewriting of a query Q over a schema $\mathcal{RS}$ is another query Q_r over $\mathcal{RS}$ such that $Q_r^{\mathcal{D}} = ans_s(Q, \mathcal{RS}, \mathcal{D})$ for each database instance $\mathcal{D}$ of $\mathcal{RS}$.

[5] Given two substitutions σ_1 and σ_2 we use $\sigma_1\sigma_2(D)$ as a shortcut for $\sigma_1(\sigma_2(D))$.

Definition 6.3.2 Given an atom $g_1 = r(X_1, \ldots, X_n)$ and an atom $g_2 = r(Y_1, \ldots, Y_n)$, we say that g_1 *and* g_2 *unify* if there exists a variable substitution σ such that $\sigma(g_1) = \sigma(g_2)$. Each such a σ is called *unifier*. Moreover, if g_1 and g_2 unify, we denote as $\mathcal{U}(g_1, g_2)$ *a most general unifier (mgu) of* g_1 *and* g_2.

Informally, two atoms unify if they can be made equal through a substitution of the variable symbols with other terms (either variables or constants) occurring in the atoms. We recall that each occurrence of ξ in the two atoms actually represents a different, new variable symbol, therefore each occurrence of ξ in g_1 and g_2 can be assigned to a different term.

For example, let $g_1 = r(X, \xi, c, \xi)$ and an atom $g_2 = r(\xi, Y, Z, \xi)$. Actually, g_1 and g_2 correspond to $g_1 = r(X, \xi_1, c, \xi_2)$, $g_2 = r(\xi_3, Y, Z, \xi_4)$. Then, g_1 and g_2 unify, and $\mathcal{U}(g_1, g_2) = \{\xi_3 \rightarrow X,\ \xi_1 \rightarrow Y,\ Z \rightarrow c,\ \xi_2 \rightarrow \xi_4\}$.

The algorithm ID-rewrite

In Figure 6.1 we define the algorithm ID-rewrite that computes the perfect rewriting of a UCQ Q over a relational schema $\mathcal{RS} = \langle \Psi, \Sigma_I \rangle$. In the algorithm, it is assumed that unbound variables in the input query Q are represented by the symbol ξ.

Specifically, the algorithm generates a set of conjunctive queries that constitute a perfect rewriting of Q, by computing the closure of the set Q with respect to the following two rules:

1. if there exists a conjunctive query $q \in Q$ such that q contains two atoms g_1 and g_2 that unify, then the algorithm adds to Q the conjunctive query reduce(q, g_1, g_2) obtained from q by the following algorithm[6]:

   Algorithm reduce(q, g_1, g_2)
   Input: conjunctive query q, atoms $g_1, g_2 \in body(q)$
   such that g_1 and g_2 unify
   Output: reduced conjunctive query q'
   $q' = q$;
   $\sigma := \mathcal{U}(g_1, g_2)$;
   $body(q') := body(q') - \{g_2\}$;
   $q' := \sigma(q')$;
   $q' := \tau(q')$;
   return q'

   Informally, the above algorithm reduce starts by eliminating g_2 from the query body; then, the substitution $\mathcal{U}(g_1, g_2)$ is applied to the whole query (both the head and the body), and finally the function τ is applied. Such a function replaces with ξ each variable symbol X such that there is a single occurrence of X in the query. The use of τ is necessary in order to guarantee that, in the generated query, each unbound variable is represented by the symbol ξ.

[6] With a little abuse, in the algorithms we use set operators on the body of conjunctive queries.

Algorithm ID-rewrite($\mathcal{RS}, Q$)
Input: relational schema $\mathcal{RS} = \langle \Psi, \Sigma_I \rangle$,
where Σ_I is a set of inclusion dependencies, UCQ Q
Output: perfect rewriting of Q
$Q' := Q$;
repeat
$Q_{aux} := Q'$;
for each $q \in Q_{aux}$ **do**
(a) **for each** $g_1, g_2 \in body(q)$ **do**
if g_1 and g_2 unify
then $Q' := Q' \cup \{\mathsf{reduce}(q, g_1, g_2)\}$;
(b) **for each** $g \in body(q)$ **do**
for each $I \in \Sigma_I$ **do**
if I is applicable to g
then $Q' := Q' \cup \{\mathsf{atom\text{-}rewrite}(q, g, I)\}$
until $Q_{aux} = Q'$;
return Q'

Figure 6.1: Algorithm ID-rewrite

2. if there exists an inclusion I and a conjunctive query $q \in Q$ containing an atom g such that I is applicable to g, then the algorithm adds to Q the query obtained from q by replacing g with $gr(g, I)$ and by applying the substitution $\sigma_{g,I}$ to q. Namely, this step adds new queries obtained by applying inclusion dependencies as rewriting rules (applied from right to left), through the following algorithm atom-rewrite:

Algorithm atom-rewrite(q, g, I)
Input: conjunctive query q, atom $g \in body(q)$,
ID I such that I is applicable to g
Output: rewritten conjunctive query q'
$q' = q$;
$body(q') := body(q') - \{g\}$;
$q' := \sigma_{g,I}(q')$;
$body(q') := body(q') \cup \{gr(g, I)\}$;
return q'

The above two rules correspond respectively to steps (a) and (b) of the algorithm.

Termination of the algorithm is immediately implied by the fact that the number of conjunctive queries that can be generated by the algorithm is finite, since the maximum length (i.e., the number of atoms) of a generated query is equal to the maximum length of the queries in Q, and the number of different atoms that can be generated by the algorithm is finite, since the alphabet of relation symbols used is finite (and corresponds to the relation symbols occurring in Q and in

Σ_I), as well as the set of terms used (corresponding to the set of variable names and constants occurring in the query Q plus the symbol ξ).

Example 6.3.3 Consider the relational schema $\mathcal{RS}_1 = \langle \Psi_1, \Sigma_{I_1} \rangle$, where Ψ_1 contains the following relations:

$$\begin{aligned} &\mathit{employee}(\mathit{Ename}, \mathit{Salary}, \mathit{Dep}, \mathit{Boss}) \\ &\mathit{manager}(\mathit{Mname}, \mathit{DOB}) \\ &\mathit{department}(\mathit{Code}, \mathit{Director}) \end{aligned}$$

and Σ_{I_1} contains the following inclusion dependencies:

$$\begin{aligned} \mathit{employee}[4] &\subseteq \mathit{manager}[1] && (I_1) \\ \mathit{manager}[1] &\subseteq \mathit{department}[2] && (I_2) \\ \mathit{department}[1] &\subseteq \mathit{employee}[3] && (I_3), \end{aligned}$$

where I_1 states that a boss has to be also a manager, I_2 states that a manager has to be the director of a department, and I_3 states that at least one employee works in each department.

Consider the user query Q_1 constituted by the only CQ $q(Dr) \leftarrow \mathit{department}(C, Dr), \mathit{employee}(Dr, S, D, B)$, which asks for directors of departments that are also employees.

For ease of exposition we pose $Q_1 = \{q_0\}$ where q_0 indicates the CQ in Q_1. Then we execute ID-rewrite$(\mathcal{RS}_1, Q_1)$. At the beginning, $Q_{aux} = Q' = Q_1 = \{q_0\}$. It is easy to see that step (a) has not to be performed for q_0, whereas, for step (b), the ID I_2 is applicable to the atom $g_1 = \mathit{department}(\xi, Dr)$. Hence, $g_r(g_1, I_2) = \mathit{manager}(Dr, \xi)$ and $Q' = \{q_0\} \cup \{q_1\}$, where q_1 has the form $q(Dr) \leftarrow \mathit{manager}(Dr, \xi), \mathit{employee}(Dr, \xi, \xi, \xi)$. As for the query q_1, I_1 is applicable to the atom $g_2 = \mathit{manager}(Dr, \xi)$, and $g_r(g_2, I_1) = \mathit{employee}(Dr, \xi)$. Hence, we have that $Q' = \{q_0, q_1\} \cup \{q_2\}$, where q_2 has the form $q(Dr) \leftarrow \mathit{employee}(\xi, \xi, \xi, Dr), \mathit{employee}(Dr, \xi, \xi, \xi)\}$. Since condition in step (a) holds for the two atoms of q_2, ID-rewrite suitably executes the algorithm reduce that returns the query q_3 having the form $q(Dr) \leftarrow \mathit{employee}(Dr, \xi, \xi, Dr)$, which is added to Q'. Since there are no other IDs applicable to atoms in Q', ID-rewrite returns the following UCQ:

$$\begin{aligned} q(Dr) &\leftarrow \mathit{department}(C, Dr), \mathit{employee}(Dr, S, D, B) \\ q(Dr) &\leftarrow \mathit{manager}(Dr, D'), \mathit{employee}(Dr, S, D, B) \\ q(Dr) &\leftarrow \mathit{employee}(N, S, D, Dr), \mathit{employee}(Dr, S', D', B) \\ q(Dr) &\leftarrow \mathit{employee}(Dr, S, D, Dr) \end{aligned}$$

where, for the sake of clarity, the symbols ξ have been replaced by the new fresh variable symbols $B, C, C'D, D', N, S, S'$. ∎

Example 6.1.1 (contd.) Let us assume that only the inclusion dependency $\mathit{team}[\mathit{Tleader}] \subseteq \mathit{player}[\mathit{Pcode}]$, is expressed on the global schema $\mathcal{G}_0$, and disregard the KDs and the ED. Such inclusion dependency is applicable to the unique atom in the user query $q(X) \leftarrow \mathit{player}(X, Y, Z)$. By running the algorithm ID-rewrite, we obtain the following UCQ:

$$\begin{aligned} q(X) &\leftarrow \mathit{player}(X, Y, Z) \\ q(X) &\leftarrow \mathit{team}(V, W, X) \end{aligned}$$

■

For ease of exposition, in the following we denote as Π_{ID} the UCQ returned by $\mathsf{ID\text{-}rewrite}(\mathcal{RS}, Q)$.

Theorem 6.3.4 *Let $\mathcal{RS} = \langle \Psi, \Sigma_I \rangle$ be a relational schema, where Σ_I is a set of inclusion dependencies, and let Q be a UCQ over $\mathcal{RS}$. Then, Π_{ID} is a perfect rewriting of Q w.r.t. $\mathcal{RS}$.*

Proof. In the proof we make use of the following definition: let q be a conjunctive query of arity n, and let $\bar{t}$ be an n-tuple; a set of facts G is an *image of* $q(\bar{t})$ if there exists a substitution σ such that $\sigma(head(q)) = q(\bar{t})$ and $\sigma(body(q)) = G$. It is easy to see that a tuple $\bar{t}$ belongs to the evaluation of the query q over a database instance $\mathcal{D}$, i.e., $\bar{t} \in q^{\mathcal{D}}$, iff there exists $G \subseteq \mathcal{D}$ such that G is an image of $q(\bar{t})$.

First, we prove that the algorithm is sound, i.e., for each $q \in \Pi_{ID}$ and for each instance $\mathcal{D}$, $q^{\mathcal{D}} \subseteq ans_s(Q, \mathcal{RS}, \mathcal{D})$. The proof is by induction on the structure of the set Π_{ID}, which can be seen as inductively defined from the initial set Q by the two operations (a) and (b) in the algorithm in Figure 6.1. Let $\mathcal{D}$ be a database for $\mathcal{RS}$. The base step is immediately proven since, for each $q \in Q$ and for each $\mathcal{D}$, $q^{\mathcal{D}} \subseteq ans_s(Q, \mathcal{RS}, \mathcal{D})$. As for the inductive step, let q be a conjunctive query such that $q^{\mathcal{D}} \subseteq ans_s(Q, \mathcal{RS}, \mathcal{D})$ for each instance $\mathcal{D}$ (inductive hypothesis), and let q_1 be a query obtained from q by applying step (a) of the algorithm. Then, $q_1 = \mathsf{reduce}(q, g_1, g_2)$, i.e., there exist two atoms $g_1, g_2 \in body(q)$ such that g_1 and g_2 unify. Then, if there exists a tuple $\bar{t}$ and a set of facts $G \subseteq \mathcal{D}$ such that G is an image of $q_1(\bar{t})$, then G is also an image of $q(\bar{t})$. Consequently, $q_1^{\mathcal{D}} \subseteq q^{\mathcal{D}}$, which implies that $q_1^{\mathcal{D}} \subseteq ans_s(Q, \mathcal{RS}, \mathcal{D})$. Under the same inductive hypothesis, now let q_1 be a query obtained from q by applying step (b) of the algorithm. Suppose $\bar{t} \in q_1^{\mathcal{D}}$. Let g be the atom of q transformed by step (b) of the algorithm, i.e., $q_1 = \mathsf{atom\text{-}rewrite}(q, g, I)$. It is immediate to see that there exists a fact in $\mathcal{D}$ to which the ID chase rule triggered by I is applicable[7], which implies that there exists an image of $q(\bar{t})$ in $chase(\mathcal{RS}, \mathcal{D})$. Therefore, $\bar{t} \in ans_s(q, \mathcal{RS}, \mathcal{D})$, which implies that $q_1^{\mathcal{D}} \subseteq ans_s(q, \mathcal{RS}, \mathcal{D})$, consequently $q_1^{\mathcal{D}} \subseteq ans_s(Q, \mathcal{RS}, \mathcal{D})$.

Then, we prove completeness of the algorithm. Let $\mathcal{D}$ be a database and $\bar{t}$ an n-tuple such that $\bar{t} \in ans_s(Q, \mathcal{RS}, \mathcal{D})$. We prove that there exists $q \in \Pi_{ID}$ such that $\bar{t} \in q^{\mathcal{D}}$. First, since $\bar{t} \in ans_s(Q, \mathcal{RS}, \mathcal{D})$, it follows that there exists $q_0 \in Q$ such that $\bar{t} \in ans_s(q_0, \mathcal{RS}, \mathcal{D})$. Therefore, from Lemma 5 of [101], it follows that there exists a finite number k such that there is an image G_k of $q_0(\bar{t})$ in $chase_k(\mathcal{RS}, \mathcal{D})$, where $chase_k(\mathcal{RS}, \mathcal{D})$ represents the database obtained from $\mathcal{D}$ after k applications of the ID chase rule. Moreover, without loss of generality we can assume that every application of the ID chase rule is necessary in order to generate such an image G_k: i.e., $chase_k(\mathcal{RS}, \mathcal{D})$ can be seen as a forest (set of trees) where: (i) the roots correspond to the facts of $\mathcal{D}$; (ii) there are exactly k edges, where each edge corresponds to an application of the ID chase rule; (iii) each leaf is either a fact in $\mathcal{D}$ or a fact in G_k. In the following, for each i such that $0 \leq i \leq k$, we denote as G_i the *pre-image* of $q_0(\bar{t})$ in $chase_i(\mathcal{RS}, \mathcal{D})$, i.e.,

$G_i = \{f \in chase_i(\mathcal{RS}, \mathcal{D}) \mid$ there exists $f' \in G_k$ s.t. f is an ancestor of f' in

[7] See Section 5.2.1 for the definition of ID chase rule.

$chase_k(\mathcal{RS}, \mathcal{D})$ and in $chase_i(\mathcal{RS}, \mathcal{D})$ there exists no successor of f which is an ancestor of f' in $chase_k(\mathcal{RS}, \mathcal{D})\}$

Now we prove that, starting from G_k, we can "go back" through the rule application and find a query q in Π_{ID} such that the pre-image G_0 of $q_0(\bar{t})$ in $chase_0(\mathcal{RS}, \mathcal{D}) = \mathcal{D}$ is also an image of $q(\bar{t})$, i.e., such that $\bar{t} \in q^{\mathcal{D}}$. To this aim, we prove that there exists $q \in \Pi_{ID}$ such that G_0 is an image of $q(\bar{t})$ and $|body(q)| = |G_0|$. The proof is by induction on the structure of $chase_k(\mathcal{RS}, \mathcal{D})$.

Base step: There exists $q' \in \Pi_{ID}$ such that G_k is an image of $q'(\bar{t})$ and $|body(q')| = |G_k|$. This is an immediate consequence of the facts that: (i) $q_0 \in \Pi_{ID}$; (ii) Π_{ID} is closed with respect to step (a) of the algorithm. Indeed, if $|G_k| < |body(q_0)|$ then there exist two atoms $g_1, g_2 \in body(q_0)$ and a fact f in G_k such that f and g_1 unify and f and g_2 unify, which implies that g_1 and g_2 unify, therefore by step (a) of the algorithm it follows that there exists a query $q_1 \in \Pi_{ID}$ (with $q_1 = \mathsf{reduce}(q_0, g_1, g_2)$) such that G_k is an image of $q_1(\bar{t})$ and $|body(q_1)| = |body(q_0)| - 1$. Now, if $|G_k| < |body(q_1)|$, we can iterate the above proof, thus we conclude that there exists $q' \in \Pi_{ID}$ such that G_k is an image of $q'(\bar{t})$ and $|body(q')| = |G_k|$.

Inductive step: suppose that there exists $q \in \Pi_{ID}$ such that G_i is an image of $q(\bar{t})$ and $|body(q)| = |G_i|$. Let $I = r_1[\mathbf{X}] \subseteq r_2[\mathbf{Y}]$ be the ID applied to $chase_{i-1}(\mathcal{RS}, \mathcal{D})$ to obtain $chase_i(\mathcal{RS}, \mathcal{D})$. Since non-propagated attributes in r_2 contain new constants, i.e., constants not appearing elsewhere in G_i, and since $|body(q)| = |G_i|$, it follows that the corresponding variables of the atom r_2 in $body(q)$ are unbound variables (represented by ξ); moreover, if $X_i = X_j$ then $r_2[Y_i] = r_2[Y_j]$, by the ID chase rule. Therefore, there exists an atom $g = r_2(\overline{Z}) \in body(q_i)$ such that I is applicable to g. Therefore, by step (b) of the algorithm it follows that there exists a query $q_1 \in \Pi_{ID}$ (with $q_1 = \mathsf{atom\text{-}rewrite}(q, g, I)$) such that G_{i-1} is an image of $q_1(\bar{t})$. Now, there are two possible cases: either $|body(q_1)| = |G_{i-1}|$, and in this case the thesis follows; or $|body(q_1)| = |G_{i-1}| + 1$. This last case arises if and only if the fact of the form $f = r_1(\bar{t}')$ to which the ID chase rule is applied is both in G_{i-1} and in G_i. This implies that there exist two atoms $g_1, g_2 \in body(q_1)$ such that f and g_1 unify and f and g_2 unify, hence g_1 and g_2 unify, therefore by step (a) of the algorithm (applied to q_1) it follows that there exists $q_2 \in \Pi_{ID}$ (with $q_2 = \mathsf{reduce}(q_1, g_1, g_2)$) such that G_{i-1} is an image of $q_2(\bar{t})$ and $|body(q_2)| = |G_{i-1}|$, which proves the thesis. □

Data Integration Setting

We now go back to the data integration setting. According Theorem 6.3.4, the algorithm ID-rewrite provides a perfect rewriting Π_{ID} of a UCQ Q with respect to the global schema $\mathcal{G}$. However, since data are stored at the sources, we are interested in reformulating the user query in terms of the source relations in $\mathcal{S}$ rather than in another query on the global relations in $\mathcal{G}$. To this aim, we make use of the mapping $\mathcal{M}$. Our use of $\mathcal{M}$ corresponds in principle to *unfold*, i.e., replace, each atom in Π_{ID} with a query over the sources. More precisely, we define the non-recursive Datalog$^\neg$ program $\Pi_{\mathcal{M}} = \{q_{\mathcal{S}} \mid \langle r_{\mathcal{G}}, q_{\mathcal{S}} \rangle \in \mathcal{M}\}$, and use it to produce the perfect rewriting of Q.

Theorem 6.3.5 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system and let Q be a user query over $\mathcal{I}$. Then, $\Pi_{ID} \cup \Pi_{\mathcal{M}}$ is a perfect rewriting of q w.r.t. $\mathcal{I}$.*

Proof. It is easy to see that the evaluation of $\Pi_{\mathcal{M}}$ over each source database $\mathcal{D}$ for $\mathcal{I}$ produces the retrieved global database $ret(\mathcal{I}, \mathcal{D})$. Then, the thesis follows immediately from Theorem 6.3.4. □

Example 6.1.1 (contd.) Let us now to take into account the mapping $\mathcal{M}_0$. The perfect rewriting for the query Q_0 w.r.t. $\mathcal{I}_0$, is as follows

$$\begin{aligned} q(X) &\leftarrow player(X, Y, Z) \\ q(X) &\leftarrow team(V, W, X) \\ player(X, Y, Z) &\leftarrow s_1(X, Y, Z, W) \\ team(X, Y, Z) &\leftarrow s_2(X, Y, Z) \\ team(X, Y, Z) &\leftarrow s_3(X, Y, Z) \\ coach(X, Y, Z) &\leftarrow s_4(X, Y, Z) \end{aligned}$$

■

6.3.2 Query Rewriting in the presence of IDs, KDs, and EDs

Let us now consider the general setting in which NKCIDs, KDs, and EDs are expressed on the global schema. As for the setting of a single database addressed in Section 5.2.1, in the case of a NKC data integration system, we can apply the same technique developed for IDs alone, provided that data at the sources are consistent with the key and the exclusion dependencies on the global schema, i.e., the retrieved global database $ret(\mathcal{I}, \mathcal{D})$ does not violate the key dependencies, and the closure of EDs w.r.t. to implication by IDs and EDs.

Theorem 6.3.6 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ with $\mathcal{G} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ be a NKC data integration system, $\mathcal{D}$ is a source database for $\mathcal{I}$ such that $ret(\mathcal{I}, \mathcal{D})$ is consistent with $\Sigma_K \cup \Sigma_E^*$, and Q is a user query over $\mathcal{I}$. Then, $(\Pi_{ID} \cup \Pi_{\mathcal{M}})^{\mathcal{D}} = ans_s(Q, I, \mathcal{D})$.*

Proof. Follows directly from Theorems 5.2.16 and 6.3.4. □

Observe, however, that, once $ret(\mathcal{I}, \mathcal{D})$ is computed, the unfolding step becomes unnecessary, since it is possible to obtain the answers to Q by directly evaluating Π_{ID} over $ret(\mathcal{I}, \mathcal{D})$. We will see in Chapter 9, that this will be exploited in the implementation of the algorithm.

Conversely, when data at the sources are such that $ret(\mathcal{I}, \mathcal{D})$ is not consistent $\Sigma_K \cup \Sigma_E^*$, then, under the strictly-sound semantics, any tuple is in the answer to any query, i.e., the answer to the query is meaningless.

Notice that $(\Pi_{ID} \cup \Pi_{\mathcal{M}})^{\mathcal{D}} = ans_s(Q, I, \mathcal{D})$ only for those source databases $\mathcal{D}$ for which $ret(\mathcal{I}, \mathcal{D})$ is consistent with $\Sigma_K \cup \Sigma_E^*$, hence we cannot say that $(\Pi_{ID} \cup \Pi_{\mathcal{M}})$ is a perfect rewriting for Q. Nonetheless, as shown in [30], in order to obtain a perfect rewriting of Q, it is sufficient to augment $\Pi_{ID} \cup \Pi_{\mathcal{M}}$ with another set of rules that, evaluated over $\mathcal{D}$, return all the tuples of length n, where n is the arity of Q, in the case in which $ret(\mathcal{I}, \mathcal{D})$ is not consistent with $\Sigma_K \cup \Sigma_E^*$. Since, as we will show in the next section, in this case we resort to query answering under the loosely-sound semantics, and compute consistent answers to the query rather than meaningless answers, we prefer here explicitly distinguish between the case in which retrieved

data are consistent and the case in which they are not, and refer the reader to [30] for more details on the perfect rewriting under the strictly-sound semantics.

6.4 Query Rewriting under the Loosely-sound Semantics

We now address the problem of query answering under the loosely-sound semantics. Specifically, we present a sound and complete rewriting technique to compute certain answers to queries posed to NKC systems.

Our method relies on Theorem 6.2.3 stating that for NKC systems it is possible to first deal with inclusion dependencies and then with key and exclusion dependencies, provided the computation of Σ_E^*: actually, for the IDs we exploit the algorithm $\mathsf{ID\text{-}rewrite}(\mathcal{RS}, Q)$, whereas for the other dependencies we make use of a set of Datalog$^\neg$ rules, constructed according to KDs and EDs in Σ_K and Σ_E^*. Intuitively, such rules, together with the rules in the mapping, allow us to compute the maximal subsets of $ret(\mathcal{I}, \mathcal{D})$ that are consistent with $\Sigma_K \cup \Sigma_E^*$, in such a way that we can then evaluate the program Π_{ID} produced by the algorithm ID-rewrite over each such subset.

For the sake of clarity, we first consider the case of IDs and KDs, and then the general setting. In this case, we define a Datalog$^\neg$ program Π_{KD} that, for each relation $r \in \mathcal{G}$, comprises the rules

$$\begin{aligned}
r(\vec{\mathbf{x}}, \vec{\mathbf{y}}) &\leftarrow r_{\mathcal{D}}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \ , \ not\ \overline{r}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \\
\overline{r}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) &\leftarrow r_{\mathcal{D}}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \ , \ r(\vec{\mathbf{x}}, \vec{\mathbf{z}}) \ , \ Y_1 \neq Z_1 \\
&\cdots \\
\overline{r}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) &\leftarrow r_{\mathcal{D}}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \ , \ r(\vec{\mathbf{x}}, \vec{\mathbf{z}}) \ , \ Y_m \neq Z_m
\end{aligned}$$

where: in $r(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ the variables in $\vec{\mathbf{x}}$ correspond to the attributes constituting the key of the relation r; $\vec{\mathbf{y}} = Y_1, \ldots, Y_m$ and $\vec{\mathbf{z}} = Z_1, \ldots, Z_m$[8].

