On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases

Andrea Calì Domenico Lembo Riccardo Rosati

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
{cali,lembo,rosati}@dis.uniroma1.it

ABSTRACT

In databases with integrity constraints, data may not satisfy the constraints. In this paper, we address the problem of obtaining consistent answers in such a setting, when key and inclusion dependencies are expressed on the database schema. We establish decidability and complexity results for query answering under different assumptions on data (soundness and/or completeness). In particular, after showing that the problem is in general undecidable, we identify the maximal class of inclusion dependencies under which query answering is decidable in the presence of key dependencies. Although obtained in a single database context, such results are directly applicable to data integration, where multiple information sources may provide data that are inconsistent with respect to the global view of the sources.

1. INTRODUCTION

In database applications, integrity constraints represent fundamental knowledge about the domain of interest [8, 1]. In many scenarios, data may not satisfy integrity constraints; this happens, for instance, in data integration [20, 17], where integrity constraints enrich the semantics of the global view of a set of autonomous information sources, while such constraints may be violated by data at the sources [14, 6]. In principle, the issue of dealing with integrity constraint violations is relevant in all applications involving the integration of heterogeneous information (e.g., Data Warehouses, Enterprise Resource Planning Systems, etc.). The current integration methodologies deal with this problem in a data reconciliation step, in which data are cleaned by *ad hoc* algorithms that eliminate all violations.

In the general case of a database in which data violate integrity constraints, the problem arises of how to interpret such a database. This problem has been extensively studied in several works in the area of *inconsistent databases* that have proposed a new semantic approach to the treatment of integrity constraints [15, 11, 5, 21, 3, 4, 19, 16], which we briefly illustrate in the following.

Traditionally, database theory adopts an *exact* interpretation of data, based on the *closed world assumption* [23], i.e., the interpretation of each relation r exactly corresponds to the extension of r in the database instance. In order to cope with data inconsistencies, other assumptions about data are adopted in the literature. In particular, the interpretation of each relation r can be considered either as a superset (*sound* semantics) or a subset (*complete* semantics) of the extension of r in the database instance. Although in many cases such assumptions are sufficient to guarantee the existence of a consistent interpretation of the data, in general a less strict interpretation is needed. In particular, several studies [15, 21, 3, 19] propose a *loose* semantics which selects, among all possible databases satisfying the integrity constraints, only the ones that are "as close as possible" to the actual database instance.

In this paper, we address the problem of query answering in a relational setting under the above semantics, when key and inclusion dependencies are expressed on the database schema. Specifically: *(i)* we identify the frontier between decidability and undecidability of query answering for the various semantics; *(ii)* for the decidable cases, we establish the computational complexity of the query answering problem.

A detailed summary of the results of this paper is presented in Section 6 (see Figure 1). We remark that the results we have obtained for the sound semantics extend previous studies on query containment under integrity constraints [18], while the results for the loose semantics extend known results in the field of inconsistent databases, 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.

The paper is organized as follows. In Section 2 we recall the formal framework of relational databases with integrity constraints. In Section 3 we study decidability and complexity of query answering under sound, complete, and exact semantics. In Section 4 we introduce a loose semantics for inconsistent data. In Section 5 we prove results about decidability and complexity of query answering under the loose semantics. Section 6 concludes the paper.

PODS 2003, June 9-12, 2003, San Diego, CA.

2. FRAMEWORK

In this section we present the syntax and semantics of the relational model with integrity constraints. We assume that the reader is familiar with the basic notions of relational databases [1].

2.1 Syntax

We consider to have an infinite, fixed alphabet Γ of values representing real world objects, and we take into account only database instances having Γ as domain. Moreover, we assume that different values in Γ denote different objects, i.e., we adopt the so-called *unique name assumption*.

Basically, in the relational model we have to account for a set of relation symbols and a set of integrity constraints, i.e., assertions on the relation symbols that express conditions that are intended to be satisfied by database instances.