Informally, for each relation r, Π_{KD} contains (i) a relation $r_{\mathcal{D}}$ that represents $r^{ret(\mathcal{I},\mathcal{D})}$; (ii) a relation r that represents a subset of $r^{ret(\mathcal{I},\mathcal{D})}$ that is consistent with the KD for r; (iii) an auxiliary relation $\overline{r}$. The above rules force each stable model M of Π_{KD}, i.e., $M \in \mathrm{SM}(\Pi_{KD})$, to be such that r^M is a maximal subset of tuples from $r^{ret(\mathcal{I},\mathcal{D})}$ that are consistent with the KD for r.

Let us now consider the case in which also exclusion dependencies are specified on the global schema, i.e., $\mathcal{G} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$. We have now to force each stable model M of our program to be such that r^M is a maximal subset of tuples from $r^{ret(\mathcal{I},\mathcal{D})}$ consistent with both the KD for r and the EDs in Σ_E^* that involve r.

To this aim, for each exclusion dependency $(r[\mathbf{A}] \cap\ s[\mathbf{B}]) = \emptyset$ in Σ_E^* we add to Π_{KD} the set of rules:

$$\begin{aligned}
\overline{r}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) &\leftarrow r_{\mathcal{D}}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \ , \ s(\vec{\mathbf{x}}, \vec{\mathbf{z}}) \ , \\
\overline{s}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) &\leftarrow s_{\mathcal{D}}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \ , \ r(\vec{\mathbf{x}}, \vec{\mathbf{z}}) \ ,
\end{aligned}$$

[8] Without loss of generality we assume that the attributes in the key precede all other attributes in r.

where in $r(\vec{\mathbf{x}}, \vec{\mathbf{z}})$ the variables in $\vec{\mathbf{x}}$ correspond to the sequence of attributes $\mathbf{A}$ of r, and in $s(\vec{\mathbf{x}}, \vec{\mathbf{z}})$ the variables in $\vec{\mathbf{x}}$ correspond to the sequence of attributes $\mathbf{B}$ of s[9]. Let us call Π_{ED} such set of rules.

Then, we consider the Datalog$^\neg$ program $\Pi = \Pi_{ID} \cup \Pi_{\mathcal{MD}} \cup \Pi_{KD} \cup \Pi_{ED}$, where Π_{ID} is obtained through $\mathsf{ID\text{-}rewrite}(\mathcal{RS}, q)$, and $\Pi_{\mathcal{MD}}$ is obtained from $\Pi_{\mathcal{M}}$ by replacing each symbol r with $r_{\mathcal{D}}$.

Example 6.1.1 (contd.) According to the above discussion, the program Π in our ongoing example, is as follows:

$$
\begin{aligned}
q(X) &\leftarrow player(X,Y,Z)\\
q(X) &\leftarrow team(V,W,X)\\
player_{\mathcal{D}}(X,Y,Z) &\leftarrow s_1(X,Y,Z,W)\\
team_{\mathcal{D}}(X,Y,Z) &\leftarrow s_2(X,Y,Z)\\
team_{\mathcal{D}}(X,Y,Z) &\leftarrow s_3(X,Y,Z)\\
coach_{\mathcal{D}}(X,Y,Z) &\leftarrow s_4(X,Y,Z)\\
player(X,Y,Z) &\leftarrow player_{\mathcal{D}}(X,Y,Z)\ ,\ not\ \overline{player}(X,Y,Z)\\
\overline{player}(X,Y,Z) &\leftarrow player(X,V,W)\ ,\ player_{D}(X,Y,Z)\ ,\ Y \neq V\\
\overline{player}(X,Y,Z) &\leftarrow player(X,V,W)\ ,\ player_{D}(X,Y,Z)\ ,\ Z \neq W\\
team(X,Y,Z) &\leftarrow team_{\mathcal{D}}(X,Y,Z)\ ,\ not\ \overline{team}(X,Y,Z)\\
\overline{team}(X,Y,Z) &\leftarrow team(X,V,W)\ ,\ team_{\mathcal{D}}(X,Y,Z)\ ,\ Y \neq V\\
\overline{team}(X,Y,Z) &\leftarrow team(X,V,W)\ ,\ team_{\mathcal{D}}(X,Y,Z)\ ,\ Z \neq W\\
coach(X,Y,Z) &\leftarrow coach_{\mathcal{D}}(X,Y,Z)\ ,\ not\ \overline{coach}(X,Y,Z)\\
\overline{coach}(X,Y,Z) &\leftarrow coach(X,V,W)\ ,\ coach_{D}(X,Y,Z)\ ,\ Y \neq V\\
\overline{coach}(X,Y,Z) &\leftarrow coach(X,V,W)\ ,\ coach_{D}(X,Y,Z)\ ,\ Z \neq W\\
\overline{player}(X,Y,Z) &\leftarrow player_{\mathcal{D}}(X,Y,Z)\ ,\ coach(X,V,W)\\
\overline{coach}(X,Y,Z) &\leftarrow coach_{\mathcal{D}}(X,Y,Z)\ ,\ team(V,W,X)\\
\overline{coach}(X,Y,Z) &\leftarrow coach_{\mathcal{D}}(X,Y,Z)\ ,\ player(X,V,W)\\
\overline{team}(X,Y,Z) &\leftarrow team_{\mathcal{D}}(X,Y,Z)\ ,\ coach(Z,V,W)
\end{aligned}
$$

where Π_{ID} is constituted by the first two rules, $\Pi_{\mathcal{MD}}$ by the rules ranging from the 3rd to the 6th, Π_{KD} by the rules ranging from the 7th to the 15th, and Π_{ED} by the last four rules. Notice that in the rules of Π_{ED} we considered also the exclusion dependency $team[Tleader] \cap coach[Ccode] = \emptyset$ that is a logical consequence of the ID and the EDs expressed on $\mathcal{G}_0$. ∎

We are now able to prove soundness and completeness of our technique with respect to the problem of computing the certain answer to a user query under the loosely-sound semantics.

Theorem 6.4.1 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a NKC system, and Q be a query over $\mathcal{G}$. Then, $\Pi_{KD} \cup \Pi_{ED} \cup \Pi_{ID} \cup \Pi_{\mathcal{MD}}$ is a perfect rewriting of q w.r.t. $\mathcal{I}$.*

[9] Without loss of generality, we assume that $\mathbf{A}$ and $\mathbf{B}$ precede all other attributes in r and s, respectively.

6.5 Discussion and Related Work

Discussion In this chapter we have presented techniques for query processing in GAV data integration systems under the strictly-sound and the loosely-sound semantics, in the relational setting, when inclusion, key, and exclusion dependencies are expressed on the global schema. We underline that our approach is strongly intensional, since treatment of constraints is carried out at the query and schema level, and query answering is solved by first computing the perfect rewriting of a query issued on the global schema, and then evaluating the rewritten query. As we will see in Chapter 9, this allowed us to realize a prototype implementing our algorithms where components for query processing are decoupled from the extensional layer devoted to the extraction of the data from the sources. With regard to query answering under the strictly-sound semantics, the feasibility of our approach is also witnessed by the fact that, since perfect rewritings are expressed in terms of union of conjunctive queries, they can be easily handled by standard database query evaluation techniques. Furthermore, the tractability of our algorithms can be easily proved, since evaluating a UCQ over a database is in PTIME in data complexity. Finally, our overall technique resorts to more complex rewritings expressed in Datalog$^{\neg}$ only when needed, i.e., only in the case in which data at the sources are inconsistent with KDs or EDs, and classical methods developed under the strict semantics would not be able to provide significative answers to queries.

Related work Several works in the literature address the problem of query answering in data integration in the presence of integrity constraints on the global schema. In this respect, query rewriting under integrity constraints has been first studied in the LAV setting. In particular, the inverse rules algorithm, that we briefly described in its basic version in Section 4.1, has been suitably extended by its author in order to deal with functional dependencies expressed over the global schema [61]. The algorithm computes the perfect rewriting in the case of queries and mapping expressed in terms of conjunctive queries, and "maximally contained" rewritings in the case of recursive queries.

Then, in [90] Gryz analyzes query rewriting under inclusion dependencies in LAV systems. Moreover, a method is presented which is able to deal simultaneously with acyclic IDs and functional dependencies. It is worth noticing that, in that paper, an algorithm for computing the rewriting of a conjunctive query in a database with inclusion dependencies is presented. The algorithm is based on a brilliant intuition, i.e., rewriting a conjunctive query by computing the rewriting of each atom in a way "almost independent" of the other atoms. This can be obtained if the body of the initial query q is preliminarily transformed ("minimized"). However, we have found out that Gryz's algorithm does not actually compute the perfect rewriting, in the sense that some conjunctions of the perfect rewriting are missing. Indeed, the algorithm ID-rewrite presented in Section 6.3.1 is certainly inspired by Gryz's main intuition, but overcomes the above mentioned incompleteness through a new technique for generating the conjunctions of the rewriting.

In GAV integration systems, the role played by integrity constraints for query processing has been in general overlooked. A first attempt to deal with integration under constraints in

the GAV approach has been addressed in [24]. Compared to our approach, this work is able to handle a more restricted class of integrity constraints (KDs and foreign key dependencies). Moreover, such an approach is technically more involved than the one presented in this paper, since it makes use of Skolem functions and SLD-refutation for computing the perfect rewriting of a CQ[10].

Moreover, [69] presents an approach for dealing with integrity constraints in the context of data exchange, where the mapping is GLAV (a generalization of LAV and GAV). However, in data exchange the global schema (called *target schema*) has to be materialized, therefore the class of dependencies that is considered in this work (and that is incomparable with the classes we consider in this paper) is such that the chase of the retrieved global database is always finite.

We point out that all the above mentioned approaches do not address the problem of inconsistent data, hence they can be compared only with our algorithms for query answering under the strictly-sound semantics.

As already said, works that deal with data inconsistencies have been developed mainly in a context of a single database. Furthermore, techniques for computing consistent answers proposed so far in the literature are able to deal only with universally quantified constraints, or treat existentially quantified constraints under restrictive assumptions [7, 86, 9]. Furthermore, they consider repair semantics that are different from the loosely-sound semanitcs considered in this chapter. A more detailed description of these works has been presented in Section 5.5.

More recently, data inconsistency in a purely data integration setting has been studied in [13] and [16]. Differently from our approach, both these papers address a LAV framework and hence their results cannot be directly compared with the technique described in this chapter. Furthermore, as widely discussed in Section 3.4, the semantics proposed in [13] and [16] is different from the loose semantics for LAV introduced in Section 3.3.

More in details, [13] extends to LAV integration systems the method for consistent query answering originally presented in [7] in a single database context. Such result is achieved by combining the query reformulation technique of [7] with the Duschka inverse rules algorithm [61], that is extended here to the case in which the query over the global database is a non-recursive Datalog$^{\neg}$ query. The obtained rewriting is shown to be a maximally contained rewriting. As in [7], only binary universally quantified constraints and non-existentially quantified global queries are considered.

Conversely, [16] takes into account a very general class of constraints and queries, namely almost all constraints expressed as first order logic formulae, and stratified Datalog$^{\neg}$ queries. However, the authors introduce the assumption that the constants used for repairing are only the constants in the active domain, i.e., the constants occurring in the source database. Hence, they consider only the cases in which the chase is finite. Repairs in [16] can be obtained from the stable models of a suitable Datalog$^{\vee,\neg}$ program, i.e., Datalog$^{\neg}$ where disjunction is allowed in the head of the rules.

In conclusion, we point out that none of the above mentioned works provides a solution to the query answering problem for the case of cyclic inclusion dependencies under the semantics (both strict and loose) considered in this thesis.

[10] See Section 4.2.2 for more details on this paper.

Chapter 7

Query Answering and Rewriting in LAV Data Integration Systems

Whereas in Chapter 6 we have addressed the data integration framework in which the mapping is given in the GAV approach, we now focus our attention on LAV integration systems, and provide an in-depth study of the problem of query answering in such a framework. As usual, we consider integration systems in which inclusion, key and exclusion dependencies are issued on the global schema and user queries are union of conjunctive queries. differently from the GAV case, queries in the mapping in our LAV framework are conjunctive queries.

The main contributions of this chapter are the following:

1. we show that query answering is undecidable in LAV systems in the presence of 1KCIDs w.r.t. KDs specified over the global schema for the strictly-sound, loosely-sound and loosely-exact semantics;

2. we extend to LAV systems some complexity results for query answering given in Chapter 5 in the setting of a single database;

3. we provide actual methods for query answering under integrity constraints for the strictly-sound semantics;

4. we provide a sound and complete query rewriting technique under the strictly-sound semantics, for the case of IDs and EDs. More precisely, we define an off-line compiling technique that, starting from a LAV system specification, produces the specification of a GAV system which is query-equivalent to the initial LAV system. After such a compilation, we can reuse the query processing technique defined for GAV systems also in the LAV case.

We point out that, analogously to the the GAV setting, our technique allows for a modular treatment of *(i)* the integration systems in the presence of integrity constraints, through the compilation of LAV into GAV, and of *(ii)* KDs and IDs, through the separability property of NKC dependencies. Furthermore, the compilation of a LAV system into a GAV one can be executed off-line, only at each change of the system specification. Based on such a modularity,

we exploit both off-line and intensional processing of queries in order to provide an effective approach to query processing, as witnessed by the implementation described in Chapter 9.

7.1 Decidability and Complexity of Query Answering

We extend now to LAV integration systems some results originally provided in Chapter 5 in the framework of a single relational schema. To this aim, we first reduce query answering in this latter setting to query answering in a LAV integration system, as done in Section 6.2 for GAV systems. The only difference here is in the construction of the mapping, which is now given by n assertions of the form $\langle s_i, s_i(X_1, \ldots, X_k) \leftarrow g_i(X_1, \ldots, X_k)\rangle$, for each i, $1 \leq i \leq n$.

On the basis of this reduction, we can easily prove that query answering is undecidable for 1KC LAV integration systems under the strictly-sound, the loosely-sound and the loosely-exact semantics. Moreover, we are able to extend to NKC LAV integration systems all hardness results provided in Chapter 5.

Now, the question arises of whether the separation property between NKCIDs and KDs holds for LAV systems, i.e., whether the analogous of Theorem 6.2.3 exists for the LAV framework. Furthermore, we would also establish the complexity classes for which the query answering problem for LAV systems is complete under the different semantics for different combinations of integrity constraints on the global schema.

In Section 6.2, we introduced the notion of retrieved global database that allowed us to solve query answering in a GAV system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M}\rangle$ essentially by means of the query answering algorithms of Chapter 5 applied to the global schema $\mathcal{G}$, once provided the computation of the retrieved global database. Hence, we have been able to extend to GAV systems the separation property originally stated in Theorem 5.2.16, and generalize to such framework all the complexity results provided in Chapter 5 for a single relational schema. Actually, the complexity of query answering over GAV integration systems coincides with the complexity of query answering over a relational schema, modulo the complexity of computing the retrieved global database $ret(\mathcal{I}, \mathcal{D})$.

Unfortunately, there does not exist a particular global database that plays for LAV systems the same role played by the retrieved global database for GAV systems. This can be easily seen, since LAV mappings provide in general only partial information about the data that satisfy the global schema, and several ways of populating the global schema according to the mapping may exist (even if the mapping is considered exact).

However, even if it is not possible to characterize query answering in LAV under the strict and loose semantics as done for the GAV systems, we are able to provide a definition of retrieved global database in LAV that is specific for the strictly-sound semantics, and that allow us to show that a separation theorem also holds for NKC LAV systems.

In order to achieve this final result, in the next subsection we first study query answering under the strictly-sound semantics in the presence of only IDs, then we consider IDs and EDs, and finally, we address the general setting of NKCIDs, KDs and EDs.

7.1.1 Query Answering under the Strictly-sound Semantics

Our plan of attack is as follows. First, we consider LAV systems without constraints, and define the notion of retrieved global database for this kind of systems. We show that, analogously to the GAV case, in LAV systems without constraints the answer to a user query Q can be calculated by evaluating Q over the retrieved global database. With this notion in place, we can apply the results of Section 5.2, which show that, in the presence of IDs, the answer to a query Q over a database $\mathcal{DB}$ is computed by evaluating Q over the chase of $\mathcal{DB}$. We then consider the case of IDs and EDs, and show that the notion of retrieved global database still allows for query answering in this setting. Finally, for the case in which also KDs are specified on the global schema a different definition of retrieved global database will be taken into account.

In order to distinguish between the retrieved global database of GAV systems and LAV systems, hereinafter we denote the former with ret_G and the latter with ret_L.

Definition 7.1.1 Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a LAV data integration system, and $\mathcal{D}$ a source database for $\mathcal{I}$. The *retrieved global database* $ret_L(\mathcal{I}, \mathcal{D})$ is defined constructively as follows. Consider a mapping assertion $\langle s, V_s \rangle \in \mathcal{M}$ where $s \in \mathcal{S}$ and V_s is of the form

$$s(\vec{\mathbf{x}}, \vec{\mathbf{c}}) \leftarrow conj(\vec{\mathbf{x}}, \vec{\mathbf{e}}, \vec{\mathbf{y}})$$

where $conj$ is a conjunction of predicates, $\vec{\mathbf{x}} = X_1, \ldots, X_{n_d}$ is a sequence of n_d distinguished variables, $\vec{\mathbf{c}} = c_1, \ldots, c_{n_{c,h}}$ and $\vec{\mathbf{e}} = e_1, \ldots, e_{n_c}$ are constants of the domain $\mathcal{U}$, and $\vec{\mathbf{y}} = Y_1, \ldots, Y_{n_n}$ is a sequence of n_n non-distinguished variables. Without loss of generality, and for the sake of clearness of presentation, we assume that, for all predicates in $conj$, the symbols (when present) appear in the following order: first the distinguished variables, then the constants, and finally the non-distinguished variables.

To each tuple $(a_1, \ldots, a_{n_d}, c_1, \ldots, c_{n_{c,h}}) \in s^{\mathcal{D}}$ we associate the constant a_i to X_i, and a fresh constant z_i of $\mathcal{U}$ (not appearing elsewhere in the database, and that has never been introduced in the construction) to Y_i. Then, for each atom $g \in conj$ of the form $g(X_1, \ldots, X_k, e_1, \ldots, e_\ell, Y_1, \ldots, Y_m)$, where $\{X_1, \ldots, X_k\} \subseteq \vec{\mathbf{x}}$, $\{e_1, \ldots, e_\ell\} \subseteq \vec{\mathbf{e}}$, and $\{Y_1, \ldots, Y_m\} \subseteq \vec{\mathbf{y}}$, we add the fact $g(a_1, \ldots, a_k, e_1, \ldots, e_\ell, z_1, \ldots, z_m)$ in $ret_L(\mathcal{I}, \mathcal{D})$. Note that the join variables in the body of the view are assigned with the same fresh constants. Moreover, the construction process is finite.

Lemma 7.1.2 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a LAV data integration system, with $\mathcal{G} = \langle \Psi, \emptyset \rangle$ and $\mathcal{D}$ a source database for $\mathcal{I}$. We have that for any database $\mathcal{B} \in sem_s(\mathcal{I}, \mathcal{D})$ there exists a well-defined homomorphism from $ret_L(\mathcal{I}, \mathcal{D})$ to $\mathcal{B}$ that sends constants of the active domain $\mathcal{U}_\mathcal{D}$ to themselves and fresh constants to constants of $\mathcal{U}$, and such that each fact of $g^{ret_L(\mathcal{I}, \mathcal{D})}$ is sent to a fact of $g^{\mathcal{B}}$.*

Proof. By definition of sound mapping $\mathcal{M}$, and by construction of $ret_L(\mathcal{I}, \mathcal{D})$, any database in $sem_s(\mathcal{I}, \mathcal{D})$ is forced to have a minimum set of facts $\mathcal{B}_m$, constituted by the facts obtained from $ret_L(\mathcal{I}, \mathcal{D})$ by replacing each fresh constant with some constant in $\mathcal{U}$. The homomorphism μ is easily constructed by mapping each fact of $ret_L(\mathcal{I}, \mathcal{D})$ to the corresponding fact of $\mathcal{B}_m$. □

Theorem 7.1.3 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a LAV data integration system, with $\mathcal{G} = \langle \Psi, \emptyset \rangle$; let $\mathcal{D}$ be a source database for $\mathcal{S}$, and Q a query over $\mathcal{G}$ of arity n. Moreover, let t be an n-tuple of constants of $\mathcal{U}_\mathcal{D}$, i.e., t does not contain any of the fresh constants z_i introduced in the construction of $ret_L(\mathcal{I}, \mathcal{D})$. Then, we have that $t \in ans_s(Q, \mathcal{I}, \mathcal{D})$ iff $t \in q^{ret_L(\mathcal{I},\mathcal{D})}$.*

Proof.

"$\Rightarrow$"This is trivial because $ret_L(\mathcal{I}, \mathcal{D})$ belongs to $sem_s(\mathcal{I}, \mathcal{D})$.

"$\Leftarrow$"Consider a tuple $t \in Q^{ret_L(\mathcal{I},\mathcal{D})}$; we show that t is in $Q^\mathcal{B}$ for *any* database $\mathcal{B} \in sem_s(\mathcal{I}, \mathcal{D})$. The existence of t witnesses the existence of a query homomorphism $\varkappa$ from the atoms of Q to facts of $ret_L(\mathcal{I}, \mathcal{D})$, that sends the distinguished variables of Q to the constants of t. Consider a generic database $\mathcal{B} \in sem_s(\mathcal{I}, \mathcal{D})$; by applying Lemma 7.1.2, we know that there exists a homomorphism μ from $ret_L(\mathcal{I}, \mathcal{D})$ to $\mathcal{B}$. The composition $\varkappa \circ \mu$ is a homomorphism from the atoms of Q to $\mathcal{B}$, that preserves the image of the distinguished variables, and witnesses that $t \in ans_s(q, \mathcal{I}, \mathcal{D})$. □

Hence, in the case in which no dependency is specified on the global schema query answering in LAV is analogous to query answering over a single relational schema, modulo the computation of $ret_L(\mathcal{I}, \mathcal{D})$. Since such computation is polynomial both in data and combined complexity it easily turns out that computational complexity in LAV without constraints is in PTIME in data complexity and NP-complete in combined complexity [2].

The above Theorem can be easily extended to the case in which IDs are specified over the global schema.

Theorem 7.1.4 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a LAV data integration system, with $\mathcal{G} = \langle \Psi, \Sigma_I \rangle$; let $\mathcal{D}$ be a source database for $\mathcal{I}$, and Q a query over $\mathcal{G}$ of arity n. Moreover, let t be an n-tuple of constants of $\mathcal{U}_\mathcal{D}$ Then, we have that $t \in ans_s(Q, \mathcal{I}, \mathcal{D})$ iff $t \in Q^{chase(\mathcal{G}, ret_L(\mathcal{I},\mathcal{D}))}$.*

Proof.

"$\Rightarrow$"This follows from the fact that $chase(\mathcal{G}, ret_L(\mathcal{I}, \mathcal{D}))$ belongs to $sem_s(\mathcal{I}, \mathcal{D})$. Indeed, from Lemma 5.2.1 it follows that $chase(\mathcal{G}, ret_L(\mathcal{I}, \mathcal{D}))$ satisfies Σ_I and hence it is consistent with $\mathcal{G}$, whereas satisfaction of the mapping $\mathcal{M}$ can be easily seen from the fact that $ret_L(\mathcal{I}, \mathcal{D})$ satisfies the mapping, and adding new facts to $ret_L(\mathcal{I}, \mathcal{D})$ cannot make any mapping assertion to be violated.

"$\Leftarrow$"Consider a tuple $t \in q^{chase(\mathcal{G}, ret_L(\mathcal{I},\mathcal{D}))}$; we show that t is in $Q^\mathcal{B}$ for *any* database $\mathcal{B} \in sem_s(\mathcal{I}, \mathcal{D})$. Analogously to Theorem 7.1.3, the existence of t witnesses the existence of a query homomorphism $\varkappa$ from the atoms of Q to facts of $chase(\mathcal{G}, ret_L(\mathcal{I}, \mathcal{D}))$, that sends the distinguished variables of Q to the constants of t. Then, as in Theorem 5.2.2, it can be proved by induction on the structure of $chase(\mathcal{G}, ret_L(\mathcal{I}, \mathcal{D}))$ that, for any database instance $\mathcal{B} \in sem_s(\mathcal{G}, ret_L(\mathcal{I}, \mathcal{D}))$, there exists a homomorphism μ that sends the tuples of $chase(\mathcal{G}, ret_L(\mathcal{I}, \mathcal{D}))$ to the tuples of $\mathcal{B}$. It is also easy to see that $sem_s(\mathcal{G}, ret_L(\mathcal{I}, \mathcal{D})) = sem_s(\mathcal{I}, \mathcal{D})$. Indeed every $\mathcal{B} \in sem_s(\mathcal{G}, ret_L(\mathcal{I}, \mathcal{D})$ is consistent with $\mathcal{G}$ and contains $ret_L(\mathcal{I}, \mathcal{D})$, modulo renaming the fresh constants introduced in the construction of $ret_L(\mathcal{I}, \mathcal{D})$. Hence, each such $\mathcal{B}$ satisfies the mapping, since it contains necessarily the minimum set of facts that guarantee $s^\mathcal{B} = V_s^\mathcal{D}$, thus $sem_s(\mathcal{G}, ret_L(\mathcal{I}, \mathcal{D})) \subseteq sem_s(\mathcal{I}, \mathcal{D})$. In a similar way we can prove that

$sem_s(\mathcal{I},\mathcal{D}) \subseteq sem_s(\mathcal{G}, ret_L(\mathcal{I},\mathcal{D}))$. Hence, the composition $\varkappa \circ \mu$ is a homomorphism from the atoms of Q to $\mathcal{B} \in sem_s(\mathcal{I},\mathcal{D})$, that preserves the image of the distinguished variables, and witnesses that $t \in ans_s(q,\mathcal{I},\mathcal{D})$. □

Obviously, the above Theorem provides an actual procedure to answer a query over a LAV system in the presence of IDs on the global schema. Also in this case, the computational complexity is the same as in the framework of a single relational schema, i.e., it is in PTIME in data complexity and PSPACE-complete in combined complexity (see Section 5.4).

It should be easy to see that the above Theorem also holds in the case in which IDs and EDs are specified on the global schema, provided that $ret_L(\mathcal{I},\mathcal{D})$ is consistent with the closure of EDs w.r.t. logical implication of IDs and EDs. Intuitively, this holds since the introduction of fresh constants in $ret_L(\mathcal{I},\mathcal{D})$ cannot lead to the violation of exclusion dependencies (constants are never equal), and that ED violations can be caused only by the data stored at the sources.

Theorem 7.1.5 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a LAV data integration system, with $\mathcal{G} = \langle \Psi, \Sigma_I, \Sigma_E \rangle$; let $\mathcal{D}$ be a source database for $\mathcal{I}$, and Q a query of arity n over $\mathcal{G}$. Moreover, let t be an n-tuple of constants of $\mathcal{U}_\mathcal{D}$ Then, we have that $t \in ans_s(Q,\mathcal{I},\mathcal{D})$ iff $ret_L(\mathcal{I},\mathcal{D})$ is consistent with Σ_E^* and $t \in Q^{chase(\mathcal{G}, ret_L(\mathcal{I},\mathcal{D}))}$.*

Proof. Proceed as in Theorem 7.1.4 taking into account Lemma 5.2.4. □

From the above theorem it follows that query answering is in PTIME in data complexity and PSPACE-complete in combined complexity.

Now, we are able to answer our initial question and show that the analogous of Theorem 6.2.3 holds for LAV systems. In the following, we show that indeed a separation theorem also holds for NKC LAV systems, which in turn allows to establish decidability of query answering in such systems.

To this aim, we define the notion of *retrieved global database* for a LAV system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$. Such a database, denoted as $ret_{LK}(\mathcal{I},\mathcal{D})$, is obtained starting from the database $ret_L(\mathcal{I},\mathcal{D})$ formalized in Definition 7.1.1, by repeatedly applying the following rule that unifies fresh constants (the new constant symbols introduced in the construction of $ret_L(\mathcal{I},\mathcal{D})$):

KD-RULE: if the KD $key(r) = \mathbf{A}$ is in Σ_K and there are two facts $r(t_1), r(t_2)$ in $ret_{LK}(\mathcal{I},\mathcal{D})$ such that $t_1[\mathbf{A}] = t_2[\mathbf{A}]$, then, for each non-key attribute X of r: if $t_1[X]$ is a fresh constant z_i , then replace each occurrence of z_i with $t_2[X]$ in $ret_{LK}(\mathcal{I},\mathcal{D})$; otherwise, if $t_2[X]$ is a fresh constant z_j, then replace each occurrence of z_j with $t_1[X]$ in $ret_{LK}(\mathcal{I},\mathcal{D})$.

Informally, in the presence of key dependencies, the retrieved global database $ret_L(\mathcal{I},\mathcal{D})$ of Definition 7.1.1 is not a good representative of the data that, through the mapping, the source data in $\mathcal{D}$ force to be present in the global relations, since, due the presence of key dependencies, some of the fresh constants introduced in the construction process of $ret_L(\mathcal{I},\mathcal{D})$ could be made equal. Instead, by applying the KD-rule, we obtain a correct representative of such data, which are common to all databases in $sem_s(\mathcal{I},\mathcal{D})$.

It is immediate to verify that $ret_{LK}(\mathcal{I},\mathcal{D})$ can be computed in time polynomial in the size of $\mathcal{D}$.

Analogously to the GAV case, when $ret_{LK}(\mathcal{I}, \mathcal{D})$ is consistent with $\Sigma_K \cup \Sigma_E^*$, we are able to separately deal with Σ_I and $\Sigma_K \cup \Sigma_E$.

Theorem 7.1.6 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a NKC LAV system, where $\mathcal{G} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$, let $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G}' = \langle \Psi, \Sigma_I \rangle$, be the system obtained by $\mathcal{I}$ by eliminating the KDs and the EDs of $\mathcal{G}$; let $\mathcal{D}$ be a source database for $\mathcal{I}$ and $\mathcal{I}'$. Moreover, let Q be a query of arity n over $\mathcal{G}$ and $\mathcal{G}'$, and t an n-tuple of constants of $\mathcal{U}_\mathcal{D}$. We have that $t \notin ans_s(Q, \mathcal{I}, \mathcal{D})$ iff $ret_{LK}(\mathcal{I}, \mathcal{D})$ is consistent with $\Sigma_K \cup \Sigma_E^*$ and $t \notin ans_s(Q, \mathcal{I}', \mathcal{D})$.*

Proof. As in the proof of Theorem 7.1.4, we can show that $sem_s(\mathcal{I}, \mathcal{D}) = sem_s(\mathcal{G}, ret_{LK}(\mathcal{I}, \mathcal{D}))$. Hence, from Theorem 5.2.16 the thesis follows. □

Therefore, the above theorem implies decidability of query answering in NKC LAV systems.

7.2 Query Rewriting under the Strictly-sound Semantics

In this section we present a procedure for computing the perfect rewriting of a UCQ Q posed to a LAV integration system under the strictly-sound semantics in the presence of IDs and EDs on the global schema. The basic idea of our approach is to first compile the LAV system into an equivalent GAV one, and then make use of the algorithm developed for the GAV case.

7.2.1 Compiling LAV into GAV

We now present a technique for compiling a LAV system into an equivalent GAV one. The notion of equivalence is given in terms of queries, and it is the same as in [23]. In particular, given two integration systems $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ and $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}' \rangle$ over the same source schema $\mathcal{S}$ and such that all relations of $\mathcal{G}$ are also relations of $\mathcal{G}'$, we say that $\mathcal{I}'$ is *query-preserving* with respect to $\mathcal{I}$, if for every query Q over $\mathcal{I}$ and for every source database $\mathcal{D}$ for $\mathcal{I}$, we have that $ans_s(q, \mathcal{I}, \mathcal{D}) = ans_s(q, \mathcal{I}', \mathcal{D})$. In other words, we say that $\mathcal{I}'$ is query-preserving with respect to $\mathcal{I}$ if, for any query over the global schema of $\mathcal{I}$, and for any source database $\mathcal{D}$, the answers we get for the query on the two integration systems are identical.

Our compiling technique extends a similar procedure presented in [23], which does not allow for query processing and is not able to deal with integrity constraints. Furthermore, in [23] only mapping queries without constants are considered. Here, we improve this technique as follows:

(i) we extend the class of LAV systems that can be compiled into GAV ones, allowing the presence of IDs and EDs on the global schema;

(ii) we allow mapping queries having constants of the domain $\mathcal{U}$ appearing in the body and/or in the head;

(iii) we avoid the use of the *simple equality-generating dependencies* introduced in the transformation of [23] together with IDs: we only make use of IDs where attributes may appear more than once in the left-hand side.

We point out that the simplification indicated at point *(iii)* allows for adopting, in the compiled system, the algorithm ID-rewrite for query processing in GAV under IDs that has been described in Section 6.3.1. Indeed, such an algorithm is able to deal with also IDs with repetition in their left-hand side[1].

We now present the actual transformation of a LAV system into a query-preserving GAV one. Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be the initial LAV system, with $\mathcal{G} = \langle \Psi, \Sigma_I, \Sigma_E \rangle$, and $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}' \rangle$, with $\mathcal{G}' = \langle \Psi', \Sigma_I', \Sigma_E \rangle$, be the transformed GAV one. The transformation is performed as follows.

- The set of sources $\mathcal{S}$ remains unchanged.
- The global schema $\mathcal{G}'$ is obtained from $\mathcal{G}$ by introducing in Ψ', for each relation s/n in $\mathcal{S}$:
  (i) a new relation image_$s/(n + n_c)$, where n_c is the number of constants appearing in $body(V_s)$, where V_s is the view associated to s[2];
  (i) a new relation expand_$s/(n + n_c + n_n)$, where n_n is the number of non-distinguished variables appearing in V_s.
- The GAV mapping $\mathcal{M}'$ associates to each global relation image_s the query

$$\mathsf{image\_}s(X_1, \ldots, X_{n_d}, e_1, \ldots, e_{n_c}) \leftarrow s(X_1, \ldots, X_{n_d}, c_1, \ldots, c_{n_{c,h}})$$

  where $X_1^h, \ldots, X_{n_d}^h$ are the distinguished variables of V_s, $e_1, \ldots, e_{n_c}$ are constants in $body(V_s)$, and $c_1, \ldots, c_{n_{c,h}}$ are constants in $head(V_s)$. We do not associate any query to the remaining global relations.
- The set of inclusion dependencies Σ_I' contains the dependencies in Σ_I plus the IDs constructed as follows. We suppose to enumerate distinguished variables, constants and non-distinguished variables in V_s as $Z_1, \ldots, Z_{n_d+n_c+n_n}$, where $Z_1, \ldots, Z_{n_d}$ are distinguished variables and $Z_{n_d+1}, \ldots, Z_{n_d+n_c}$ are constants.
  (i) For each relation image_$s/(n + n_c)$ in Ψ' we add the inclusion dependency

$$\mathsf{image\_}s[1, \ldots, n + n_c] \subseteq \mathsf{expand\_}s[1, \ldots, n + n_c];$$

  (ii) for each relation expand_s in Ψ' and for each atom $g(Z_{i_1}, \ldots, Z_{i_{k+\ell+m}})$ occurring in $body(V_s)$, we add the inclusion dependency

$$\mathsf{expand\_}s[i_1, \ldots, i_{k+\ell+m}] \subseteq g[1, \ldots, k + \ell + m].$$

  Note that the inclusion dependencies introduced here may have repeated attributes on the left-hand side, since the same variable may occur more than once in each atom.
- The set of EDs Σ_E remains unchanged.

[1] Notice that the algorithm ID-rewrite presented in this thesis extends the one presented in [30] that cannot handle such form of IDs.

[2] Hereinafter V_s indicates the view that the mapping associates to s.

It is immediate to verify that, given a LAV integration system $\mathcal{I}$, the corresponding GAV integration system $\mathcal{I}'$ defined as above can be constructed in time that is linear in the size of $\mathcal{I}$.

Example 7.2.1 Consider now the LAV system $\mathcal{I}_L = \langle \mathcal{G}, \mathcal{S}_L, \mathcal{M}_L \rangle$ where the global schema coincides with the relational schema $\mathcal{RS}_1$ described in Example 6.3.3, i.e. $\mathcal{G} = \mathcal{RS}_1$[3], $\mathcal{S}_L$ comprises the two relations s_3 and s_4 and $\mathcal{M}_L$ is as follows.

$$\begin{aligned} s_3(N, Db, S) &\leftarrow \mathit{manager}(N, Db), \mathit{employee}(N, S, D, N) \\ s_4(E, B) &\leftarrow \mathit{employee}(E, S, D, B) \end{aligned}$$

Compile $\mathcal{I}_L$ into a query equivalent GAV system $\mathcal{I}_{LG} = \langle \mathcal{G}', \mathcal{S}_L, \mathcal{M}' \rangle$, where $\mathcal{G}' = \langle \Psi', \Sigma'_I \rangle$. According to the algorithm above described, Ψ' is obtained by adding to Ψ the relations $\mathsf{image\_}s_3/3$, $\mathsf{image\_}s_4/2$, $\mathsf{expand\_}s_3/3$ and $\mathsf{expand\_}s_4/4$, whereas Σ'_I is obtained by adding to $\Sigma_I = \{I_1, I_2, I_3\}$ the following inclusion dependencies

$$\begin{aligned} \mathsf{image\_}s_3[1,2,3] &\subseteq \mathsf{expand\_}s_3[1,2,3] && (I_4) \\ \mathsf{image\_}s_4[1,2] &\subseteq \mathsf{expand\_}s_4[1,4] && (I_5) \\ \mathsf{expand\_}s_3[1,2] &\subseteq \mathit{manager}[1,2] && (I_6) \\ \mathsf{expand\_}s_3[1,3,4,1] &\subseteq \mathit{employee}[1,2,3,4] && (I_7) \\ \mathsf{expand\_}s_4[1,2,3,4] &\subseteq \mathit{employee}[1,2,3,4] && (I_8). \end{aligned}$$

Moreover, the mapping $\mathcal{M}'$ is

$$\begin{aligned} \mathsf{image\_}s_3(N, Db, S) &\leftarrow s_3(N, Db, S) \\ \mathsf{image\_}s_4(E, B) &\leftarrow s_4(E, B) \end{aligned}$$

■

We now prove that our transformation is query-preserving.

Theorem 7.2.2 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a LAV integration system, with $\mathcal{G} = \langle \Psi, \Sigma_I, \Sigma_E \rangle$ and let $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}' \rangle$, with $\mathcal{G} = \langle \Psi', \Sigma'_I, \Sigma_E \rangle$ be the corresponding GAV integration system defined as above. Then $\mathcal{I}'$ is query-preserving with respect to $\mathcal{I}$, i.e., for every query Q over $\mathcal{I}$ and for every source database $\mathcal{D}$ for $\mathcal{I}$, we have that $ans_s(Q, \mathcal{I}, \mathcal{D}) = ans_s(Q, \mathcal{I}', \mathcal{D})$.*

Proof. In this proof we make use of the notion of chase, defined in Section 5.2.1.

Let us initially overlook the EDs in Σ_E, and consider the two systems $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ and $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}' \rangle$, in which $\mathcal{G} = \langle \Psi, \Sigma_I \rangle$ and $\mathcal{G}' = \langle \Psi', \Sigma'_I \rangle$. Let Q be a query over $\mathcal{G}$, and $\mathcal{D}$ a source database for $\mathcal{I}$ and $\mathcal{I}'$; we will show that $ans_s(q, \mathcal{I}, \mathcal{D}) = ans_s(q, \mathcal{I}', \mathcal{D})$.

As for the LAV system $\mathcal{I}$, from Theorem 7.1.4 it follows that $ans_s(q, \mathcal{I}, \mathcal{D}) = q^{chase(\langle \Psi, \Sigma_I \rangle, ret_L(\mathcal{I}, \mathcal{D}))}$ (where we exclude from the answer set the tuples that contain at least one of the newly introduced constants for the construction of $ret_L(\mathcal{I}, \mathcal{D})$).

Regarding the GAV system $\mathcal{I}'$, we have that $ans_s(q, \mathcal{I}', \mathcal{D}) = q^{chase(\langle \Psi', \Sigma'_I \rangle, ret_G(\mathcal{I}', \mathcal{D}))}$ (we exclude here the answers that contain at least one of the newly introduced constants for the construction of the chase).

[3]Since EDs remain unchanged in the transformation, we consider a schema without EDs.

Now, let Σ_{new} be the IDs introduced in $\mathcal{I}'$ during the transformation, with $\Sigma'_I = \Sigma_I \cup \Sigma_{new}$; observe that $chase(\langle \Psi', \Sigma_{new} \rangle, ret_G(\mathcal{I}', \mathcal{D}))$ is finite, since the dependencies in Σ_{new} are acyclic.

Moreover, it is straightforward to see that, modulo renaming of the fresh constants z_i introduced in the construction of the retrieved global database for the LAV system $\mathcal{I}$ and in the construction of the chase for the GAV system $\mathcal{I}'$, we have that, for each relation symbol g in $\mathcal{G}$ (and therefore in $\mathcal{G}'$), $g^{chase(\langle \Psi', \Sigma_{new} \rangle, ret_G(\mathcal{I}', \mathcal{D}))} = g^{ret_L(\mathcal{I}, \mathcal{D})}$. In fact, by applying the ID chase rule for $chase(\langle \Psi', \Sigma_{new} \rangle, ret_G(\mathcal{I}', \mathcal{D}))$, we populate the relations of $\mathcal{G}'$ that are also in $\mathcal{G}$ exactly in the same way in which they are populated in the construction of $ret_L(\mathcal{I}, \mathcal{D})$. The fresh variables are first generated by applying the chase rule for the IDs of the form $\mathsf{image\_}s[1, \ldots, n + n_c] \subseteq \mathsf{expand\_}s[1, \ldots, n + n_c]$, and then propagated to the global relations g_i in the positions where the non-distinguished variables appear. The constants appearing in the body of the LAV mapping queries are properly introduced in the head of the GAV mapping queries, and then propagated to the global relations g_i in the positions where the constants appear. Finally, constants of the source database $\mathcal{D}$, are introduced in $ret_G(\mathcal{I}', \mathcal{D})$ according to the mapping $\mathcal{M}'$, and then propagated to the global relations in the positions where the distinguished variables appear.

Now, observe that, since the dependencies in Σ_I do not involve the relations introduced in the transformation from $\mathcal{I}$ to $\mathcal{I}'$, the application of the ID chase rule to $chase(\langle \Psi', \Sigma_{new} \rangle, ret_G(\mathcal{I}', \mathcal{D}))$ is independent of the newly introduced relations. Therefore, for each relation symbol in $\mathcal{G}$ (and also in $\mathcal{G}'$), we have $g^{chase(\langle \Psi, \Sigma_I \rangle, ret_L(\mathcal{I}, \mathcal{D}))} = g^{chase(\langle \Psi', \Sigma_I \rangle, chase(\langle \Psi', \Sigma_{new} \rangle, ret_G(\mathcal{I}', \mathcal{D})))} = g^{chase(\langle \Psi', \Sigma'_I \rangle, ret_G(\mathcal{I}', \mathcal{D}))}$.

Now, let us reintroduce the EDs Σ_E both in $\mathcal{I}$ and in $\mathcal{I}'$. We recall that, given a relational schema $\mathcal{RS} = \langle \Psi, \Sigma_I, \Sigma_E \rangle$, a database instance $\mathcal{DB}$ for $\mathcal{RS}$ and a query Q over $\mathcal{RS}$, we have that $ans_s(Q, \mathcal{RS}, \mathcal{D}) = Q^{chase(\mathcal{RS}, \mathcal{D})}$ iff $\mathcal{D}$ satisfies $\Sigma_E^{*,\Sigma}$, where $\Sigma = \Sigma'_I \cup \Sigma_E$ indicates the set of constraints which the logical implication of EDs by IDs and EDs is computed for (see Section 5.2.1). As proved in Theorem 5.2.4, this means that $chase(\mathcal{RS}, \mathcal{D})$ satisfies Σ_E. Conversely, if $\mathcal{D}$ does not satisfy $\Sigma_E^{*,\Sigma}$, query answering is meaningless, indeed some ED in Σ_E should be violated by $chase(\mathcal{RS}, \mathcal{D})$. Notice now that EDs in Σ_E in $\mathcal{G}'$ involve only global relation g that belong also to $\mathcal{G}$. Hence, since $g^{chase(\langle \Psi, \Sigma_I \rangle, ret_L(\mathcal{I}, \mathcal{D}))} = g^{chase(\langle \Psi', \Sigma'_I \rangle, ret_G(\mathcal{I}', \mathcal{D}))}$ it follows that *(a)* if $ret_L(\mathcal{I}, \mathcal{D})$ does not satisfy $\Sigma_E^{*,\Sigma_I \cup \Sigma_E}$, then $ret_G(\mathcal{I}', \mathcal{D})$ does not satisfy $\Sigma_E^{*,\Sigma'_I \cup \Sigma_E}$, or *(b)* if $ret_L(\mathcal{I}, \mathcal{D})$ satisfy $\Sigma_E^{*,\Sigma_I \cup \Sigma_E}$, then $ret_G(\mathcal{I}', \mathcal{D})$ satisfy $\Sigma_E^{*,\Sigma'_I \cup \Sigma_E}$. Notice that in the case *(a)* query answering is meaningless for both $\mathcal{I}$ and $\mathcal{I}'$, i.e., every tuple is in the answer to any query, whereas in case *(b)* query answering is analogous to the case in which no ED is specified on the global schema, i.e., $ans_s(q, \mathcal{I}, \mathcal{D}) = Q^{chase(\langle \Psi, \Sigma_I \rangle, ret_L(\mathcal{I}, \mathcal{D}))}$ and $ans_s(Q, \mathcal{I}', \mathcal{D}) = Q^{chase(\langle \Psi', \Sigma'_I \rangle, ret_G(\mathcal{I}', \mathcal{D}))}$.

Since Q involves only global relations that belong both to $\mathcal{G}$ and $\mathcal{G}'$, and the fresh constants are not considered in the evaluation of Q over $chase(\langle \Psi', \Sigma'_I \rangle, ret_G(\mathcal{I}', \mathcal{D}))$ and over $chase(\langle \Psi, \Sigma_I \rangle, ret_L(\mathcal{I}, \mathcal{D}))$, the claim follows immediately. □

7.2.2 Query Rewriting in the Presence of IDs and EDs

With the results of the previous section in place, we can now address the problem of computing the answers to a query posed to a LAV data integration system, in the presence of IDs and EDs on the global schema. Since we are able to transform a LAV system into a query-preserving GAV one, we can apply the same technique we developed for GAV systems.

Theorem 7.2.3 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, where $\mathcal{G} = \langle \Psi, \Sigma_I, \Sigma_E \rangle$, Q a query over $\mathcal{G}$, and $\mathcal{D}$ a source database for $\mathcal{I}$; moreover, let $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}' \rangle$ the system obtained by compiling $\mathcal{I}$ into an equivalent GAV system, where $\mathcal{G}' = \langle \Psi', \Sigma'_I, \Sigma_E \rangle$, and let Π_{ID} be the UCQ returned by* ID-rewrite(Ψ', Σ'_I, Q) *and $\Pi_{\mathcal{M}'}$ the program $\{V_g \mid \langle g, V_g \rangle \in \mathcal{M}'\}$. Then, we have that $\Pi_{ID} \cup \Pi_{\mathcal{M}'} = ans_s(Q, \mathcal{I}, \mathcal{D})$ iff $ret_L(\mathcal{I}, \mathcal{D})$ is consistent with $\Sigma_E^{*, \Sigma_I \cup \Sigma_E}$.*

Proof. The theorem is an immediate consequence of Theorem 7.2.2 and Theorem 6.3.5. □

Example 7.2.1 (contd.) Consider now the same query of Example 6.3.3 Q_1 that contains the only CQ q_0 of the form $q(Dr) \leftarrow \mathit{department}(C, Dr), \mathit{employee}(Dr, S, D, B)$ that asks for directors of departments that are also employees. After compiling the LAV system $\mathcal{I}_L = \langle \mathcal{G}, \mathcal{S}_L, \mathcal{M}_L \rangle$ into the query-preserving GAV system $\mathcal{I}_{LG} = \langle \mathcal{G}', \mathcal{S}_L, \mathcal{M}' \rangle$, we are able to compute the answers to the query Q_1, by executing ID-rewrite(Ψ', Σ'_I, Q_1)[4]. At the beginning, $Q_{aux} = Q' = Q_1 = \{q_0\}$, and inclusion dependencies I_1, I_2, I_3 are applicable to the atoms of the queries in Q' exactly as performed by the execution of ID-rewrite(Ψ, Σ_I, Q) described in Section 6.3.1. Hence, we can describe the execution of ID-rewrite(Ψ', Σ'_I, Q_1) starting from the output of ID-rewrite(Ψ, Σ_I, Q_1), i.e., when $Q' = \{q_0, q_1, q_2, q_3\}$.

Consider for example q_3 and its unique atom $g_3 = \mathit{employee}(Dr, \xi, \xi, Dr)$. It is easy to see that I_7 is applicable to the atom in g_3 and that $gr(I_7, g_3) = \mathsf{expand\_}s_3(Dr, \xi, \xi, \xi)$. Then, by applying I_4 to such an atom we obtain $\mathsf{image\_}s_3(Dr, \xi, \xi)$ to which no ID in Σ'_I is applicable. The algorithm proceeds analogously for the other queries in Q'. The result of the computation is

$$
\begin{aligned}
q(Dr) &\leftarrow \mathsf{image\_}s_3(Dr, Db, S) \\
q(Dr) &\leftarrow \mathsf{image\_}s_3(Dr, Db, S), \mathsf{image\_}s_4(Dr, B) \\
q(Dr) &\leftarrow \mathsf{image\_}s_4(Dr, Dr) \\
q(Dr) &\leftarrow \mathsf{image\_}s_3(Dr, Db, S), \mathsf{image\_}s_4(E, Dr) \\
\mathsf{image\_}s_3(N, Db, S) &\leftarrow s_3(N, Db, S) \\
\mathsf{image\_}s_4(E, B) &\leftarrow s_4(E, B)
\end{aligned}
$$

■

7.2.3 Query Rewriting in the Presence of KDs

We conclude this section by briefly considering query rewriting for NKC LAV systems. According to Theorem 7.1.6, query answering in this setting is decidable, and it is possible to separately deal with IDs and KDs. Hence, an actual procedure for processing a query Q over a NKC LAV

[4]Since no ED is specified over $\mathcal{G}$ the the retrieved global database of $\mathcal{I}_L$ trivially satisfies the EDs.

system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G} = \langle \Psi, \Sigma_I, \Sigma_K, \Sigma_E \rangle$ consists of computing $\mathsf{ID\text{-}rewrite}(\Psi, \Sigma_I, Q)$, then computing $ret_{LK}(\mathcal{I}, \mathcal{D})$, and finally evaluating $\mathsf{ID\text{-}rewrite}(\Psi, \Sigma_I, Q)$ over $ret_{LK}(\mathcal{I}, \mathcal{D})$. If $ret_{LK}(\mathcal{I}, \mathcal{D})$ is not consistent with Σ_K, then query answering is meaningless, i.e., every tuple is in the answer of any query.

Comparing the two query processing techniques presented for GAV and LAV systems with KDs and IDs, we note that, while in the GAV case we can evaluate the query $\Pi_{ID} \cup \Pi_{\mathcal{M}}$ over the source database $\mathcal{D}$, where Π_{ID} is the output of $\mathsf{ID\text{-}rewrite}$ and $\Pi_{\mathcal{M}}$ is the program associated to the mapping, in the LAV case we necessarily have to evaluate Π_{ID} over $ret_{LK}(\mathcal{I}, \mathcal{D})$.

Now the question arises if, also in the presence of KDs, it is possible to first compile a LAV system into a query equivalent GAV one, and then proceed as GAV. However, such a possibility does not actually hold, since in LAV systems with KDs it is not possible to obtain a perfect rewriting of a UCQ expressed in terms of a UCQ.

Indeed, through the following example we prove that *recursive* queries are needed to answer a UCQ in a LAV system with key dependencies on the global schema. The example is adapted from the one used in [61] to show that a recursive query rewriting is needed in the presence of *functional dependencies*, i.e., a more general class of dependencies than KDs.[5]

Example 7.2.4 Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a system in which $\mathcal{G}$ comprises the relations $pilot(Name, Airline)$ and $aircraft(Code, Airline)$ where the key dependencies $key(pilot) = \{1\}$, and $key(aircraft) = \{1\}$ are expressed on $\mathcal{G}$. Consider the source relation $s(Pilot, Aircraft)$ and the LAV mapping $s(P, C) \leftarrow pilot(P, A), aircraft(C, A)$. Finally, let us pose the following query Q on the global schema: $q(P) \leftarrow pilot(\mathsf{Mike}, A), pilot(P, A)$, asking for pilots of the same company of Mike. Since s does not maintain information about which airlines pilots work for, it is not possible to answer the query disregarding the KDs on $\mathcal{G}$. Moreover, it can be shown that no UCQ is a perfect rewriting of the query, and that the perfect rewriting must necessarily be expressed in terms of a recursive query: for instance, the following recursive query is a perfect rewriting of Q:

$$\begin{aligned} q(P) &\leftarrow s(\mathsf{Mike}, C), s(P, C) \\ q(P) &\leftarrow q(P_1), s(P_1, C), s(P, C) \end{aligned}$$

■

7.3 Discussion

In this chapter we have addressed the problem of query answering in LAV integration systems, and have presented techniques for query answering, when key, inclusion and exclusion dependencies are expressed on the global schema. In the presence of only IDs and EDs our approach is purely intensional, since query answering is solved by first computing the perfect rewriting of a query issued on the global schema, and then evaluating the rewritten query. On the other hand, in the presence of KDs, we resort to query answering, i.e., we use constructively data at the sources in order to provide answers to the user query. As said in the above section, this is needed in order

[5] Actually, [61] refers to maximally contained rewritings: however, when queries in the mapping are conjunctive queries, the maximally-contained rewriting coincides with the perfect rewriting.

to take into account KDs in the construction of the retrieved global database, which may force fresh constants introduced during such a process to be equal. Nonetheless, the definition of a perfect rewriting also for this setting turns out to be a challenging and interesting problem that has not been addressed in this thesis, but can be seen as a future work on such matters.

We point out that, although developed for dealing with integrity constraints, our technique for query rewriting in LAV turns out to be simpler than the Duschka inverse rules algorithm [61], even in the absence of inclusion and exclusion dependencies. Indeed, the perfect rewriting produced by the algorithm in [61] is a logic program with Skolem functions representing existential variables in the head of the rules. In order to evaluate such a rewriting over a database, a non-trivial procedure to eliminate Skolem function from the program has to be performed for each query issued on the system. Conversely, by applying our technique for query processing in LAV, which is based on the transformation of the system into an equivalent GAV one, we directly obtain a UCQ that can be easily handled without resorting to other pre-processing procedures.

We also underline that the compilation of a LAV system into a GAV one can be executed off-line, only at each change of the system specification.

Chapter 8

Efficient Evaluation of Logic Programs for Consistent Query Answering

In Chapter 6 we have provided a sound and complete query rewriting technique that allows for computing the certain answers to a query posed over a GAV data integration system under the loosely-sound semantics in the presence of IDs, KDs and EDs on the global schema. More specifically, we have shown that the perfect rewriting to a user query in this setting can be given in terms of a suitable Datalog$^\neg$ program whose stable models correspond to the databases in the loosely-sound semantics of the system.

As already said, several approaches dealing with inconsistent databases have been proposed in the last years (see Section 5.5). In all such approaches, the inconsistency is eliminated by modifying the original database and reasoning on the "repaired" database. The suitability of a possible repair depends on the underlying semantic assertions which are adopted for the database. Analogously to our approach, some of these works have proposed to formalize repair semantics by using logic programs [8, 86, 13, 16]. The common basic idea is to encode the constraints specified on the global schema into a function-free logic program, Π, using unstratified negation or disjunction, such that the stable models of this program yield the repairs of the global database. Answering a user query amounts to cautious reasoning over the logic program Π augmented with the query, cast into rules, and the retrieved facts.

An attractive feature of this approach is that logic programs serve as executable logical specifications of repair, and thus allow to state repair policies in a declarative manner rather than in a procedural way. However, a drawback of this approach is that with current implementations of stable model engines, such as DLV [109] or Smodels [133], the evaluation of queries over large data sets quickly becomes infeasible because of lacking scalability. This calls for suitable optimization methods that help in speeding up the evaluation of queries expressed as logic programs [16].

In this paper, we face this problem and make the following contributions:

(1) We present a basic formal model of data integration via logic programming specification, which abstracts from several proposals in the literature [8, 86, 13, 30, 16]. Results which are obtained on this model may then be inherited by the respective approaches.

(2) We foster a *localization approach* to reduce complexity, in which irrelevant rules are discarded and the retrieved data is decomposed into two parts: facts which will possibly be touched by a repair and facts which for sure will be not. The idea which is at the heart of the approach is to reduce the usage of the nonmonotonic logic program to the essential part for conflict resolution. This requires some technical conditions to be fulfilled in order to make the part "affected" by a repair small (ideally, as much as possible).

(3) We develop *techniques for recombining* the decomposed parts for query answering, which interleave logic programming and relational database engines. This is driven by the fact that database engines are geared towards efficient processing of large data sets, and thus will help to achieve scalability. To this end, we present a marking and query rewriting technique for compiling the reasoning tasks which emerge for user query evaluation into a relational database engine.

In our overall approach, the attractive features of a nonmonotonic logic programming system, such as DLV, can be fruitfully combined with the strengths of an efficient relational database engine. The experimental results are encouraging and show that this combination has potential for building advanced data integration systems with reasonable performance.

8.1 A Logic Framework for Query Answering

In this section, we present an abstract framework for modelling query answering in data integration systems using logic programs. We first describe the integration setting addressed in this chapter, and then we introduce a comprehensive logic programming framework for computing certain (or consistent) answers from inconsistent data integration systems.

8.1.1 Data Integration Scenario

In this chapter we consider GAV integration systems $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ where queries in the mapping $\mathcal{M}$ are Datalog$^{\neg_s}$, user queries over $\mathcal{I}$ are non-recursive Datalog$^{\neg}$, and constraints on the global schema are universally quantified constraints [2], i.e., first order formulas of the form:

$$\forall(\vec{\mathbf{x}}) \bigwedge_{i=1}^{l} A_i \supset \bigvee_{j=1}^{m} B_j \vee \bigvee_{k=1}^{n} \phi_k, \tag{8.1}$$

where $l + m > 0$, $n \geq 0$, $\vec{\mathbf{x}}$ is a list of distinct variables, $A_1, \ldots, A_l$ and $B_1, \ldots, B_m$ are positive literals, and $\phi_1, \ldots, \phi_n$ are built-in literals.

We point out that such class of constraints comprises key and exclusion dependencies, which have been considered in the integration framework studied so far in this thesis, as well as functional dependencies and inclusion dependencies of the form $\forall(\vec{\mathbf{x}})p_1(\vec{\mathbf{x}}) \supset p_2(\vec{\mathbf{x}})$. On the other hand, we do not consider here existentially quantified IDs on the global schema. Actually, this

does not limit the application of the optimization technique that we present in this chapter to the approach to query answering in GAV systems described in Chapter 6. Indeed, in such an approach IDs are suitably preprocessed by means of the ID-rewrite algorithm, and then query answering is carried out as if IDs were not specified on the global schema, but the constraints taken into account are only the KDs and the EDs that are logical consequence of the IDs and the EDs originally specified over the global schema. Finally, it is also worth noticing that the user query and the mapping language considered in this chapter subsume the user query and the mapping language adopted in the GAV framework studied so far. Hence, the framework studied in Chapter 6 perfectly fits the one addressed in this chapter also for the query and mapping language.

Example 8.1.1 As a running example, we consider here the same integration system $\mathcal{I}_0 = \langle \mathcal{G}_0,\ \mathcal{S}_0,\ \mathcal{M}_0 \rangle$ of Example 6.1.1. We recall that in the global schema $\mathcal{G}_0 = \langle \Psi_0, \Sigma_0 \rangle$, Ψ_0 consists of the relations $player(Pcode, Pname, Pteam)$, $team(Tcode, Tname, Tleader)$, and $coach(Ccode, Cname, Cteam)$, whereas constraints in Σ_0 impose that the keys of *player*, *team*, and *coach* are respectively the attributes *Pcode*, *Tcode*, and *Ccode*, that a leader has to be a player (ID), and that a coach cannot be a player (ED). According to the integration setting studied in this chapter, we do not consider here the ID in Σ_0, whereas we take into account also the ED that is logically implied by the original ED and ID, i.e., we add to Σ_0 an ED stating that a coach cannot be a team leader (other than a player). Σ_0 can be formally defined as set of first order formulae as follows (quantifiers are omitted):

$$\begin{aligned} player(X,Y,Z) \wedge player(X,Y1,Z1) &\supset Y{=}Y1; & player(X,Y,Z) \wedge player(X,Y1,Z1) &\supset Z{=}Z1 \\ team(X,Y,Z) \wedge team(X,Y1,Z1) &\supset Y{=}Y1; & team(X,Y,Z) \wedge team(X,Y1,Z1) &\supset Z{=}Z1 \\ coach(X,Y,Z) \wedge coach(X,Y1,Z1) &\supset Y{=}Y1; & coach(X,Y,Z) \wedge coach(X,Y1,Z1) &\supset Z{=}Z1 \\ coach(X,Y,Z) \wedge player(X1,Y1,Z1) &\supset X{\neq}X1; & coach(X,Y,Z) \wedge team(X1,Y1,Z1) &\supset X{\neq}Z1 \end{aligned}$$

The first three rows encode the key dependencies, whereas the last row models the two exclusion dependencies.

The source schema $\mathcal{S}_0$ comprises the relations s_1, s_2, s_3 and s_4, whereas, the mapping $\mathcal{M}_0$ is defined by the Datalog program $player(X,Y,Z) \leftarrow s_1(X,Y,Z,W)$; $team(X,Y,Z) \leftarrow s_2(X,Y,Z)$; $team(X,Y,Z) \leftarrow s_3(X,Y,Z)$; $coach(X,Y,Z) \leftarrow s_4(X,Y,Z)$.

Finally, we consider the user query $q(X) \leftarrow player(X,Y,Z)$; $q(X) \leftarrow team(V,W,X)$, which asks for the codes of all players and team leaders. Note that this query is actually the rewriting of the query $q(X) \leftarrow player(X,Y,Z)$ of Example 6.1.1 that has been produced by the algorithm ID-rewrite, as described in Section 6.3.1. ∎

As for the semantics of $\mathcal{I}$, we assume in this chapter that the domain the discourse $\mathcal{U}$ is finite, and that the mapping is loosely-exact[1], i.e., given a source database $\mathcal{D}$ for $\mathcal{I}$ we have that

$$sem_{ls}(\mathcal{I},\mathcal{D}) = \{\mathcal{B} \mid \mathcal{B} \text{ is consistent with } \mathcal{G} \text{ and is minimal w.r.t. } \leq_{ret(\mathcal{I},\mathcal{D})}\}$$

[1]The loosely-exact semantics has been defined in Section 3.3. Here we recall its definition given in terms of the retrieved global database $ret(\mathcal{I},\mathcal{D})$.

$player^{ret(\mathcal{I}_0,\mathcal{D}_0)}$:

10	Totti	RM
9	Beckham	MU

$team^{ret(\mathcal{I}_0,\mathcal{D}_0)}$:

RM	Roma	10
MU	Man. Utd.	8
RM	Real Madrid	10

$coach^{ret(\mathcal{I}_0,\mathcal{D}_0)}$:

7	Ferguson	MU

Figure 8.1: Retrieved global database for the football teams scenario.

where $\mathcal{B}_1 \leq_{\mathcal{DB}} \mathcal{B}_2$ iff $\triangle(\mathcal{B}_1, \mathcal{DB}) \subseteq \triangle(\mathcal{B}_2, \mathcal{DB})$, where $\triangle(X, Y)$ denotes the symmetric difference between sets X and Y.

According to a classical terminology in the area of inconsistent databases, in the following we say that the global databases in $sem_{ls}(\mathcal{I}, \mathcal{D})$ are the *repairs* of the integration system $\mathcal{I}$ w.r.t. $\mathcal{D}$. Generally speaking, given a relational schema $\mathcal{RS} = \langle \Psi, \Sigma \rangle$ and a database instance $\mathcal{DB}$ for $\mathcal{RS}$, we indicate with $rep(\mathcal{DB})$ w.r.t. Σ the set of repairs of $\mathcal{DB}$ computed with respect to the constraints in Σ; when clear from the context, Σ is omitted. Hence, we have that $sem_{ls}(\mathcal{I}, \mathcal{D})$ is actually the set $rep(ret(\mathcal{I}, \mathcal{D}))$ w.r.t. Σ.

We point out that, in the presence of only KDs and EDs the loosely-exact semantics coincides with the loosely-sound semantics addressed in Chapter 6, i.e., the databases in the two semantics are the same. Furthermore, since in this case repairing is performed only by deleting facts, assuming that the domain $\mathcal{U}$ is finite or infinite does not make any difference. Hence, modulo the preprocessing of IDs, the framework studied in Chapter 6 is perfectly compliant with these two assumptions.

Example 8.1.1 (contd.) Assume that the information content of the sources is given by the database $\mathcal{D}_0 = \{ s_1(10, \mathsf{Totti}, \mathsf{RM}, 27),\ s_1(9, \mathsf{Beckham}, \mathsf{MU}, 28),\ s_2(\mathsf{RM},\ \mathsf{Roma}, 10),\ s_3(\mathsf{MU}, \mathsf{Man.\ Utd.}, 8),\ s_3(\mathsf{RM}, \mathsf{Real\ Madrid}, 10),\ s_4(7, \mathsf{Ferguson}, \mathsf{MU})\}$. Then, the retrieved global database $ret(\mathcal{I}_0, \mathcal{D}_0)$ shown in Fig. 8.1 violates the key constraint on *team*, witnessed by the facts $team(\mathsf{RM}, \mathsf{Roma}, 10)$ and $team(\mathsf{RM},\ \mathsf{Real\ Madrid}, 10)$, which coincide on *Tcode* but differ on *Tname*. A repair results by removing exactly one of these facts; hence, $sem_{ls}(\mathcal{I}_0, \mathcal{D}_0) = \{\mathcal{B}_1, \mathcal{B}_2\}$, where $\mathcal{B}_1$ and $\mathcal{B}_2$ are as shown in Figure 8.2.

$player^{\mathcal{B}_1}$:

10	Totti	RM
9	Beckham	MU

$team^{\mathcal{B}_1}$:

RM	Roma	10
MU	Man. Utd.	8

$coach^{\mathcal{B}_1}$:

7	Ferguson	MU

$player^{\mathcal{B}_2}$:

10	Totti	RM
9	Beckham	MU

$team^{\mathcal{B}_2}$:

MU	Man. Utd.	8
RM	Real Madrid	10

$coach^{\mathcal{B}_2}$:

7	Ferguson	MU

Figure 8.2: Repairs of $\mathcal{I}_0$ w.r.t. $\mathcal{D}_0$.

For the query $q(X) \leftarrow player(X, Y, Z)$; $q(X) \leftarrow team(V, W, X)$, we thus obtain that $ans_{ls}(q, \mathcal{I}_0, \mathcal{D}_0) = \{8, 9, 10\}$. If we consider the query $q'(Y) \leftarrow team(X, Y, Z)$, we have that $ans_{ls}(q', \mathcal{I}_0, \mathcal{D}_0) = \{\mathsf{Man.\ Utd.}\}$, while considering $q''(X, Z) \leftarrow team(X, Y, Z)$, we have that $ans_{ls}(q'', \mathcal{I}_0, \mathcal{D}_0) = \{\langle \mathsf{RM}, 10\rangle, \langle \mathsf{MU}, 8\rangle\}$. ∎

8.1.2 Logic Programming for Consistent Query Answering

We now describe a generic logic programming framework for computing certain answers to queries posed to a data integration system in which inconsistency possibly raises.

According to several proposals in the literature, we provide answers to user queries by encoding the mapping assertions in $\mathcal{M}$ and the constraints in Σ in a Datalog program enriched with unstratified negation or disjunction, in such a way that the stable models of this program map to the repairs of the retrieved global database.

Definition 8.1.2 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system where $\mathcal{G} = \langle \Psi, \Sigma \rangle$, $\mathcal{D}$ is a source database for $\mathcal{I}$, and q is a non-recursive Datalog$^\neg$ query over $\mathcal{G}$. Then, a* logic specification for querying *$\mathcal{I}$ is a Datalog$^{\vee,\neg}$ program $\Pi_\mathcal{I}(q) = \Pi_\mathcal{M} \cup \Pi_\Sigma \cup \Pi_q$ such that*

1. *$ret(\mathcal{I}, \mathcal{D}) \rightleftharpoons \mathrm{SM}(\Pi^\mathcal{D}_\mathcal{M})$, where $\Pi_\mathcal{M}$ is a Datalog$^{\neg_s}$ program,*
2. *$sem_{ls}(\mathcal{I}, \mathcal{D}) \rightleftharpoons \mathrm{SM}(\Pi_\Sigma^{ret(\mathcal{I},\mathcal{D})})$, and*
3. *$ans_{ls}(q, \mathcal{I}, \mathcal{D}) \rightleftharpoons \{t \mid q(t) \in M \text{ for each } M \in \mathrm{SM}((\Pi_q \cup \Pi_\Sigma)^{ret(\mathcal{I},\mathcal{D})})\}$,*
   where Π_q is a non-recursive Datalog$^\neg$ program,

where $\rightleftharpoons$ denotes a polynomial-time computable correspondence between two sets.

This definition establishes a connection between the semantics of $\Pi_\mathcal{I}(q)$ and the certain answers to a query posed to $\mathcal{I}$ (Item 3) provided some syntactic transformations, which typically are simple encodings such that $\rightleftharpoons$ is a linear-time computable bijection. In particular, $\Pi_\mathcal{I}(q)$ is composed by three modules that can be hierarchically evaluated, i.e., $\Pi_\mathcal{M} \triangleright \Pi_\Sigma \triangleright \Pi_q$ [66], using Splitting Sets [122]:

- $\Pi_\mathcal{M}$ is used for retrieving data from the sources: the retrieved global database can be derived from its unique stable model (Item 1);
- Π_Σ is used for enforcing the constraints on the retrieved global database, whose repairs can be derived from the stable models of $\Pi_\Sigma[ret(\mathcal{I}, \mathcal{D})]$ (Item 2);
- finally, Π_q is used for encoding the user query q.

The above framework generalizes logic programming formalizations proposed in different integration settings, such as the one described in Chapter 6 of this thesis and the ones recently proposed in [13, 16]. In this respect, the precise structure of the program $\Pi_\mathcal{I}(q)$ depends on the form of the mapping, the language adopted for specifying mapping views and user queries, and the nature of constraints expressed on the global schema. In particular, compared to the perfect rewriting $\Pi_{KD} \cup \Pi_{ED} \cup \Pi_{ID} \cup \Pi_{\mathcal{MD}}$ of Section 6.4, we have that $\Pi_\mathcal{M} = \Pi_{\mathcal{MD}}$, $\Pi_\Sigma = \Pi_{KD} \cup \Pi_{ED}$, and $\Pi_q = \Pi_{ID}$.

We point out that, logic programming specifications proposed in the setting of a single inconsistent database [86, 8] are also captured by the above framework. Indeed, a single inconsistent database can be conceived as the retrieved global database of a GAV data integration system in which views of the mapping are assumed exact. The logic programs for consistently querying a single database are of the form $\Pi_\mathcal{I}(q) = \Pi_\Sigma \cup \Pi_q$.

8.2 Optimization of Query Answering

The source of complexity in evaluating the program $\Pi_{\mathcal{I}}(q)$ defined in the above section, actually lies in the conflict resolution module Π_{Σ}, and in the evaluation of Π_q. Indeed, $\Pi_{\mathcal{M}}$ is a Datalog$^{\neg_s}$ program that can be evaluated in polynomial time over the source database $\mathcal{D}$ for constructing $ret(\mathcal{I}, \mathcal{D})$, whereas Π_q is a non-recursive Datalog$^{\neg}$ program that has to be evaluated over each repair of the retrieved global database, and Π_{Σ} is in general a Datalog$^{\vee,\neg}$ program whose evaluation complexity is at the second level of the polynomial hierarchy in data complexity [86]. Furthermore, also evaluating programs with lower complexity over large data sets by means of stable models solvers, such as DLV [109] or Smodels [133], quickly becomes infeasible. This calls for suitable optimization methods speeding up the evaluation (as recently stated in [16]).

Concentrating on the most relevant and computational expensive aspects of the optimization, we focus here on Π_{Σ}, assuming that $ret(\mathcal{I}, \mathcal{D})$ is already computed, and devise intelligent techniques for the evaluation of Π_q.

Roughly speaking, in our approach we first localize in the retrieved global database $ret(\mathcal{I}, \mathcal{D})$ the facts that are not "affected" (formally specified below) by any violation. Then, we compute the repairs by taking into account only the affected facts, and finally we recombine the repairs to provide answers to the user query. Since in practice, the size of the set of the affected facts is much smaller than the size of the retrieved global database, the computation of the stable models of Π_{Σ}, i.e., repairs of $ret(\mathcal{I}, \mathcal{D})$ (Item 2 in Def. 8.1.2), over the affected facts is significantly faster than the naive evaluation of $\Pi_{\mathcal{I}}(q)$ on the whole retrieved global database.

In a nutshell, our overall optimization approach comprises the following steps:

Pruning: We first eliminate from $\Pi_{\mathcal{I}}(q)$ the rules that are not relevant for computing answers to a user query q. This can be done by means of a static syntactic analysis of the program $\Pi_{\mathcal{I}}(q)$. In this step we also localize the portion of $ret(\mathcal{I}, \mathcal{D})$ that is needed to answer the query q.

Decomposition: We localize inconsistency in the retrieved global database, and compute the set of facts that are affected by repair. Finally, we compute repairs, $\mathcal{R}_1, \ldots, \mathcal{R}_n$, of this set.

Recombination: We suitably recombine the repairs $\mathcal{R}_1, \ldots, \mathcal{R}_n$ for computing the answers to the query q.

In the rest of this section, we describe in detail the three steps.

8.2.1 Pruning

In this step, we localize the portion of $ret(\mathcal{I}, \mathcal{D})$ that is needed to answer the query q, thus avoiding to retrieve from the sources tuples that do not contribute to answering the query; moreover, we also select the rules of $\Pi_{\mathcal{I}}(q)$ that are necessary for handling constraint violations in such a portion of the retrieved global database, thus disregarding non relevant rules. Indeed, $ret(\mathcal{I}, \mathcal{D})$ is in general a superset of the data that are needed to answer the query q. For instance, when no integrity constraint is specified on the global schema it is not necessary to consider the

extension of global relations that do not appear in the body of the rules in q, i.e., that are not EDB predicates of q. Of course, when integrity constraints are specified on the global schema the situation is not so easy.

We next provide some definitions that allow us to suitably prune the retrieved global database and the logic program $\Pi_{\mathcal{I}}(q)$. We point out that our studies on pruning are in a preliminary stage, and that the solution provided can be optimized in several directions. Indeed, the pruning step strongly depends on the user query and the form of constraints on the global schema, and proper solutions for particular and significant classes of queries and constraints should be adopted. For example, solutions that exploit information provided by selections in the query, or take advantage from the form of the dependencies if they are particularly simple. For instance, the case in which only key dependencies are specified on the relations involved in the query can be basically faced by computing only the extension of the EDB predicates of the query q. Nonetheless, our proposal, even if preliminary, should shed light on this point of the overall approach to the efficient evaluation of the logic programs $\Pi_{\mathcal{I}}(q)$.

Definition 8.2.1 Given a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G} = \langle \Psi, \Sigma \rangle$, and a query q over $\mathcal{I}$, we say that a relation r in $\mathcal{G}$ is *relevant w.r.t. q* if

(a) r is a predicate in the body of the rules of q, or

(b) r is involved in a dependency in Σ^* with a relation s relevant w.r.t. q, where Σ^* denotes the set of dependencies that are logically implied by Σ.

We denote by $ret_r(\mathcal{I}, \mathcal{D}, q)$ the subset of $ret(\mathcal{I}, \mathcal{D})$ that contains only the extension of relations that are relevant w.r.t. q, i.e., $ret_r(\mathcal{I}, \mathcal{D}, q) = \{r(t) | r(t) \in ret(\mathcal{I}, \mathcal{D})$ and r is relevant w.r.t. $q\}$.

For instance, in our ongoing example, the relations *player* and *team* are relevant due to condition (a), whereas relation *coach* is relevant due to condition (b), witnessed by the EDs $coach(X, Y, Z) \wedge player(X1, Y1, Z1) \supset X \neq X1$ and $coach(X, Y, Z) \wedge team(X1, Y1, Z1) \supset X \neq Z1$. Hence, in this case $ret_r(\mathcal{I}_0, \mathcal{D}_0, q) = ret(\mathcal{I}_0, \mathcal{D}_0)$.

Let us now turn our attention to the program $\Pi_{\mathcal{I}}(q)$, and extend the notion of relevance to its rules.

Definition 8.2.2 *Given a data integration system $\mathcal{I}$, and a logic specification $\Pi_{\mathcal{I}}(q)$ for $\mathcal{I}$, we say that a rule ρ in $\Pi_{\mathcal{I}}(q)$ is* relevant w.r.t. *q if ρ contains a predicate expressed in terms of a relation r that is relevant w.r.t. q. We denote by $\Pi_{\mathcal{I}r}(q)$ the set of rules in $\Pi_{\mathcal{I}}(q)$ that are relevant w.r.t. q.*

The following proposition, whose proof is straightforward, allows us to focus on $\Pi_{\mathcal{I}r}(q)$, and $ret_r(\mathcal{I}, \mathcal{D}, q)$ in order to compute the certain answers to the query q.

Proposition 8.2.3 *Given a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, a source database $\mathcal{D}$ for $\mathcal{I}$, and a query q over $\mathcal{I}$, we have that*

$$ans_{ls}(q, \mathcal{I}, \mathcal{D}) \rightleftharpoons \{t \mid q(t) \in M \text{ for each } M \in \mathrm{SM}((\Pi_q \cup \Pi_\Sigma)_r^{ret_r(\mathcal{I}, \mathcal{D}, q)})\}$$

where $(\Pi_q \cup \Pi_\Sigma)_r$ indicates the relevant portion of $\Pi_q \cup \Pi_\Sigma$.

In the following we implicitly consider $ret_r(\mathcal{I}, \mathcal{D}, q)$ and $\Pi_{\mathcal{I}r}(q)$. Nonetheless, to keep things simple, we do not use the subscript $_r$, and refer to $ret(\mathcal{I}, \mathcal{D})$ and $\Pi_{\mathcal{I}}(q)$

8.2.2 Decomposition

We start with some concepts for a single relational schema. Let $\mathcal{RS} = \langle \Psi, \Sigma \rangle$ be a relational schema. We call two facts $p_1(a_1, \ldots, a_n)$ and $p_2(b_1, \ldots, b_m)$, where $p_1, p_2 \in \Psi$ and each a_i, b_j is in the domain $\mathcal{U}$, *constraint-bounded in* $\mathcal{RS}$, if they occur in the same ground constraint $\sigma^g \in ground(\Sigma)$. Furthermore, for any $\sigma^g \in ground(\Sigma)$, we use $facts(\sigma^g)$ to denote the set of all facts $p(t)$, $p \in \Psi$, which occur in σ^g.

Let $\mathcal{DB}$ be a database for $\mathcal{RS}$. Then, the *conflict set* for $\mathcal{DB}$ w.r.t. $\mathcal{RS}$ is the set of facts $C^{\mathcal{RS}}_{\mathcal{DB}} = \{p(t) \mid \exists \sigma^g \in ground(\Sigma) \wedge p(t) \in facts(\sigma^g) \wedge \sigma^g \text{ is violated in } \mathcal{DB}\}$, i.e., the set of facts occurring in the constraints of $ground(\Sigma)$ that are violated in $\mathcal{DB}$.

For instance, in our ongoing example, the conflict set consists of the facts that violate the key constraints on *team*, i.e., $A_{\mathcal{DB}}(\mathcal{I}_0, \mathcal{D}_0) = \{team(\mathsf{RM}, \mathsf{Roma}, 10), team(\mathsf{RM}, \mathsf{Real\ Madrid}, 10)\}$.

Definition 8.2.4 *Let $\mathcal{RS}$=$\langle \Psi, \Sigma \rangle$ be a relational schema and $\mathcal{DB}$ a database for $\mathcal{RS}$. Then, the* conflict closure *for $\mathcal{DB}$, denoted by $C^{\mathcal{RS}^*}_{\mathcal{DB}}$, is the least set such that $t \in C^{\mathcal{RS}^*}_{\mathcal{DB}}$ if either* (i) *$t \in C^{\mathcal{RS}}_{\mathcal{DB}}$, or* (ii) *$t$ is constraint-bounded in $\mathcal{RS}$ with a fact $t' \in C^{\mathcal{RS}^*}_{\mathcal{DB}}$. Moreover, we call $S^{\mathcal{RS}}_{\mathcal{DB}} = \mathcal{DB} - C^{\mathcal{RS}^*}_{\mathcal{DB}}$ and $A^{\mathcal{RS}}_{\mathcal{DB}} = \mathcal{DB} \cap C^{\mathcal{RS}^*}_{\mathcal{DB}}$ the* safe database *and the* affected database *for $\mathcal{DB}$, respectively.*

We drop the superscript $\mathcal{RS}$ if it is clear from the context. Intuitively, $C^*_{\mathcal{DB}}$ contains all facts involved in constraint violations, i.e., facts belonging to $C_{\mathcal{DB}}$, and facts which possibly must be changed in turn to avoid new inconsistency with Σ by repairing.

We now consider the following two subsets of $ground(\Sigma)$:

(i) $\Sigma^A_{\mathcal{DB}}$ consists of all the ground constraints in which at least one fact from $C^*_{\mathcal{DB}}$ occurs, i.e., $\Sigma^A_{\mathcal{DB}} = \{\sigma^g \in ground(\Sigma) \mid facts(\sigma^g) \cap C^*_{\mathcal{DB}} \neq \emptyset\}$, and

(ii) $\Sigma^S_{\mathcal{DB}}$ consists of all the ground constraints in which at least one fact occurs which is not in $C^*_{\mathcal{DB}}$, i.e., $\Sigma^S_{\mathcal{DB}} = \{\sigma^g \in ground(\Sigma) \mid facts(\sigma^g) \not\subseteq C^*_{\mathcal{DB}}\}$.

We first show that $\Sigma^A_{\mathcal{DB}}$ and $\Sigma^S_{\mathcal{DB}}$ form a partitioning of $ground(\Sigma)$.

Proposition 8.2.5 *Let $\mathcal{RS} = \langle \Psi, \Sigma \rangle$ be a relational schema, and let $\mathcal{DB}$ a database for $\mathcal{RS}$. Then,*

1. $\bigcup_{\sigma^g \in \Sigma^A_{\mathcal{DB}}} facts(\sigma^g) \subseteq C^*_{\mathcal{DB}}$;
2. $\bigcup_{\sigma^g \in \Sigma^S_{\mathcal{DB}}} facts(\sigma^g) \cap C^*_{\mathcal{DB}} = \emptyset$;
3. $\Sigma^A_{\mathcal{DB}} \cap \Sigma^S_{\mathcal{DB}} = \emptyset$ *and* $\Sigma^A_{\mathcal{DB}} \cup \Sigma^S_{\mathcal{DB}} = ground(\Sigma)$.

Proof. 1. All facts occurring in a ground constraint $\sigma^g \in \Sigma^A_{\mathcal{DB}}$ must belong to $C^*_{\mathcal{DB}}$. Indeed, by definition of $\Sigma^A_{\mathcal{DB}}$, σ^g contains at least one fact t in $C^*_{\mathcal{DB}}$; each other fact in σ^g is constraint-bounded in $\mathcal{RS}$ with t, and hence it also is in $C^*_{\mathcal{DB}}$.

2. Assume by contradiction that some $\sigma^g \in \Sigma^S_{\mathcal{DB}}$ with $facts(\sigma^g) \cap C^*_{\mathcal{DB}} \neq \emptyset$ exists. Then, Definition 8.2.4 implies $facts(\sigma^g) \subseteq C^*_{\mathcal{DB}}$, which contradicts $\sigma^g \in \Sigma^S_{\mathcal{DB}}$. Part 3 is straightforward from Part 1 and Part 2. □

The separation property allows us to shed light on the structure of repairs:

Proposition 8.2.6 (Safe Database) *Let $\mathcal{RS} = \langle \Psi, \Sigma \rangle$ be a relational schema, and let $\mathcal{DB}$ be a database for $\mathcal{RS}$. Then, for each repair $\mathcal{R} \in rep(\mathcal{DB})$ w.r.t. Σ, $S_{\mathcal{DB}} = \mathcal{R} - C^*_{\mathcal{DB}}$.*

Proof. Towards a contradiction, suppose there exists a repair $\mathcal{R}$ for $\mathcal{DB}$ w.r.t. Σ such that $\mathcal{R} - C^*_{\mathcal{DB}} \neq S_{\mathcal{DB}}$. Let us construct a new database $\mathcal{R}'$ in the following way: $\mathcal{R}'$ contains all the facts of $\mathcal{R}$ that are also in $C^*_{\mathcal{DB}}$, and all the facts of $\mathcal{DB}$ that are not in $C^*_{\mathcal{DB}}$. Consider any $\sigma^g \in ground(\Sigma)$. By Proposition 8.2.5, either (i) $\sigma^g \in \Sigma^A_{\mathcal{DB}}$ or (ii) $\sigma^g \in \Sigma^S_{\mathcal{DB}}$. Moreover, since $\mathcal{R}$ satisfies σ^g, it follows that in Case (i) $\mathcal{R}'$ also satisfies σ^g. Similarly, since $S_{\mathcal{DB}}$ satisfies σ^g in Case (ii) $\mathcal{R}'$ also satisfies σ^g.

Thus, $\mathcal{R}'$ is consistent with Σ. However, it differs from $\mathcal{R}$ only in the tuples that are not in $C^*_{\mathcal{DB}}$, and all these tuples do not need to be repaired; it follows that $\Delta(\mathcal{R}', \mathcal{DB}) \subseteq \Delta(\mathcal{R}, \mathcal{DB})$. This contradicts that $\mathcal{R}$ is a repair of $\mathcal{DB}$. □

Prior to the main result of this section, we give the following lemma:

Lemma 8.2.7 *Let $\mathcal{RS} = \langle \Psi, \Sigma \rangle$ be a relational schema, and let $\mathcal{DB}$ be a database for $\mathcal{RS}$. Then, for each $S \subseteq S_{\mathcal{DB}}$, we have that*

1. *for each $\mathcal{R} \in rep(S \cup A_{\mathcal{DB}})$ w.r.t. Σ, $(\mathcal{R} \cap C^*_{\mathcal{DB}}) \in rep(A_{\mathcal{DB}})$ w.r.t. $\Sigma^A_{\mathcal{DB}}$;*
2. *for each $\mathcal{R}_a \in rep(A_{\mathcal{DB}})$ w.r.t. $\Sigma^A_{\mathcal{DB}}$, there exists some set of facts S', $S' \cap C^*_{\mathcal{DB}} = \emptyset$, such that $(\mathcal{R}_a \cup S') \in rep(S \cup A_{\mathcal{DB}})$ w.r.t. Σ.*

Proof. 1. Let $\mathcal{R}$ be a repair of $S \cup A_{\mathcal{DB}}$ w.r.t. Σ, and let $\mathcal{R}_a = \mathcal{R} \cap C^*_{\mathcal{DB}}$. Since $\mathcal{R}$ is consistent with Σ, Proposition 8.2.5 implies that $\mathcal{R}_a$ satisfies all constraints in $\Sigma^A_{\mathcal{DB}}$, while $\mathcal{R} - \mathcal{R}_a$ satisfies all constraints in $\Sigma^S_{\mathcal{DB}}$. Assume $\mathcal{R}_a$ is not a repair of $A_{\mathcal{DB}}$ w.r.t. $\Sigma^A_{\mathcal{DB}}$. Then, there exists some $\mathcal{R}'_a \in rep(A_{\mathcal{DB}})$ w.r.t. $\Sigma^A_{\mathcal{DB}}$, such that $\mathcal{R}'_a <_{A_{\mathcal{DB}}} \mathcal{R}_a$. Since $\mathcal{R}'_a$ satisfies $\Sigma^A_{\mathcal{DB}}$, Proposition 8.2.5 implies that $\mathcal{R}'_a \cup (\mathcal{R} - \mathcal{R}_a)$ satisfies Σ. But $\mathcal{R}'_a \cup (\mathcal{R} - \mathcal{R}_a) <_{S \cup A_{\mathcal{DB}}} \mathcal{R}_a$, which contradicts that $\mathcal{R}$ is a repair of $S \cup A_{\mathcal{DB}}$.
2. Let $\mathcal{R}_a$ be a repair of $A_{\mathcal{DB}}$ w.r.t. $\Sigma^A_{\mathcal{DB}}$. We show that there exists a repair $\mathcal{R}$ of $S \cup A_{\mathcal{DB}}$ w.r.t. Σ of form $\mathcal{R} = \mathcal{R}_a \cup S'$, where S' is a set that satisfies $\Sigma^S_{\mathcal{DB}}$, such that $\mathcal{R} \cap A_{\mathcal{DB}} = \mathcal{R}_a$. In particular, let $S' \in rep(S)$ w.r.t. $\Sigma^S_{\mathcal{DB}}$ be arbitrary. Using Prop. 8.2.5, we can see that $\mathcal{R}_a \cup S'$ satisfies Σ; note that $S' \cap C^*_{\mathcal{DB}} = \emptyset$. Thus, if $\mathcal{R}_a \cup S'$ is not a repair of $S \cup A_{\mathcal{DB}}$, there must exist some $\mathcal{R}'$ consistent with Σ such that $\mathcal{R}' <_{S \cup A_{\mathcal{DB}}} \mathcal{R}_a \cup S'$. Since we always can write $\mathcal{R}' = \mathcal{R}'_a \cup \mathcal{R}'_s$, where $\mathcal{R}'_a \subseteq C^*_{\mathcal{DB}}$ satisfies $\Sigma^A_{\mathcal{DB}}$ and $\mathcal{R}'_s = \mathcal{R} - \mathcal{R}'_a$ satisfies $\Sigma^S_{\mathcal{DB}}$, this implies either $\mathcal{R}'_a <_{A_{\mathcal{DB}}} \mathcal{R}_a$ or $\mathcal{R}'_s <_S S'$. This contradicts $\mathcal{R}_a \in rep(A_{\mathcal{DB}})$ w.r.t. $\Sigma^A_{\mathcal{DB}}$ and $S' \in rep(S)$ w.r.t. $\Sigma^S_{\mathcal{DB}}$. □

Armed with the above concepts and results, we now turn to the data integration setting in which we have to repair the retrieved global database $ret(\mathcal{I}, \mathcal{D})$. The following theorem shows that its repairs can be computed by looking only at $A_{ret(\mathcal{I},\mathcal{D})}$.

Theorem 8.2.8 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, where $\mathcal{G} = \langle \Psi, \Sigma \rangle$, and let $\mathcal{D}$ be a source database for $\mathcal{I}$. Then,*

1. *$\forall \mathcal{B} \in sem_{ls}(\mathcal{I}, \mathcal{D})$, $\exists \mathcal{R} \in rep(A_{ret(\mathcal{I},\mathcal{D})})$ w.r.t. Σ such that $\mathcal{B} = \mathcal{R} \cap C^*_{ret(\mathcal{I},\mathcal{D})} \cup S_{ret(\mathcal{I},\mathcal{D})}$;*
2. *$\forall \mathcal{R} \in rep(A_{ret(\mathcal{I},\mathcal{D})})$, $\exists \mathcal{B} \in sem_{ls}(\mathcal{I}, \mathcal{D})$ w.r.t. Σ such that $\mathcal{B} = \mathcal{R} \cap C^*_{ret(\mathcal{I},\mathcal{D})} \cup S_{ret(\mathcal{I},\mathcal{D})}$.*

Proof. Recall that $ret(\mathcal{I}, \mathcal{D}) = S_{ret(\mathcal{I},\mathcal{D})} \cup A_{ret(\mathcal{I},\mathcal{D})}$ and that $sem_{ls}(\mathcal{I}, \mathcal{D})$ coincides with $rep(ret(\mathcal{I}, \mathcal{D}))$ w.r.t. Σ. Thus, applying Lemma 8.2.7, first Part 1 for $S = S_{ret(\mathcal{I},\mathcal{D})}$ and then Part 2 for $S = \emptyset$, we obtain that for every $\mathcal{B} \in sem_{ls}(\mathcal{I}, \mathcal{D})$, there exists some $\mathcal{R} \in rep(A_{ret(\mathcal{I},\mathcal{D})})$ w.r.t. Σ of form $\mathcal{R} = (\mathcal{B} \cap C^*_{ret(\mathcal{I},\mathcal{D})}) \cup S'$, where $S' \cap C^*_{ret(\mathcal{I},\mathcal{D})} = \emptyset$. Hence, $\mathcal{R} \cap C^*_{ret(\mathcal{I},\mathcal{D})} = \mathcal{B} \cap C^*_{ret(\mathcal{I},\mathcal{D})}$. By Prop. 8.2.6, every $\mathcal{B} \in sem_{ls}(\mathcal{I}, \mathcal{D})$, i.e., $\mathcal{B} \in rep(ret(\mathcal{I}, \mathcal{D}))$ is of form $\mathcal{B} = (\mathcal{B} \cap C^*_{ret(\mathcal{I},\mathcal{D})}) \cup S_{ret(\mathcal{I},\mathcal{D})}$. Therefore, $\mathcal{B} = (\mathcal{R} \cap C^*_{ret(\mathcal{I},\mathcal{D})}) \cup S_{ret(\mathcal{I},\mathcal{D})}$.

Similarly, applying Lemma 8.2.7, first Part 1 for $S = \emptyset$ and then Part 2 for $S = S_{ret(\mathcal{I},\mathcal{D})}$, we obtain that for every $\mathcal{R} \in rep(A_{ret(\mathcal{I},\mathcal{D})})$ w.r.t. Σ, there exists some $\mathcal{B} \in sem_{ls}(\mathcal{I}, \mathcal{D})$ w.r.t. Σ such that $\mathcal{B} = (\mathcal{R} \cap C^*_{ret(\mathcal{I},\mathcal{D})}) \cup S'$, where $S' \cap C^*_{ret(\mathcal{I},\mathcal{D})} = \emptyset$. Moreover, Prop. 8.2.6 implies $S' = S_{ret(\mathcal{I},\mathcal{D})}$, which proves 2. □

As a consequence, for computing the repairs of the retrieved global database $ret(\mathcal{I}, \mathcal{D})$, it is sufficient to evaluate the program Π_Σ on $A_{ret(\mathcal{I},\mathcal{D})}$, i.e., to exploit the correspondence $sem_{ls}(\mathcal{I}, \mathcal{D}) \rightleftharpoons \mathrm{SM}(\Pi_\Sigma[A_{ret(\mathcal{I},\mathcal{D})}])$, intersect with $C^*_{ret(\mathcal{I},\mathcal{D})}$, and unite with $S_{ret(\mathcal{I},\mathcal{D})}$. Nonetheless, computing $A_{ret(\mathcal{I},\mathcal{D})}$ is expensive in general since it requires computing the closure of $C_{ret(\mathcal{I},\mathcal{D})}$. Furthermore, in repairs of $A_{ret(\mathcal{I},\mathcal{D})}$ many facts not in $C^*_{ret(\mathcal{I},\mathcal{D})}$ might be computed which are stripped off subsequently. Fortunately, in practice, for many important cases this can be avoided: repairs can be made fully local and even focused just on the immediate conflicts in the database.

Proposition 8.2.9 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, where $\mathcal{G} = \langle \Psi, \Sigma \rangle$, and let $\mathcal{D}$ be a source database for $\mathcal{I}$. Then,*

1. *if each $\sigma \in \Sigma$ is of the form 8.1 with $l > 0$, then each $\mathcal{R}$ repair of $A_{ret(\mathcal{I},\mathcal{D})}$ w.r.t. Σ satisfies $\mathcal{R} \subseteq C^*_{ret(\mathcal{I},\mathcal{D})}$;*
2. *if each $\sigma \in \Sigma$ is of the form 8.1 with $m = 0$, then*
   - *i) each $\mathcal{R}$ repair of $A_{ret(\mathcal{I},\mathcal{D})}$ w.r.t. Σ satisfies $\mathcal{R} \subseteq C_{ret(\mathcal{I},\mathcal{D})}$,*
   - *ii) $C_{ret(\mathcal{I},\mathcal{D})} \subseteq ret(\mathcal{I}, \mathcal{D})$, and*
   - *iii) $rep(A_{ret(\mathcal{I},\mathcal{D})})$ w.r.t. Σ coincides with $rep(C_{ret(\mathcal{I},\mathcal{D})})$ w.r.t. Σ.*

This proposition allows us to exploit Theorem 8.2.8 in a constructive way for many significant classes of constraints, for which it implies a bijection between the repairs of the retrieved global database, $ret(\mathcal{I}, \mathcal{D})$, and the repairs of its affected part $A_{ret(\mathcal{I},\mathcal{D})}$ w.r.t. the constraints Σ. In particular, Condition 1 is satisfied by all constraints that do not unconditionally enforce inclusion of some fact in every repair, while Condition 2 is satisfied by constraints that can be repaired by just deleting facts from the database, such as key constraints, functional dependencies, and exclusion dependencies.

According to Theorem 8.2.8, in case of Condition 2 the set $sem_{ls}(\mathcal{I},\mathcal{D})$ can be obtained by simply computing the repairs of the conflicting facts, $C_{ret(\mathcal{I},\mathcal{D})}$, in place of $A_{ret(\mathcal{I},\mathcal{D})}$ and by adding $S_{ret(\mathcal{I},\mathcal{D})}$ to each repair.

The following corollary formalizes the above discussion for Condition 2.

Corollary 8.2.10 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, where $\mathcal{G} = \langle \Psi, \Sigma \rangle$, and let $\mathcal{D}$ be a source database for $\mathcal{I}$. Assume each constraint in Σ has form 8.1 with $m = 0$. Then, there exists a bijection $\mu : sem_{ls}(\mathcal{I},\mathcal{D}) \rightarrow rep(C_{ret(\mathcal{I},\mathcal{D})})$, such that for each $\mathcal{R} \in sem_{ls}(\mathcal{I},\mathcal{D})$, $\mathcal{B} = \mu(\mathcal{B}) \cup S_{ret(\mathcal{I},\mathcal{D})}$ (where $C_{ret(\mathcal{I},\mathcal{D})} \subseteq ret(\mathcal{I},\mathcal{D})$).*

We point out that constraints specified in the GAV framework of Chapter 6 verify condition 2. It is also worth noticing that the computation of the set $C_{ret(\mathcal{I},\mathcal{D})}$ can be carried out very efficiently, by means of suitable SQL statements as described below.

8.2.3 Conflicting set for $ret(\mathcal{I},\mathcal{D})$ in the presence of KDs and EDs

Let us first consider the key dependencies specified over $\mathcal{G}$.

Let $r(X_1, ..., X_n, Y_1, ..., Y_m)$ be a relation in $\mathcal{G}$, such that the variables in $X_1, ..., X_n$ correspond to the attributes constituting the key. Then, the following SQL statement:

```
INSERT INTO r_i
SELECT X_1, ..., X_n, COUNT(*)
FROM r
GROUP BY X_1, ..., X_n
HAVING COUNT(*) > 0;
```

collects into $\mathbf{r_i}$ all the tuples that violate the key dependency.

Now turn the attention to the exclusion dependencies. Let $r(X_1, ..., X_n, Y_1, ..., Y_m)$, and $s(X_1, ..., X_n, Z_1, ..., Z_l)$ be relations in $\mathcal{G}$, such that $r[X_1, ..., X_n] \cap s[X_1, ..., X_n] \neq \emptyset$ is an exclusion dependency over $\mathcal{G}$. Then, the following SQL statement:

```
INSERT INTO rs_e
SELECT r.*
FROM r, s
WHERE r.X_1 = s.X_1 AND...
AND r.X_n = s.X_n;
```

collects into $\mathbf{rs_e}$ all the tuples that violate the exclusion dependency.

8.2.4 Recombination

Let us turn our attention to the evaluation of a user query q. According to the definition of certain answers (Section 3.2), we need to evaluate q over each database $\mathcal{B} \in sem_{ls}(\mathcal{I},\mathcal{D})$, and by Theorem 8.2.8 we can exploit the correspondence $sem_{ls}(\mathcal{I},\mathcal{D}) \rightleftharpoons \mathrm{SM}(\Pi_{\Sigma}[A_{ret(\mathcal{I},\mathcal{D})}])$ for computing each such $\mathcal{B}$. More precisely, we need to recombine the repairs of $A_{ret(\mathcal{I},\mathcal{D})}$ computed in the decomposition step with $S_{ret(\mathcal{I},\mathcal{D})}$ as stated by the following theorem.

$player_m{}^{ret(\mathcal{I}_0,\mathcal{D}_0)}$:

10	Totti	RM	'11'
9	Beckham	MU	'11'

$team_m{}^{ret(\mathcal{I}_0,\mathcal{D}_0)}$:

RM	Roma	10	'10'
MU	Man. Utd.	8	'11'
RM	Real Madrid	10	'01'

$coach_m{}^{ret(\mathcal{I}_0,\mathcal{D}_0)}$:

7	Ferguson	MU	'11'

Figure 8.3: The retrieved global database of our running example after marking.

Theorem 8.2.11 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, let $\mathcal{D}$ be a source database for $\mathcal{I}$, and let q be a query over $\mathcal{I}$. Then,*

$$ans_{ls}(q, \mathcal{I}, \mathcal{D}) = \bigcap_{\mathcal{R} \in rep(A_{ret(\mathcal{I},\mathcal{D})})} \Pi_q^{\mathcal{R} \cap C^*_{ret(\mathcal{I},\mathcal{D})} \cup S_{ret(\mathcal{I},\mathcal{D})}} \tag{8.2}$$

Note that the number of repairs of $A_{ret(\mathcal{I},\mathcal{D})}$ is exponential in the number of violated constraints, and hence efficient computation of the intersection in (8.2) requires some intelligent strategy. Clearly, the overall approach is beneficial only if the recombination cost does not compensate the gain of repair localization. In the next section, we present an efficient technique for the recombination step which allows us to validate the approach in experiments described subsequently.

8.3 A Technique for Efficient Recombination

In this section, we describe a technique for implementing the recombination step in a way which circumvents the evaluation of Π_q on each repair of $ret(\mathcal{I}, \mathcal{D})$ separately. For the sake of simplicity, we deal here with constraints of the form 8.1, when $l > 0$ and $m = 0$. In this case, according to Proposition 8.2.9, $rep(A_{ret(\mathcal{I},\mathcal{D})})$ coincide with $rep(C_{ret(\mathcal{I},\mathcal{D})})$, and $\mathcal{R} \subseteq C_{ret(\mathcal{I},\mathcal{D})} \subseteq ret(\mathcal{I}, \mathcal{D})$ for each $\mathcal{R} \in rep(C_{ret(\mathcal{I},\mathcal{D})})$. Furthermore, thesis in Theorem 8.2.11 can be rewritten as $ans_{ls}(q, \mathcal{I}, \mathcal{D}) = \bigcap_{\mathcal{R} \in rep(C_{ret(\mathcal{I},\mathcal{D})})} \Pi_q^{\mathcal{R} \cup S_{ret(\mathcal{I},\mathcal{D})}}$.

The basic idea of our approach is to encode all repairs into a single database over which the query Π_q can be evaluated by means of standard database techniques. More precisely, for each global relation r, we construct a new relation r_m by adding an auxiliary attribute *mark*. The mark value is a string $'b_1 \ldots b'_n$ of bits $b_i \in \{0, 1\}$ such that, given any tuple t, $b_i = 1$ if and only if t belongs to the i-th repair $\mathcal{R}_i \in rep(C_{ret(\mathcal{I},\mathcal{D})}) = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$, for every $i \in \{1, \ldots, n\}$ (indexing the repairs is easy, e.g. using the order in which the deductive database system computes them). The set of all marked relations constitutes a *marked* database, denoted by $M_{ret(\mathcal{I},\mathcal{D})}$. Note that the facts in $S_{ret(\mathcal{I},\mathcal{D})}$ (the bulk of data) can be marked without any preprocessing, as they belong to every repair $\mathcal{R}_i$; hence, their mark is $'11 \ldots 1'$. For our running example, the marked database derived from the repairs in Figure 8.2 is shown in Figure 8.3.

We next show how the query Π_q can be reformulated in a way such that its evaluation over the marked database computes the set of certain answers to q. Before proceeding, we point out that our recombination technique also applies in the presence of constraints of general form. In such a case, the source of complexity lies in the computation of $C^*_{ret(\mathcal{I},\mathcal{D})}$.

8.3.1 Query Reformulation

We next focus on non-recursive Datalog queries and provide a technique for rewriting them into SQL. The extension to non-recursive Datalog$^\neg$ queries is straightforward.

Let a query $q(\vec{\mathbf{x}})$, where $\vec{\mathbf{x}} = X_1, \ldots X_n$, be given by the rules $q(\vec{\mathbf{x}}) \leftarrow q_j(\vec{\mathbf{y}}_j)$, $1 \leq j \leq n$, where each $q_j(\vec{\mathbf{y}}_j)$ is a conjunction of atoms $p_{j,1}(\vec{\mathbf{y}}_{j,1}), ..., p_{j,m}(\vec{\mathbf{y}}_{j,m})$, where $\vec{\mathbf{y}}_j = \bigcup_{i=1}^{m} \vec{\mathbf{y}}_{j,i}$. Moreover, let us call each variable Y such that $Y \in \vec{\mathbf{y}}_{j,i}$ and $Y \in \vec{\mathbf{y}}_{j,h}$ a *join variable of* $p_{j,i}$ *and* $p_{j,h}$.

In query reformulation, we use the following functions `ANDBIT` and `SUMBIT`:

- `ANDBIT` is a binary function that takes as its input two bit strings $'a_1 \ldots a'_n$ and $'b_1 \ldots b'_n$ and returns $'(a_1 \wedge b_1) \ldots (a_n \wedge b_n)'$, where $\wedge$ is boolean and;
- `SUMBIT` is an aggregate function such that given m strings of form $'b_{i,1} \ldots b'_{i,n}$, it returns $'(b_{1,1} \vee \ldots \vee b_{m,1}) \ldots (b_{1,n} \vee \ldots \vee b_{m,n})'$, where $\vee$ is boolean or.

Then, each $q(\vec{\mathbf{x}}) \leftarrow q_j(\vec{\mathbf{y}}_j)$ can be translated in the SQL statement SQL_{q_j} of the form

`SELECT` $\mathbf{X_1}, \ldots \mathbf{X_n}$, ($\mathbf{p_{j,1}}$.**mark** `ANDBIT` ... `ANDBIT` $\mathbf{p_{j,m}}$.**mark**) `AS` **mark**
`FROM` $\mathbf{p_1} \ldots \mathbf{p_m}$
`WHERE` $\mathbf{p_{j,i}}.\mathbf{Y}$=$\mathbf{p_{j,h}}.\mathbf{Y}$ (*for each join variable* $\mathbf{Y}$ *of* $\mathbf{p_{j,i}}$ *and* $\mathbf{p_{j,h}}, 1 \leq i, h \leq m$).

In addition to the answers to $q(\vec{\mathbf{x}}) \leftarrow q_j(\vec{\mathbf{y}}_j)$, the statement computes the marks of the repairs in which an answer holds by applying the `ANDBIT` operator to the *mark* attributes of the joined relations. The results of all SQL_{q_j} can be collected into a view uq_m by the SQL statement SQL_{q_1} `UNION` $SQL_{q_2} \ldots$ `UNION` SQL_{q_n}. Finally, SQL_q is

`SELECT` $\mathbf{X_1}, \ldots \mathbf{X_n}$, `SUMBIT`(**mark**)
`FROM` $\mathbf{uq_m}$
`GROUP BY` $\mathbf{X_1}, \ldots \mathbf{X_n}$
`HAVING SUMBIT`(**mark**) $=' 1 \ldots 1'$.

It computes query answers by considering only the facts that belong to all repairs.

Proposition 8.3.1 *Given a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, a source database $\mathcal{D}$ and a query q over $\mathcal{G}$, the set $ans_{ls}(q, \mathcal{I}, \mathcal{D})$ coincides with the set computed by executing SQL_q over the marked database $M_{ret(\mathcal{I},\mathcal{D})}$.*

Example 8.3.2 For $q(X) \leftarrow player(X, Y, Z)$; $q(X) \leftarrow team(V, W, X)$, SQL_q is

```
CREATE VIEW uq_m (X, mark) AS
SELECT Pcode, mark FROM player
  UNION
SELECT Tleader, mark FROM team;
```

```
SELECT X, SUMBIT(mark)
FROM uq_m
GROUP BY X
HAVING SUMBIT(mark) =' 11';
```

It is easy to see that the answers consist of the codes 8, 9, 10. ∎

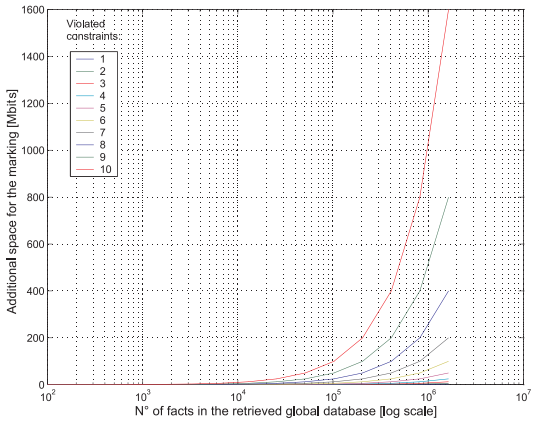


Figure 8.4: Additional space for marking w.r.t. the size of the global database $ret(\mathcal{I}, \mathcal{D})$

8.3.2 Scaling the Technique

Since the number of repairs is exponential in the number of violated constraints, the marking string can be of considerable length. For instance, 10 constraint violations, involving two facts each, give rise to 2^{10} repairs, and hence 1 Kbit is needed for marking each tuple.

Figure 8.4 shows the additional space needed in our running example, depending on the size of the retrieved global database, $ret(\mathcal{I}, \mathcal{D})$, for different numbers of constraint violations (assuming each violation has two conflicting facts). This clearly limits the effectiveness of the technique. For instance, a retrieved global database of 'only' 100000 facts requires almost 100 Mbits additional space, most of it for marking the safe part.

We thus refine our technique to mark only the affected database part. More specifically, we split each global relation r into a "safe" part $r_{safe}^{ret(\mathcal{I},\mathcal{D})} = \{t \mid r(t) \in S_{ret(\mathcal{I},\mathcal{D})}\}$ and an "affected" part $r_{aff}^{ret(\mathcal{I},\mathcal{D})} = \{t \mid r(t) \in A_{ret(\mathcal{I},\mathcal{D})}\}$.

With this approach, the additional space depends only on the size of $A_{ret(\mathcal{I},\mathcal{D})}$ but not on the size of $S_{ret(\mathcal{I},\mathcal{D})}$. For example, for 10 constraint violations involving two tuples each, the additional space is bound by 20 Kbits, independently of the size of $ret(\mathcal{I}, \mathcal{D})$.
We now need to use a different rewriting of q , obtained as follows:

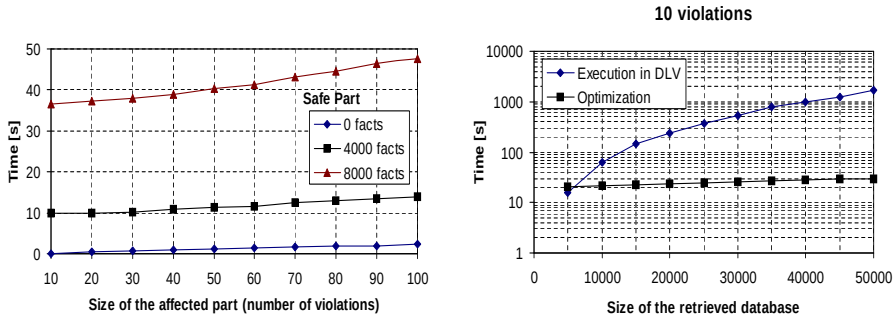
– in each rule $q(\vec{\mathbf{x}}) \leftarrow q_j(\vec{\mathbf{y}}_j)$, we replace each atom $p(\vec{\mathbf{y}})$, with $p_{aff}(\vec{\mathbf{y}}) \vee p_{safe}(\vec{\mathbf{y}})$;

– we then rewrite each resulting formula into disjunctive normal form $q_{j,1}(\vec{\mathbf{z}}) \vee \ldots \vee q_{j,n_j}(\vec{\mathbf{z}})$ and replace each $q(\vec{\mathbf{x}}) \leftarrow q_j(\vec{\mathbf{y}}_j)$ with the rules $q(\vec{\mathbf{x}}) \leftarrow q_{j,i}(\vec{\mathbf{z}})$ for each $1 \leq i \leq n_j$.

– each $q_{j,i}(\vec{\mathbf{z}})$ has form $p_{safe_{j,1}}(\vec{\mathbf{z}}_1), \ldots, p_{safe_{j,n}}(\vec{\mathbf{z}}_n), p_{aff_{j,n+1}}(\vec{\mathbf{z}}_{n+1}), \ldots, p_{aff_{j,m}}(\vec{\mathbf{z}}_m)$, and evaluated by a statement $SQL_{q_{j,i}}$, where $'1 \ldots 1'$ replaces each $p_{safe_{j,k}}.mark$.

Notice that this rewriting is exponential in the size of q (more precisely, in the number of atoms). However, as commonly agreed in the database community, the overhead in query complexity usually pays off the gain in data complexity.

It should be easy to see that, by allotting 5 Mbytes marking space, our technique may scale up to 20 violated constraints involving two tuples each. With this marking space, any number



(a) (b)

Figure 8.5: (a) Execution time in DLV system w.r.t. $|A_{ret(\mathcal{I}_0,\mathcal{D}_{syn})}|$, for different sizes of $S_{ret(\mathcal{I}_0,\mathcal{D}_{syn})}$. (b) Comparison with the optimization method.

of violations n can be handled by evaluating q incrementally over a sequence of partially marked databases $M^i_{ret(\mathcal{I},\mathcal{D})}$, $1 \leq i \leq \lceil n/20 \rceil$, grouping the marks of 20 repairs at a time, i.e., each relation r in $M^i_{ret(\mathcal{I},\mathcal{D})}$ is marked with the bits $'b_{20(i-1)+1} \ldots b'_{20i}$ of $'b_1 \ldots b'_n$.

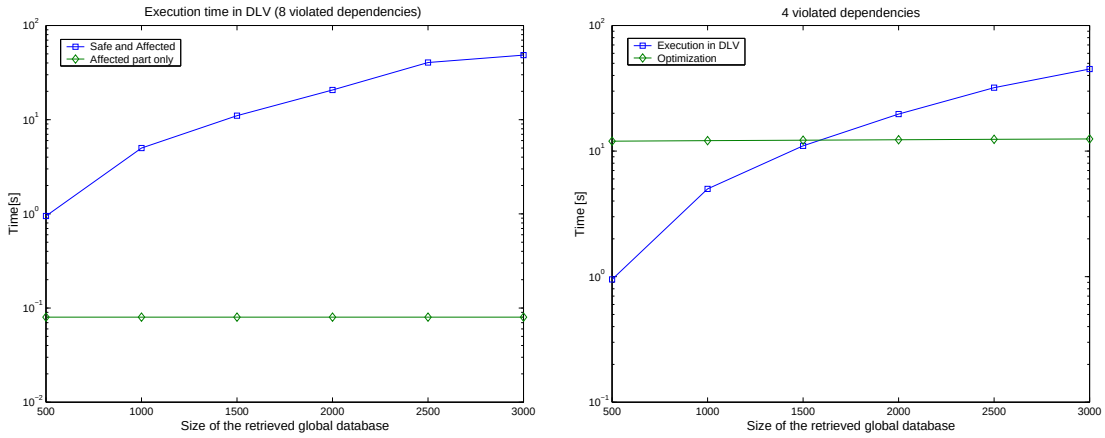
8.4 Experimental Results

In this section, we present experimental results obtained by means of a prototype implementation that couples the DLV deductive database system [109] with Postgresql, a relational DBMS which allows for a convenient encoding of the `ANDBIT` and `SUMBIT` operators. Notice that DLV supports disjunction, which is needed for encoding universal constraints into programs Π_Σ, since computing certain answers in this setting is Π^p_2-complete [86]. The first experiment has been carried out on a sun4u sparc SUNW ultra-5_10 with 256MB memory and processor working at 350MHz, under Solaris SUN operating system, whereas the second one has been executes on a 1200MHz - 256MB AMD processor under the Linux operating system.

Football Teams. For our running example, we built a synthetic data set $\mathcal{D}_{syn}$, in which the facts in *coach* and *team* satisfy the key constraints, while facts in *player* violate it. Each violation consists of two facts that coincide on the attribute *Pcode* but differ on the attributes *Pname* or *Pteam*; note that these facts constitute $A_{ret(\mathcal{I}_0,\mathcal{D}_{syn})}$. For the query $q(X) \leftarrow player(X,Y,Z)$; $q(X) \leftarrow team(V,W,X)$, we measured the execution time of the program $\Pi_{\mathcal{I}_0}(q)$ in DLV depending on $|A_{ret(\mathcal{I}_0,\mathcal{D}_{syn})}|$ for different fixed values of $|S_{ret(\mathcal{I}_0,\mathcal{D}_{syn})}|$, viz. (i) 0, (ii) 4000, and (iii) 8000. The results are shown in Fig. 8.5.(a).

We point out that in Case (i), in which $\Pi_{\mathcal{I}_0}(q)$ is evaluated only on the affected part, the execution time scales well to a significant number of violations. On the other hand, the evaluation of $\Pi_{\mathcal{I}_0}(q)$ on the whole database is significantly slower; in fact, a small database of 8000 facts requires up to 40 seconds for 50 violated constraints.

Figure 8.5.(b) shows a comparison (in log scale) between the consistent query answering by a single DLV program and the optimization approach proposed in this paper. Interestingly, for a



(a) (b)

Figure 8.6: (a) Execution time w.r.t. the size of the global database: advantages of the decomposition step. (b) Execution time in DLV and in the optimization method.

fixed number of violations (10 in the figure) the growth of the running time of our optimization method under varying database size is negligible. In fact, a major share ($\sim$ 20 seconds) is used for computing repairs of the affected database, plus marking and storing them in Postgresql; the time for query evaluation itself is negligible. In conclusion, for small databases (up to 5000 facts), consistent query answering by straight evaluation in DLV may be considered viable; nonetheless, for larger databases, the asymptotic behavior shows that our approach outperforms a naive implementation

3Coloring. As a further example, we encoded the classical NP-complete graph 3-coloring problem into querying a data integration system $\mathcal{I}_1 = \langle \mathcal{G}_1, \mathcal{S}_1, \mathcal{M}_1 \rangle$. The global schema $\mathcal{G}_1$ has relations $edge(Node, Node)$ and $colored(Node, Color)$, and $Node$ is the key for $colored$.

For a retrieved global database $\mathcal{DB}$, we fixed a number of nodes and generated facts in $edge$ producing a graph; moreover, for each node i, we generated three facts: $colored(\mathsf{i}, \mathsf{red})$, $colored(\mathsf{i}, \mathsf{blue})$, $colored(\mathsf{i}, \mathsf{yellow})$. Clearly $\mathcal{DB}$ is inconsistent, and due to the key constraint only one of the three facts can remain in the database. The remaining nodes are precolored, i.e., for them only one of the three facts above is in $\mathcal{DB}$.

Now consider the query $q \leftarrow edge(X, Y), colored(X, C), colored(Y, C)$. As easily seen, it evaluates to true on $\mathcal{DB}$ iff there is no legal 3-coloring for the graph in $\mathcal{DB}$.

Figure 8.6.(a) reports the execution time for deriving certain answers by looking at the whole database $\mathcal{DB}$ and for computing the repairs only of the affected part of $\mathcal{DB}$, varying the size of $\mathcal{DB}$. Obviously, our approach requires some further effort in the recombination: Figure 8.6.(b) reports the time needed for the whole process (almost 10 seconds).

We stress that the results can be significantly improved since our implementation is not optimized. Nonetheless, its advantage over the standard technique is evident.

8.5 Discussion

In this chapter, we have described an approach to speed up the evaluation of non-monotonic logic programs modelling query answering in data integration systems. To this end, we have provided theoretical results that allow for repairing an inconsistent retrieved database by localizing the computation of repairs to its affected part. As we have shown, for important classes of constraints such as key constraints, functional dependencies, and exclusion dependencies, repairs can be restricted to the facts in the database violating the constraints, which may be only a small fragment of a large database. Furthermore, we have developed a technique for recombining such repairs to provide answers for user queries. Finally, we have experimented the viability of our approach, considering two different scenarios.

Notice that our method, which is built upon repair by selection in terms of a particular preference ordering, is based on abstract properties and may be adapted to other logic programming systems as well. Furthermore, it can be generalized to other preference based repair semantics for an inconsistent database $\mathcal{DB}$. In particular, all repair semantics in which $\triangle(\mathcal{R}, \mathcal{DB}) \subset \triangle(\mathcal{R}', \mathcal{DB})$, i.e., $\mathcal{R}$ is closer to database $\mathcal{DB}$ than $\mathcal{R}'$ in terms of symmetric difference, implies that $\mathcal{R}$ is preferred to $\mathcal{R}'$ can be dealt with using our method. For instance, cardinality-based and weighted-based repair semantics satisfy this *non-redundancy* condition.

Notice that the logic formalization of LAV systems proposed in [13, 16] might be captured by our logic framework under suitable adaptations. Actually, given a source extension, several different ways of populating the global schema according to a LAV mapping may exist, and the notion of repair has to take into account a set of such global database instances. Nonetheless, analogously to the GAV framework addressed in this chapter, in [13, 16] the repairs are computed from the stable models of a suitable logic program.

We point out that the optimization technique provided in this chapter perfectly applies to the approach to query answering in the presence of constraints in GAV integration systems under the loosely-sound semantics described in Chapter 6. Indeed, as shown in Section 8.1.1, compiling the IDs into the user query, which is suitably rewritten by means of the ID-rewrite algorithm, allows us to carry out query answering considering only KDs and EDs, which are universally quantified constraints, as the ones addressed in this chapter. Furthermore, in such a case, loosely-sound semantics and loosely-exact semantics coincide, and the domain of discourse adopted can be indifferently considered finite of infinite.

Chapter 9

The DIS@DIS System

In this chapter we present DIS@DIS[1], a state-of-the-art prototype for semantic data integration, that is capable of reasoning about integrity constraints in order to improve query answering. The problem of designing effective tools for data integration has been addressed by several research and development projects in the last years [49, 42, 93, 138, 161, 154, 100, 84, 12], some of which have been described in Section 4. As already noticed, most of such systems, and in particular those considering the GAV framework, do not allow for the specification of integrity constraints on the global schema modelling the integration application, or do not properly take into account the presence of such constraints for query answering. Moreover, there are no systems that provide support for managing data inconsistency and incompleteness with respect to such constraints, for both the GAV and the LAV scenario.

As widely discussed in this thesis, since the global schema is a representation of the domain of interest of integration applications, integrity constraints should be allowed on it in order to enhance its expressiveness, thus improving its capability of representing the real world. To this aim, DIS@DIS enables for the design of a relational global schema with several forms of constraints, namely, key, inclusion, exclusion dependencies. Furthermore, DIS@DIS is able to properly deal with inconsistent and incomplete data.

More specifically, to deal with incompleteness, DIS@DIS adopts the sound semantics, that we have introduced in Section 3.2, in which global databases constructed according to the mapping are interpreted as subsets of the databases that satisfy the global schema. Query processing under the sound semantics provides answers that are logical consequence of the global schema and the information available at the sources.

Furthermore, in order to also cope with inconsistency w.r.t. KDs and EDs, the system resorts to the loosely-sound semantics described in Section 3.3, thus allowing for minimal repairs of the sound global database instances. Query processing under the loosely-sound semantics allows for computation of consistent answers from incomplete and inconsistent data.

With regard to the mapping specification, DIS@DIS allows for the design of both GAV and LAV applications. According to the integration framework described in Chapter 3, the language

[1] The name of the system is an acronym for *Data Integration System at the Dipartimento di Informatica e Sistemistica*

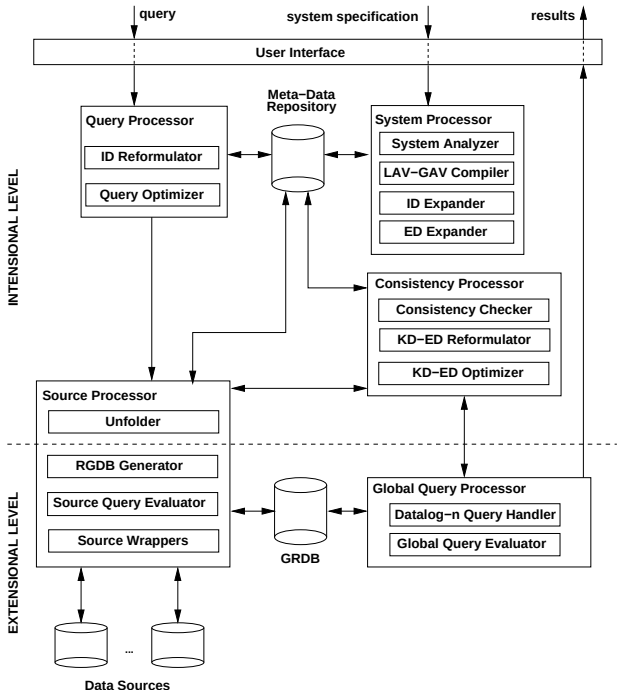


Figure 9.1: Architecture of the system

adopted for queries in the GAV mappings is non-recursive Datalog$^{\neg}$, whereas for LAV mappings we consider queries that are conjunctive queries. Notice that the above languages correspond to fragments of SQL queries (aggregate operations or subqueries are not allowed).

In the current version of the system, query processing under the loosely-sound semantics is supplied for GAV specifications, whereas for LAV specifications only query processing under the strictly-sound semantics is allowed. More specifically, for GAV, DIS@DIS implements the algorithms described in Chapter 6 for query processing in the presence of IDs, KDs and EDs. For LAV, DIS@DIS implements the off-line compiling technique, described in Section 7.2.1, that transforms a LAV system specification into a query-equivalent GAV one, in such a way that the query processing techniques defined for GAV under the strictly-sound semantics could be applied also in the LAV case, as described in Section 7.2.2. Notice that in such case, KDs on the global schema are not allowed.

As already said in Chapters 6 and 7, query processing in DIS@DIS is essentially carried out at the intensional level, since both the treatment of integrity constraints and the off-line compilation from LAV to GAV are done at the query and schema level. This strongly augments the feasibility of the approach, which is also witnessed by first encouraging experiments, and allows for interesting optimizations.

9.1 System Architecture

In this section we briefly present the architecture of DIS@DIS. The high-level architecture of the system is displayed in Figure 9.1.

In the system, the *Meta-Data Repository* stores an internal representation of the intensional information, i.e., the system specification and the user queries, while the *retrieved global database (RGDB)* is an internal relational database that, based on the data at the sources, materializes the relevant portion of the global relations for answering the query. Also, a *User Interface* takes care of the interaction with the system user.

The system architecture comprises five main modules:

1. the *System Processor* elaborates and reasons about the specification of the data integration system;
2. the *Query Processor* has the task of elaborating the user queries, according to IDs expressed on the global schema;
3. the *Source Processor* reformulates the query on the global schema taken as input from the Query Processor in terms of a query over the sources. It also executes the reformulated query, by retrieving at the sources the data relevant for the query, and stores such data in the RGDB. In this module, source wrappers provide a relational view of the sources, called *source schema*;
4. the *Consistency Processor* is activated only when IDs or EDs are defined on the global schema. It detects the situations in which data stored in the retrieved global database are not consistent with the key dependencies and the exclusion dependencies and, if needed, computes a further reformulation of the query. Access to RGDB is through the Global Query Processor;
5. the *Global Query Processor* evaluates the reformulated query over the RGDB.

Basically, there are two kinds of input to the system, i.e., *(i)* the data integration system specification, constituted by the global schema with integrity constraints, the source schema and the mapping, and *(ii)* a user query, which is a UCQ expressed over the global schema. The system output is the answer to the query. The following is a high-level view of the behaviour of the system:

(i) When the data integration system specification is inserted or updated, the System Processor analyzes such a specification. Notice that all the tasks performed by the System Processor are computed off-line, since all the computation is independent of the query that has to be processed.

(ii) When a query Q is issued by the user, the following steps are executed:

1. first, the Query Processor reformulates the query Q according to the inclusion dependencies expressed over the global schema, thus obtaining a new query Q_1;
2. then, the Source Processor, based on the query Q_1, retrieves at the sources the portion of the data that is relevant for the query[2]. In this way, an instance of the relations in the global schema is created and materialized in the retrieved global database RGDB;

[2]Relevant data are computing according to Definition 8.2.1

3. the RGDB is then analyzed by the Consistency Processor, which checks whether key dependencies and exclusion dependencies expressed over the global schema are violated by the data in the RGDB. In this case, it computes a further reformulation of the query Q_1, thus producing a new query Q_2;
4. finally, the Global Query Processor evaluates the query Q_2 with respect to the RGDB, which corresponds to computing the answers to the initial query Q.

It is important to notice that the whole computation is "pushed up" as much as possible at the intensional level. Indeed, in the whole Query Processor module and, partially, in the Source Processor and Consistency Processor, the computation is performed at the intensional level. Moreover, the system tries to minimize the access to the data sources and to retrieve the minimal subset of data that are actually needed in order to answer the user query.

We point out that, in the current implementation, DIS@DIS is able to completely perform all the above steps only for GAV system specifications. Indeed, it should be easy to see that, for such systems, Q_1 is actually the UCQ Π_{ID} produced by the algorithm ID-rewrite described in Section 6.3.1, whereas Q_2 coincides with the Datalog$^\neg$ program $\Pi_{ID} \cup \cup \Pi_{KD} \cup \Pi_{ED}$ described in section 6.4, which provides a suitable encoding of the EDs and KDs under the loosely-sound semantics. Notice that $\Pi_{\mathcal{MD}}$ is not needed here, since the computation of the RGDB has been already performed at step 2.

Conversely, for LAV system specifications, whereas step 1, 2 and 4 are are entirely executed, at step 3 the reformulation of Q_1 into Q_2 is never computed. Indeed, this part of the methodology is concerned with query answering in LAV under the loosely-sound semantics, which is a research topic currently under investigation. Furthermore, we underline that query rewriting for LAV systems needs a preliminary processing which compile the LAV system into a query-equivalent GAV one, as described in Section 7.2.1. Hence query reformulation at step 2 is actually carried out by taking into account also the new global relations and IDs introduced in the transformation phase. According to Theorem 7.2.2 and the discussion in Section 7.2.3, such a technique can be applied only in the absence of KDs on the global schema. In the presence of such dependencies, we have to resort to a different query answering procedure that relies on the construction of the retrieved global database in LAV making use of the KD-RULE, as described in Section 7.1.1. Such a technique is currently under implementation.

9.2 System Description

In this section we briefly describe the main components of the system.

9.2.1 System Processor

System Analyzer This module verifies the properties of the data integration system. In particular: *(i)* it verifies whether the system is non-key-conflicting; *(ii)* based on the constraints and the mapping, it establishes whether two global relations in a GAV or LAV system are equivalent or contained one into the other.

As concerns the first of the above points, in Chapter 5 it has been shown that, under the loosely-sound semantics (which is the one adopted in our system), the problem of query answering under arbitrary KDs and IDs is in general undecidable: however, it is decidable if the integrity constraints in the global schema belong to the class of non-key-conflicting dependencies. Indeed, if KDs and IDs are non-key-conflicting, they can be processed separately (in this case we say that the system is *separable*). Thus, the System Analyzer performs a *separability check* over the integrity constraints that are stored in the Meta-Data Repository; if the system is not separable, the analyzer provides the designer with a report of the constraints that are not non-key-conflicting.

LAV-GAV Compiler The task of the LAV-GAV Compiler is to transform a specification of a LAV system into a specification of GAV system, according to the rules of Section 7.2.1, which is equivalent to the initial one w.r.t. query processing. The transformation is performed in a purely intensional fashion, and only once for any integration system specification (unless the specification changes). Therefore, we can see this process as a "compilation" of a LAV system into an equivalent GAV one.

Both the LAV system specification that is to be transformed, and the GAV system obtained from the compilation are stored in Meta-Data repository.

ID Expander In order to speed up the query reformulation phase performed by the ID Reformulator, we process the inclusion dependencies by computing the closure of them w.r.t. implication [2]. We will explain later how the computation of the closure of IDs is useful to make query rewriting process more systematic and efficient. The computational cost of this calculation is, in the worst case, exponential in the size of the initial set of IDs; however, since this operation is to be performed ony once, i.e. when the specification of a system is loaded, such an exponential blow-up is of not critical from a practical point of view. In order to further optimize the computation of the closure, indexing structures are used: their introduction allows us to avoid useless application on the implication rule. The process terminates when no new IDs can be added.

ED Expander The ED Expander has the task of inferring new exclusion dependencies starting from the given set of EDs and IDs. Such dependencies are obtained through a recursive inference procedure that computes the closure of the initial set of EDs w.r.t. implication by EDs and IDs, as described in Section 5.2.1. Such a process is necessary to efficiently deal with inconsistent data, since it reduces the computational cost of the operations performed by the Consistency Processor.

9.2.2 Query Processor

ID Reformulator This module implements the algorithm ID-rewrite, described in Section 6.3.1, by producing a rewriting of any user query posed to the system according to the IDs on the global schema.

The rewriting is performed by iteratively applying to the conjunctive queries in the user query the functions reduce and atom-rewrite, that correspond to steps (a) and (b) of the algorithm ID-rewrite: the former applies unification, producing rewritings that are shorter than the original

query; the latter takes inclusion dependencies into account, encoding them into the rewriting.

Considering that the inclusion set has been previously closed w.r.t. logical implication by the ID Expander, the algorithm implemented by the ID Reformulator is actually an optimization of ID-rewrite, and it can be summarized as follows: conjunctive queries are grouped in subsets, sorted according to the number of atoms present in the body; then, we apply reduce and atom-rewrite in all possible ways, proceeding from the query set with maximum length to shorter queries. Different optimizations are implemented in the ID Reformulator. The most important regards the unification process: since the algorithm halts when no new query can be generated, the time needed to compare queries becomes a crucial issue in order to guarantee a reasonable computation time. In particular, the internal representation of conjunctive queries in the ID Reformulator allow us to compare queries by a straightforward string comparison; the first tests performed on the system have shown that this approach significantly improves the overall efficiency of the module.

Query Optimizer The Query Optimizer performs an optimization of the rewritten query obtained by the ID Reformulator. In particular, it can check *a priori* whether a query will not produce any result; such queries are immediately excluded. This module is currently under development.

9.2.3 Source Processor

Unfolder This module unfolds the query produced by the ID Reformulator according to the mapping assertions, i.e., expresses the query in terms of the source relations. To this aim, it is necessary to apply a unification mechanism. Such a module is used to provide answers to queries in the cases in which no constraints or only IDs are specified on the global schema of either GAV or LAV systems. Indeed, in such cases there is non need ot chack the consistency w.r.t. EDs and KDs, and we can simply unfold the query produced by the ID reformulator (or the original query in the absence of IDs), in order to obtain the certain answers. In the presence of also EDs or KDs, such a module is exploited for computing the RGDB, as shown below.

RGDB Generator The RGDB Generator builds the RGDB from the system specification, the user query, and the source data. For GAV systems, the mapping provides a procedural method to access the sources, thus the generator invokes the unfolder over the mapping queries that are relevant for the user query, and then uses the obtained rewritings to query the sources, thus retrieving the relevant data for answering the query; such data are then stored in the RGDB. For LAV systems, the generator builds the RGDB by extracting data from the sources tuple by tuple, and suitably adding tuples in the global relations; during this process, fresh constants are in general introduced, which, according to the key dependencies, may need to be changed into constants of the source database or into other fresh constants. The construction of the retrieved global database in LAV in the presence of KDs is a system capability currently under implementation.

Source Wrappers Such modules provide a relational interface to data sources, which in general may be heterogeneous and non-relational. Also this module is currently under development.

9.2.4 Consistency Processor

Consistency Checker The Consistency Checker analyzes the RGDB in order to detect the presence of violations of key and exclusion dependencies over the global schema. In such a case, the KD-ED Reformulator is invoked.

KD-ED Reformulator This module is activated only if the Consistency Checker detects inconsistencies with respect to KDs and EDs expressed over the global schema. In such a case, the query is further reformulated in Datalog$^\neg$, according to the rules presented in Section 6.4. As explained in that section, such a program encodes at the intensional level the repairing of the RGDB according to the loosely-sound semantics. Such a rewriting can thus be seen as an alternative to using data cleaning techniques over the data retrieved at the sources.

KD-ED Optimizer This module has the task of optimizing the rewriting produced by the KD-ED Reformulator. Indeed, it restricts the rewritten query by eliminating those parts of the query that cannot actually contribute to the answer, by checking *a priori* whether a subquery will not produce any result. This is done according to the pruning step of our optimization method described in Section 8.2.1. This module is currently under development.

9.2.5 Global Query Processor

The task of this module is to evaluate the perfect rewriting generated by the Query Processor and the Consistency Processor over the RGDB. If the perfect rewriting is a Datalog query with negation, then the evaluation is performed by the *Datalog-n-Query Handler*, that makes use of the Disjunctive Datalog engine DLV [68], otherwise the query is a standard union of conjunctive queries, therefore the perfect rewriting can be computed by the *Global Query Evaluator* that translates the query in SQL and then passes this query to the DBMS (currently, MySQL) that stores the RGDB. Notice that Datalog-n-Query Handler is in charge of also manage the decomposition and recombination steps of the optimization technique described in Chapter 8. The implementation of this technique is currently under development. We point out that the experiments shown in Section 8.4 have been carried out outside the DIS@DIS system.

9.3 Discussion

To the best of our knowledge, DIS@DIS is the first system that enables query processing under constraints in both LAV and GAV specifications and that is able to deal with incomplete and inconsistent data. However, the main goal of the system is to experiment the effectiveness of the intensional approach to query processing, widely discussed in this thesis. Indeed, pushing up query processing as much as possible at the intensional level allows in principle to minimize source accesses, which typically represent a bottleneck in data integration systems.

As shown in Section 9.2, query processing under the sound semantics relies on standard relational database technology, whereas resorting to a Datalog$^\neg$ engine, which causes overhead in computation, is required for answering queries under the loosely-sound semantics.

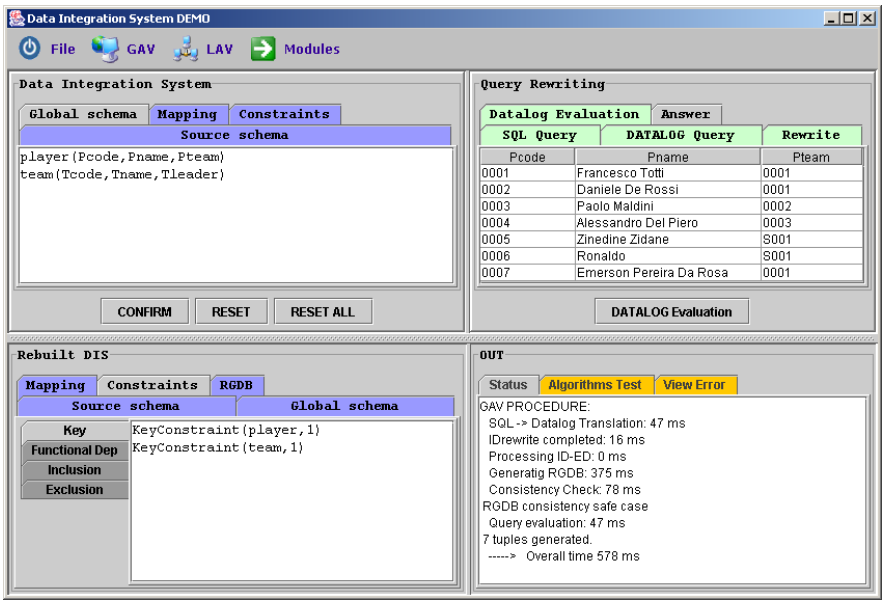


Figure 9.2: User interface of the system

The system has been developed in Java. A limited version of the system (and some integration scenarios) can be accessed on the WWW at the URL `http://www.dis.uniroma1.it/~disatdis`. A screenshot of the graphical user interface of DIS@DIS is shown in Figure 9.2.

Chapter 10

Conclusions

In this thesis we have addressed data integration issues in the presence of incomplete and inconsistent data with respect to key, inclusion and exclusion dependencies specified on the relational global schema of both LAV and GAV data integration systems. More specifically, we have studied the problem of query answering in this setting, for different combinations of such constraints, under different semantics. First of all, we have investigated query answering under the sound semantics, and we have shown that when IDs and KDs are issued on the global schema, the problem in the general case is undecidable. Then, we have provided the maximum class of IDs for which the problem is decidable in the presence of KDs, i.e., the class of NKCIDs. We have also shown that NKCIDs and KDs can be processed separately in order to provide certain answers to user queries. Interestingly, the same property holds also in the presence of EDs on the global schema: in this case this means that we can solve query answering by separately taking into account the set of IDs and the set of KDs and EDs that are logical consequences of the IDs and the original EDs. Then, we have addressed query answering under the loosely-sound and the loosely-exact semantics, two suitable relaxations of the sound and the exact semantics that allow us to properly deal with inconsistent data. We have shown that query answering is undecidable for general IDs and KDs under these two loose semantics, and that for NKC relational schemas the separation property of query answering holds in such cases.

Then, we have provided an in-depth study of the computational complexity of the problem under these three semantics. Table 5.1 of Chapter 5 summarizes the results for query answering under constraints given in these thesis. We point out that such results, that refer to the context of a single relational schema, have been completely extended in the thesis to the GAV integration framework, and partially to the LAV framework. In particular, we have not provided algorithms for query answering in LAV under the loose semantics, but in this framework we have defined methods for query answering under the sound semantics. Then we have concentrated on the problem of query rewriting both in LAV and in GAV. In particular, we have provided effective query rewriting techniques in GAV under the loosely-sound semantics in the presence of IDs, KDs and EDs, and in LAV under the sound semantics in the presence of IDs and EDs. In order to speed-up the evaluation of logic programs produced by our techniques for query rewriting under the loosely-sound semantics, we have described an optimization method that allows for

efficient query answering in the presence of inconsistent data. Actually, Such a method applies also to the class of universally quantified constraints. Finally, we have presented the DIS@DIS system, a data integration prototype incorporating the techniques described in the thesis.

The aim of our ongoing and future research is to investigate the integration settings that we have not completely analyzed in this thesis. In the following we summarize some ongoing works and possible future research directions.

- As for the LAV framework, we are currently studying query answering under the loosely-sound semantics, as well as query rewriting in the presence of IDs. As said in Section 7.2.3, the need of recursion in the latter case significantly complicates the problem with respect to the setting in which IDs and EDs are specified on the global schema.

- With respect to the study on the decidability of query answering under constraints, we are analyzing the interaction between functional and inclusion dependencies, which represents an interesting and challenging theoretical problem. Indeed, even if it is well-known that the query answering is undecidable in the general case, it is still not clear which are the maximal classes of dependencies for which the problem is decidable. Our first studies on this subject show that in this case, differently from the case of IDs and KDs, "non-separable" classes exist that are still decidable.

- Connected to the above problem, is the study of query answering under the constraints imposed by the Entity-Relationship (ER) model. Since ER schemas represent simple forms of ontologies, providing query answering techniques in this setting means shedding light on the problem of integration of ontologies, to which much attention has been recently devoted. We point out that IDs, KDs, and EDs studied in this thesis represent the core constraint language of the ER model.

- We are currently working on the DIS@DIS systems in order to (i) provide facilities for query answering under the loosely-sound semantics for LAV mappings, (ii) allow for GLAV mappings (a generalized form of mapping that comprises GAV and LAV as special cases);(iii) develop optimization techniques to further minimize access to the sources and use of the Datalog$^\neg$ engine; (iv) finalize the components currently under development. In this respect, an aspect deserving particular attention is the interface towards the sources: providing support for the design of ad-hoc source wrappers, possibly equipped with proper data cleaning facilities are some of our priorities.

- Finally, we aim at extending the framework to allow for the definition of suitable parameters that indicate the reliability and the "confidence" of a source with respect to the information that it exports. This should allow us to indicate, for example, different levels for the quality of data in different sources. Such a specification may lead us to not only consider certain answers to queries, but also answers of different levels of reliability.

Bibliography

[1] Abiteboul, S., and Duschka, O. Complexity of answering queries using materialized views. In *Proceedings of the Seventeenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'98)* (1998), pp. 254–265.

[2] Abiteboul, S., Hull, R., and Vianu, V. *Foundations of Databases*. Addison Wesley Publ. Co., Reading, Massachussetts, 1995.

[3] Abiteboul, S., Quass, D., McHugh, J., Widom, J., and Wiener, J. L. The Lorel query language for semistructured data. *International Journal on Digital Libraries 1*, 1 (1997), 68–88.

[4] Afrati, F. N., Gergatsoulis, M., and Kavalieros, T. Answering queries using materialized views with disjunction. In *Proceedings of the Seventh International Conference on Database Theory (ICDT'99)* (1999), vol. 1540 of *Lecture Notes in Computer Science*, Springer, pp. 435–452.

[5] Aho, A. V., Sagiv, Y., and Ullman, J. D. Equivalence among relational expressions. *SIAM Journal on Computing 8* (1979), 218–246.

[6] Alchourrón, C. E., Gärdenfors, P., and Makinson, D. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic 50* (1985), 510–530.

[7] Arenas, M., Bertossi, L. E., and Chomicki, J. Consistent query answers in inconsistent databases. In *Proceedings of the Eighteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'99)* (1999), pp. 68–79.

[8] Arenas, M., Bertossi, L. E., and Chomicki, J. Specifying and querying database repairs using logic programs with exceptions. In *Proceedings of the Fourth International Conference on Flexible Query Answering Systems (FQAS 2000)* (2000), Springer, pp. 27–41.

[9] Barceló, P., and Bertossi, L. Reapairing databases with annotated predicate logic. In *Proceedings of the Tenth International Workshop on Non-Monotonic Reasoning (NMR 2002)* (2002), pp. 160–170.

[10] Beeri, C., Levy, A. Y., and Rousset, M.-C. Rewriting queries using views in description logics. In *Proceedings of the Sixteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'97)* (1997), pp. 99–108.

[11] Beneventano, D., Bergamaschi, S., Castano, S., Corni, A., Guidetti, R., Malvezzi, G., Melchiori, M., and Vincini, M. Information integration: the MOMIS project demonstration. In *Proceedings of the Twentysixth International Conference on Very Large Data Bases (VLDB 2000)* (2000).

[12] Bergamaschi, S., Castano, S., Vincini, M., and Beneventano, D. Semantic integration of heterogeneous information sources. *Data and Knowledge Engineering 36*, 3 (2001), 215–249.

[13] Bertossi, L., Chomicki, J., Cortes, A., and Gutierrez, C. Consistent answers from integrated data sources. In *Proceedings of the Sixth International Conference on Flexible Query Answering Systems (FQAS 2002)* (2002), pp. 71–85.

[14] Bouzeghoub, M., Fabret, F., Galhardas, H., Matulovic-Broqué, M., Pereira, J., and Simon, E. Data warehouse refreshment. In *Fundamentals of Data Warehouses*, M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassiliadis, Eds. Springer, 1999, pp. 47–86.

[15] Bouzeghoub, M., and Lenzerini, M. Introduction to the special issue on data extraction, cleaning, and reconciliation. *Information Systems 26*, 8 (2001), 535–536.

[16] Bravo, L., and Bertossi, L. Logic programming for consistently querying data integration systems. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI 2003)* (2003). To appear.

[17] Bry, F. Query answering in information systems with integrity constraints. In *IFIP WG 11.5 Working Conf. on Integrity and Control in Information System* (1997), Chapman & Hall.

[18] Buneman, P. Semistructured data. In *Proceedings of the Sixteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'97)* (1997), pp. 117–121.

[19] Buneman, P., Davidson, S., Hillebrand, G., and Suciu, D. A query language and optimization technique for unstructured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1996), pp. 505–516.

[20] Calì, A. *Query Answering and Optimisation in Data Integration Systems.* PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 2002.

[21] Calì, A., and Calvanese, D. Optimized querying of integrated data over the Web. In *Proc. of the IFIP WG8.1 Working Conference on Engineering Information Systems in the Internet Context (EISIC 2002)* (2002), Kluwer Academic Publisher, pp. 285–301.

[22] Calì, A., Calvanese, D., De Giacomo, G., and Lenzerini, M. Accessing data integration systems through conceptual schemas. In *Proceedings of the Twentieth International Conference on Conceptual Modeling (ER 2001)* (2001), pp. 270–284.

[23] Calì, A., Calvanese, D., De Giacomo, G., and Lenzerini, M. On the expressive power of data integration systems. In *Proceedings of the Twentyfirst International Conference on Conceptual Modeling (ER 2002)* (2002).

[24] Calì, A., Calvanese, D., De Giacomo, G., and Lenzerini, M. Data integration under integrity constraints. *Information Systems* (2003). To appear.

[25] Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M., Naggar, P., and Vernacotola, F. IBIS: Data integration at work. In *Proceedings of the Fifteenth International Conference on Advanced Information Systems Engineering (CAiSE 2003)* (2003).

[26] Calì, A., De Nigris, S., Lembo, D., Messineo, G., Rosati, R., and Ruzzi, M. DIS@DIS: a system for semantic data integration under integrity constraints. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE 2003)* (2003). To appear.

[27] Calì, A., De Nigris, S., Lembo, D., Messineo, G., Rosati, R., and Ruzzi, M. `DIS@DIS` homepage, 2003. `http://www.dis.uniroma1.it/~disatdis`.

[28] Calì, A., Lembo, D., Lenzerini, M., and Rosati, R. Source integration for data warehousing. In *Multidimensional Databases: Problems and Solutions*, M. Rafanelli, Ed. Idea Group Publishing, 2003, ch. 10.

[29] Calì, A., Lembo, D., and Rosati, R. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proceedings of the Twentysecond ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS 2003)* (2003), pp. 260–271.

[30] Calì, A., Lembo, D., and Rosati, R. Query rewriting and answering under constraints in data integration systems. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI 2003)* (2003). To appear.

[31] Calvanese, D., Castano, S., Guerra, F., Lembo, D., Melchiori, M., Terracina, G., Ursino, D., and Vincini, M. Towards a comprehensive methodological framework for semantic integration of heterogeneous data sources. In *Proceedings of the Eighth International Workshop on Knowledge Representation meets Databases (KRDB 2001)* (2001), CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-45/`.

[32] Calvanese, D., De Giacomo, G., and Lenzerini, M. On the decidability of query containment under constraints. In *Proceedings of the Seventeenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'98)* (1998), pp. 149–158.

[33] Calvanese, D., De Giacomo, G., and Lenzerini, M. Answering queries using views over description logics knowledge bases. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)* (2000), pp. 386–391.

[34] Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., and Rosati, R. Data integration in data warehousing. *International Journal of Cooperative Information Systems 10*, 3 (2001), 237–271.

[35] Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. Y. Rewriting of regular expressions and regular path queries. In *Proceedings of the Eighteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'99)* (1999), pp. 194–204.

[36] Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. Y. Answering regular path queries using views. In *Proceedings of the Sixteenth IEEE International Conference on Data Engineering (ICDE 2000)* (2000), pp. 389–398.

[37] Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. Y. Containment of conjunctive regular path queries with inverse. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR 2000)* (2000), pp. 176–185.

[38] Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. Y. Query processing using views for regular path queries with inverse. In *Proceedings of the Nineteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS 2000)* (2000), pp. 58–66.

[39] Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. Y. View-based query processing and constraint satisfaction. In *Proceedings of the Fifteenth IEEE Symposium on Logic in Computer Science (LICS 2000)* (2000), pp. 361–371.

[40] Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. Y. What is query rewriting? In *Proceedings of the Seventh International Workshop on Knowledge Representation meets Databases (KRDB 2000)* (2000), CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-29/`, pp. 17–27.

[41] Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. Y. View-based query answering and query containment over semistructured data. In *Proc. of the Eighth International Workshop on Database Programming Languages (DBPL 2001)* (2001).

[42] Carey, M. J., Haas, L. M., Schwarz, P. M., Arya, M., Cody, W. F., Fagin, R., Flickner, M., Luniewski, A., Niblack, W., Petkovic, D., Thomas, J., Williams, J. H., and Wimmers, E. L. Towards heterogeneous multimedia information systems: The Garlic approach. In *Proc. of the 5th Int. Workshop on Research Issues in Data Engineering – Distributed Object Management (RIDE-DOM'95)* (1995), IEEE Computer Society Press, pp. 124–131.

[43] Casanova, M. A., Fagin, R., and Papadimitriou, C. H. Inclusion dependencies and their interaction with functional dependencies. *Journal of Computer and System Sciences 28*, 1 (1984), 29–59.

[44] Cattell, R. G. G., and Barry, D. K., Eds. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, Los Altos, 1997.

[45] Chan, E. P. F. Containment and minimization of positive conjunctive queries in OODB's. In *Proceedings of the Eleventh ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'92)* (1992), pp. 202–211.

[46] Chandra, A. K., and Merlin, P. M. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth ACM Symposium on Theory of Computing (STOC'77)* (1977), pp. 77–90.

[47] Chaudhuri, S., Krishnamurthy, S., Potarnianos, S., and Shim, K. Optimizing queries with materialized views. In *Proceedings of the Eleventh IEEE International Conference on Data Engineering (ICDE'95)* (1995).

[48] Chaudhuri, S., and Vardi, M. Y. On the equivalence of recursive and nonrecursive Datalog programs. In *Proceedings of the Eleventh ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'92)* (1992), pp. 55–66.

[49] Chawathe, S. S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J. D., and Widom, J. The TSIMMIS project: Integration of heterogeneous information sources. In *Proc. of the 10th Meeting of the Information Processing Society of Japan (IPSJ'94)* (1994), pp. 7–18.

[50] Chekuri, C., and Rajaraman, A. Conjunctive query containment revisited. In *Proceedings of the Sixth International Conference on Database Theory (ICDT'97)* (1997), pp. 56–70.

[51] Chomicki, J., and Marcinkowski, J. Minimal-change integrity maintenance using tuple deletions. Tech. Rep. `arXiv:cs.DB/0212004v1`, 2002.

[52] Chomicki, J., and Marcinkowski, J. On the computational complexity of consistent query answers. Tech. Rep. `arXiv:cs.DB/0204010v1`, 2002.

[53] Cohen, S., Nutt, W., and Serebrenik, A. Rewriting aggregate queries using views. In *Proceedings of the Eighteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'99)* (1999), pp. 155–166.

[54] Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. Complexity and expressive power of logic programming. *ACM Computing Surveys 33*, 3 (2001), 374–425.

[55] Devlin, B. *Data Warehouse: From Architecture to Implementation*. Addison Wesley Publ. Co., Reading, Massachussetts, 1997.

[56] Dong, G., and Su, J. Conjunctive query containment with respect to views and constraints. *Information Processing Letters 57*, 2 (1996), 95–102.

[57] Dung, P. M. Integrating data from possibly inconsistent databases. In *Proceedings of the Fourth International Conference on Cooperative Information Systems (CoopIS'96)* (1996), pp. 58–65.

[58] Duschka, O. *Query Planning and Optimization in Information Integration.* PhD thesis, Stanford University, 1997.

[59] Duschka, O., and Genesereth, M. R. Infomaster – an information integration tool. In *Proceedings of the International Workshop on Intelligent Information Integration during the 21st German Annual Conference on Artificial Intelligence (KI'97)* (1997).

[60] Duschka, O. M., and Genesereth, M. R. Answering recursive queries using views. In *Proceedings of the Sixteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'97)* (1997), pp. 109–116.

[61] Duschka, O. M., Genesereth, M. R., and Levy, A. Y. Recursive query plans for data integration. *Journal of Logic Programming 43*, 1 (2000), 49–73.

[62] Duschka, O. M., and Levy, A. Y. Recursive plans for information gathering. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)* (1997), pp. 778–784.

[63] Eiter, T., Fink, M., Greco, G., and Lembo, D. Efficient evaluation of logic programs for querying data integration systems. In *Proceedings of the Nineteenth International Conference on Logic Programming (ICLP 2003)* (2003). To appear.

[64] Eiter, T., and Gottlob, G. On the complexity of propositional knowledge base revision, updates and counterfactuals. *Artificial Intelligence 57* (1992), 227–270.

[65] Eiter, T., and Gottlob, G. The complexity of nested counterfactuals and iterated knowledge base revisions. *Journal of Computer and System Sciences 53*, 3 (1996), 497–512.

[66] Eiter, T., Gottlob, G., and Mannila, H. Disjunctive Datalog. *ACM Transactions on Database Systems 22*, 3 (1997), 364–417.

[67] Eiter, T., Gottlob, G., and Mannilla, H. Disjunctive Datalog. *ACM Transactions on Database Systems 22*, 3 (1997), 364–418.

[68] Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. The KR system dlv: Progress report, comparison and benchmarks. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)* (1998), pp. 636–647.

[69] Fagin, R., Kolaitis, P. G., Miller, R. J., and Popa, L. Data exchange: Semantics and query answering. In *Proceedings of the Ninth International Conference on Database Theory (ICDT 2003)* (2003), pp. 207–224.

[70] Fagin, R., Kolaitis, P. G., and Popa, L. Data exchange: Getting the core. In *Proceedings of the Twentysecond ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS 2003)* (2003).

[71] Fagin, R., Ullman, J. D., and Vardi, M. Y. On the semantics of updates in databases. In *Proceedings of the Second ACM SIGACT SIGMOD Symposium on Principles of Database Systems (PODS'83)* (1983), pp. 352–365.

[72] Fan, W., and Libkin, L. On xml integrity constraints in the presence of dtds. *Journal of the ACM 43*, 3 (2002), 368–406.

[73] Fan, W., and Siméon, J. Integrity constraints for xml. In *Proceedings of the Nineteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS 2000)* (2000).

[74] Fernandez, M. F., Florescu, D., Kang, J., Levy, A. Y., and Suciu, D. Catching the boat with Strudel: Experiences with a web-site management system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1998), pp. 414–425.

[75] Fernandez, M. F., Florescu, D., Levy, A., and Suciu, D. Verifying integrity constraints on web-sites. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)* (1999), pp. 614–619.

[76] Florescu, D., Levy, A., and Suciu, D. Query containment for conjunctive queries with regular expressions. In *Proceedings of the Seventeenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'98)* (1998), pp. 139–148.

[77] Florescu, D., Levy, A. Y., Manolescu, I., and Suciu, D. Query optimization in the presence of limited access patterns. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1999), pp. 311–322.

[78] Friedman, M., Levy, A., and Millstein, T. Navigational plans for data integration. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)* (1999), AAAI Press/The MIT Press, pp. 67–73.

[79] Galhardas, H., Florescu, D., Shasha, D., Simon, E., and Saita, C. Delarative data cleaning: language, model and algorithms. Tech. Rep. 4149, INRIA, 2001.

[80] Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J. D., Vassalos, V., and Widom, J. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems 8*, 2 (1997), 117–132.

[81] Garey, M. R., and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, Ca, 1979.

[82] Gelfond, M., and Lifschitz, V. The stable model semantics for logic programming. In *Proceedings of the Fifth Logic Programming Symposium* (1988), The MIT Press, pp. 1070–1080.

[83] Genereseth, M. R., Keller, A. M., and Duschka, O. M. Infomaster: An information integration system. In *ACM SIGMOD International Conference on Management of Data* (1997).

[84] Goh, C. H., Bressan, S., Madnick, S. E., and Siegel, M. D. Context interchange: New features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems 17*, 3 (1999), 270–293.

[85] Grahne, G., and Mendelzon, A. O. Tableau techniques for querying information sources through global schemas. In *Proceedings of the Seventh International Conference on Database Theory (ICDT'99)* (1999), vol. 1540 of *Lecture Notes in Computer Science*, Springer, pp. 332–347.

[86] Greco, G., Greco, S., and Zumpano, E. A logic programming approach to the integration, repairing and querying of inconsistent databases. In *Proceedings of the Seventeenth International Conference on Logic Programming (ICLP'01)* (2001), vol. 2237 of *Lecture Notes in Artificial Intelligence*, Springer, pp. 348–364.

[87] Grumbach, S., Rafanelli, M., and Tininini, L. Querying aggregate data. In *Proceedings of the Eighteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'99)* (1999), pp. 174–184.

[88] Gryz, J. An algorithm for query folding with functional dependencies. In *Proc. of the 7th Int. Symp. on Intelligent Information Systems* (1998), pp. 7–16.

[89] Gryz, J. Query folding with inclusion dependencies. In *Proceedings of the Fourteenth IEEE International Conference on Data Engineering (ICDE'98)* (1998), pp. 126–133.

[90] Gryz, J. Query rewriting using views in the presence of functional and inclusion dependencies. *Information Systems 24*, 7 (1999), 597–612.

[91] Gupta, H., Harinarayan, V., Rajaraman, A., and Ullman, J. D. Index selection for OLAP. In *Proceedings of the Thirteenth IEEE International Conference on Data Engineering (ICDE'97)* (1997), pp. 208–219.

[92] Halevy, A. Y. Answering queries using views: A survey. *Very Large Database Journal 10*, 4 (2001), 270–294.

[93] Hammer, J., Garcia-Molina, H., Widom, J., Labio, W., and Zhuge, Y. The Stanford data warehousing project. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 18*, 2 (1995), 41–48.

[94] Harinarayan, V., Rajaraman, A., and Ullman, J. D. Implemeting data cubes efficiently. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1996), pp. 205–216.

[95] Hernàndez, M. A., and Stolfo, S. J. The merge/purge problem for large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1995).

[96] Hull, R. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proceedings of the Sixteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'97)* (1997), pp. 51–61.

[97] Hull, R., and Zhou, G. A framework for supporting data integration using the materialized and virtual approaches. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1996), pp. 481–492.

[98] Imielinski, T., and Jr., W. L. Incomplete information in relational databases. *Journal of the ACM 31*, 4 (1984), 761–791.

[99] Inmon, W. H. *Building the Data Warehouse*, second ed. John Wiley & Sons, 1996.

[100] Jarke, M., Lenzerini, M., Vassiliou, Y., and Vassiliadis, P., Eds. *Fundamentals of Data Warehouses*. Springer, 1999.

[101] Johnson, D. S., and Klug, A. C. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences 28*, 1 (1984), 167–189.

[102] Keller, A. M., and Basu, J. A predicate-based caching scheme for client-server database architectures. *Very Large Database Journal 5*, 1 (1996), 35–47.

[103] Kirk, T., Levy, A. Y., Sagiv, Y., and Srivastava, D. The Information Manifold. In *Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments* (1995), pp. 85–91.

[104] Klug, A. C. On conjunctive queries containing inequalities. *Journal of the ACM 35*, 1 (1988), 146–160.

[105] Kolaitis, P. G., and Papadimitriou, C. H. Why not negation by fixpoint? *Journal of Computer and System Sciences 43*, 1 (1991), 125–144.

[106] Kwok, C. T., and Weld, D. Planning to gather information. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)* (1996), pp. 32–39.

[107] Lembo, D., Lenzerini, M., and Rosati, R. Source inconsistency and incompleteness in data integration. In *Proceedings of the Ninth International Workshop on Knowledge Representation meets Databases (KRDB 2002)* (2002), CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-54/`.

[108] Lenzerini, M. Data integration: A theoretical perspective. In *Proceedings of the Twentyfirst ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS 2002)* (2002), pp. 233–246.

[109] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Koch, C., Mateis, C., Perri, S., and Scarcello, F. The DLV System for Knowledge Representation and Reasoning. Tech. Rep. INFSYS RR-1843-02-14, Institut für Informationssysteme, TU Wien, A-1040 Vienna, Austria, October 2002.

[110] Levy, A., Fikes, R. E., and Sagiv, S. Speeding up inference using relevance reasoning: A formalism and algorithms. *Artificial Intelligence 97*, 1–2 (1997).

[111] Levy, A. Y. Logic-based techniques in data integration. In *Logic Based Artificial Intelligence*, J. Minker, Ed. Kluwer Academic Publisher, 2000.

[112] Levy, A. Y., Mendelzon, A. O., Sagiv, Y., and Srivastava, D. Answering queries using views. In *Proceedings of the Fourteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'95)* (1995), pp. 95–104.

[113] Levy, A. Y., Rajaraman, A., and Ordille, J. J. Query answering algorithms for information agents. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)* (1996), pp. 40–47.

[114] Levy, A. Y., Rajaraman, A., and Ordille, J. J. Querying heterogenous information sources using source descriptions. In *Proceedings of the Twentysecond International Conference on Very Large Data Bases (VLDB'96)* (1996).

[115] Levy, A. Y., Rajaraman, A., and Ullman, J. D. Answering queries using limited external query processors. In *Proceedings of the Fifteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'96)* (1996), pp. 227–237.

[116] Levy, A. Y., and Rousset, M.-C. CARIN: A representation language combining Horn rules and description logics. In *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI'96)* (1996), pp. 323–327.

[117] Levy, A. Y., Srivastava, D., and Kirk, T. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems 5* (1995), 121–143.

[118] Levy, A. Y., and Suciu, D. Deciding containment for queries with complex objects. In *Proceedings of the Sixteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'97)* (1997), pp. 20–31.

[119] Li, C., and Chang, E. Answering queries with useful bindings. *ACM Transactions on Database Systems 26*, 3 (2001), 313–343.

[120] Li, C., and Chang, E. On answering queries in the presence of limited access patterns. In *Proceedings of the Eighth International Conference on Database Theory (ICDT 2001)* (2001), pp. 219–233.

[121] Li, C., Yerneni, R., Vassalos, V., Garcia-Molina, H., Papakonstantinou, Y., Ullman, J. D., and Valiveti, M. Capability based mediation in TSIMMIS. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1998), pp. 564–566.

[122] Lifschitz, V., and Turner, H. Splitting a Logic Program. In *Proceedings ICLP-94* (Santa Margherita Ligure, Italy, June 1994), MIT-Press, pp. 23–38.

[123] Lin, J., and Mendelzon, A. O. Merging databases under constraints. *International Journal of Cooperative Information Systems 7*, 1 (1998), 55–76.

[124] Low, W. L., Lee, M. L., and Ling, T. W. A knowledge-based approach for duplicate elimination in data cleaning. *Information Systems, Special Issue on Data Extraction, Cleaning and Reconciliation 26*, 8 (December 2001).

[125] Manolescu, I., Florescu, D., and Kossmann, D. Answering XML queries on heterogeneous data sources. In *Proceedings of the Twentyseventh International Conference on Very Large Data Bases (VLDB 2001)* (2001), pp. 241–250.

[126] Millstein, T. D., Levy, A. Y., and Friedman, M. Query containment for data integration systems. In *Proceedings of the Nineteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS 2000)* (2000), pp. 67–75.

[127] Milo, T., and Suciu, D. Index structures for path expressions. In *Proceedings of the Seventh International Conference on Database Theory (ICDT'99)* (1999), vol. 1540 of *Lecture Notes in Computer Science*, Springer, pp. 277–295.

[128] Mitchell, J. C. The implication problem for functional and inclusion dependencies. *Information and Control 56* (1983), 154–173.

[129] Mitra, P. An algorithm for answering queries efficiently using views. Tech. rep., University of Southern California, Information Science Institute, Stanford (CA, USA), 1999. Available at `http://dbpubs.stanford.edu/pub/1999-46`.

[130] Monge, A. E., and Elkan, C. P. The field matching problem: algorithms and applications. In *Int. Conf. on Practical Applications of Prolog (PAP'97)* (1996).

[131] Monge, A. E., and Elkan, C. P. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proc. of the ACM-SIGMOD workshop on research issues on knowledge discovery and data mining* (1997).

[132] Monge, A. E., and Elkan, C. P. Matching algorithms within a duplicate detection system. *IEEE Data Engineering Bulletin 23*, 4 (2000), 14–20.

[133] Niemelä, I., Simons, P., and Syrjänen, T. Smodels: A System for Answer Set Programming. In *Proceedings of the Eightieth International Workshop on Non-Monotonic Reasoning (NMR 2002)* (Breckenridge, Colorado, USA, April 2000), C. Baral and M. Truszczyński, Eds.

[134] Palopoli, L., Terracina, G., and Ursino, D. The system DIKE: Towards the semi-automatic synthesis of cooperative information systems and data warehouses. In *Proc. of Symposium on Advances in Databases and Information Systems (ADBIS-DASFAA 2000)* (2000), pp. 108–117.

[135] Palopoli, L., Terracina, G., and Ursino, D. A graph-based approach for extracting terminological properties of elements of XML documents. In *Proceedings of the Seventeenth IEEE International Conference on Data Engineering (ICDE 2001)* (2001), IEEE Computer Society Press, pp. 330–340.

[136] Papadimitriou, C. H. *Computational Complexity.* Addison Wesley Publ. Co., Reading, Massachussetts, 1994.

[137] Papakonstantinou, Y., Garcia-Molina, H., and Ullman, J. D. MedMaker: A mediation system based on declarative specifications. In *Proceedings of the Twelfth IEEE International Conference on Data Engineering (ICDE'96)* (1996), S. Y. W. Su, Ed., pp. 132–141.

[138] Papakonstantinou, Y., Garcia-Molina, H., and Widom, J. Object exchange across heterogeneous information sources. In *Proceedings of the Eleventh IEEE International Conference on Data Engineering (ICDE'95)* (1995), pp. 251–260.

[139] Papakonstantinou, Y., and Vassalos, V. Query rewriting using semistructured views. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1999).

[140] Pottinger, R., and Levy, A. Y. A scalable algorithm for answering queries using views. In *Proceedings of the Twentysixth International Conference on Very Large Data Bases (VLDB 2000)* (2000), pp. 484–495.

[141] Qian, X. Query folding. In *Proceedings of the Twelfth IEEE International Conference on Data Engineering (ICDE'96)* (1996), pp. 48–55.

[142] Rahm, E., and Do, H. H. Data cleaning: problems and current approaches. *IEEE Data Engineering Bulletin 23*, 4 (2000), 3–13.

[143] Rajaraman, A., Sagiv, Y., and Ullman, J. D. Answering queries using templates with binding patterns. In *Proceedings of the Fourteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'95)* (1995).

[144] Roychoudhury, A., Ramakrishnan, I. V., and Swift, T. A rule-based data standardizer for enterprise data bases. In *Int. Conf. on Practical Applications of Prolog (PAP'97)* (1997), pp. 271–289.

[145] Sagiv, Y., and Yannakakis, M. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM 27*, 4 (1980), 633–655.

[146] Savitch, W. J. Relationship between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences 4*, 2 (1970), 177–192.

[147] Terracina, G., and Ursino, D. Deriving synonymies and homonymies of object classes in semi-structured information sources. In *Proc. of International Conference on Management of Data (COMAD 2000)* (2000), McGraw-Hill, New York, pp. 21–32.

[148] Theodoratos, D., and Sellis, T. Data warehouse design. In *Proceedings of the Twentythird International Conference on Very Large Data Bases (VLDB'97)* (1997), pp. 126–135.

[149] Tsatalos, O. G., Solomon, M. H., and Ioannidis, Y. E. The GMAP: A versatile tool for phyisical data independence. *Very Large Database Journal 5*, 2 (1996), 101–118.

[150] Ullman, J. D. Information integration using logical views. *Theoretical Computer Science 239*, 2 (2000), 189–210.

[151] van der Meyden, R. *The Complexity of Querying Indefinite Information*. PhD thesis, Rutgers University, 1992.

[152] van der Meyden, R. Logical approaches to incomplete information. In *Logics for Databases and Information Systems*, J. Chomicki and G. Saake, Eds. Kluwer Academic Publisher, 1998, pp. 307–356.

[153] Vardi, M. Y. The complexity of relational query languages. In *Proceedings of the Fourteenth ACM SIGACT Symposium on Theory of Computing (STOC'82)* (1982), pp. 137–146.

[154] Widom (ed.), J. Special issue on materialized views and data warehousing. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 18*, 2 (1995).

[155] Wiederhold, G. Mediators in the architecture of future information systems. *IEEE Computer 25*, 3 (1992), 38–49.

[156] Wiener, J. L., Gupta, H., Labio, W. J., Zhuge, Y., Garcia-Molina, H., and Widom, J. A system prototype for warehouse view maintenance. Tech. rep., Stanford University, 1996. Available at `http://www-db-stanford.edu/warehousing/warehouse.html`.

[157] Yang, J., Karlapalem, K., and Li, Q. Algorithms for materialized view design in data warehousing environment. In *Proceedings of the Twentythird International Conference on Very Large Data Bases (VLDB'97)* (1997), pp. 136–145.

[158] Yerneni, R., Li, C., Garcia-Molina, H., and Ullman, J. D. Computing capabilities of mediators. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1999), pp. 443–454.

[159] Yerneni, R., Li, C., Ullman, J. D., and Garcia-Molina, H. Optimizing large join queries in mediation systems. In *Proceedings of the Seventh International Conference on Database Theory (ICDT'99)* (1999), pp. 348–364.

[160] Zhou, G., Hull, R., and King, R. Generating data integration mediators that use materializations. *Journal of Intelligent Information Systems 6* (1996), 199–221.

[161] Zhou, G., Hull, R., King, R., and Franchitti, J.-C. Data integration and warehousing using H20. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 18*, 2 (1995), 29–40.

[162] Zhou, G., Hull, R., King, R., and Franchitti, J.-C. Using object matching and materialization to integrate heterogeneous databases. In *Proceedings of the Third International Conference on Cooperative Information Systems (CoopIS'95)* (1995), pp. 4–18.