In this paper we focus our attention on inclusion and key dependencies. More formally, we indicate a *relational schema* (or simply *schema*) $\mathcal{DB}$ as a triple $\langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$, where:

- $\mathcal{S}$ is a set of relations, each with an associated arity that indicates the number of its attributes. The attributes of a relation r of arity n are represented by the integers $1, \ldots, n$.
- $\mathcal{I}$ is a set of *inclusion dependencies* (IDs), i.e., a set of assertions of the form $r_1[\mathbf{A}] \subseteq r_2[\mathbf{B}]$, where r_1, r_2 are relations in $\mathcal{S}$, $\mathbf{A} = A_1, \ldots, A_n$ is a sequence of distinct attributes of r_1, and $\mathbf{B} = B_1, \ldots, B_n$ is a sequence of distinct attributes of r_2.
- $\mathcal{K}$ is a set of *key dependencies* (KDs), i.e., a set of assertions the form $key(r) = \mathbf{A}$, where r is a relation in the global schema, and $\mathbf{A} = A_1, \ldots, A_n$ is a sequence of attributes of r. We assume that at most one key dependency is specified for each relation.

A *relational query* (or simply *query*) over $\mathcal{DB}$ is a formula that is intended to extract a set of tuples of values of Γ. The language used to express queries over $\mathcal{DB}$ is *union of conjunctive queries* (UCQ). A UCQ q of arity n is written in the form $q(\vec{\mathbf{x}}) \leftarrow conj_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1) \vee \cdots \vee conj_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m)$, where for each $i \in \{1, \ldots, m\}$ $conj_i(\vec{\mathbf{x}}, \vec{\mathbf{y}}_i)$ is a conjunction of atoms whose predicate symbols are in $\mathcal{S}$, and involve $\vec{\mathbf{x}} = X_1, \ldots, X_n$ and $\vec{\mathbf{y}}_i = Y_{i,1}, \ldots, Y_{i,n_i}$, where X_k and Y_i, ℓ are either variables or values of Γ.

2.2 Semantics

A *database instance* (or simply *database*) $\mathcal{B}$ for a schema $\mathcal{DB}$ is a set of facts of the form $r(t)$ where r is a relation of arity n in $\mathcal{S}$ and t is an n-tuple of values from Γ. We denote as $r^{\mathcal{B}}$ the set $\{t \mid r(t) \in \mathcal{B}\}$. A database $\mathcal{B}$ for a schema $\mathcal{DB}$ is said to be *consistent* with $\mathcal{DB}$ if it satisfies all the dependencies expressed on $\mathcal{DB}$. In our framework, this means satisfying IDs in $\mathcal{I}$ and KDs in $\mathcal{K}$. More formally:

- $\mathcal{B}$ satisfies an inclusion dependency $r_1[\mathbf{A}] \subseteq r_2[\mathbf{B}]$ if for each tuple t_1 in $r_1^{\mathcal{B}}$ there exists a tuple t_2 in $r_2^{\mathcal{B}}$ such that $t_1[\mathbf{A}] = t_2[\mathbf{B}]$, where $t[\mathbf{A}]$ is the projection of the tuple t over $\mathbf{A}$. If $\mathcal{B}$ satisfies all inclusion dependencies expressed on $\mathcal{DB}$, then we say that $\mathcal{B}$ is consistent with $\mathcal{I}$;
- $\mathcal{B}$ satisfies a key dependency $key(r) = \mathbf{A}$ if for each $t_1, t_2 \in r^{\mathcal{B}}$ with $t_1 \neq t_2$ we have $t_1[\mathbf{A}] \neq t_2[\mathbf{A}]$. If $\mathcal{B}$ satisfies all key dependencies expressed on $\mathcal{DB}$ we say that $\mathcal{B}$ is consistent with $\mathcal{K}$.

Traditionally, the database theory essentially specifies a single database instance for a schema $\mathcal{DB}$. This means assuming that each relation r in S has to be considered *exact*, i.e., given a database instance $\mathcal{D}$ consistent with $\mathcal{DB}$, r is satisfied by exactly the tuples that satisfy r in $\mathcal{D}$.

On the other hand, different assumptions can be adopted for interpreting the tuples that $\mathcal{D}$ assigns to relations in $\mathcal{S}$ with respect to tuples that actually satisfy $\mathcal{DB}$. In particular, tuples in $\mathcal{D}$ can be considered a subset or a superset of the tuples that satisfy $\mathcal{DB}$, or exactly the set of tuples satisfying $\mathcal{DB}$. These interpretations give raise to three different semantics, called *sound*, *complete*, and *exact*, respectively.

Formally, given a database instance $\mathcal{D}$ for $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$, and an assumption x for $\mathcal{D}$, where $x \in \{s, c, e\}$ (for sound, complete, and exact semantics, respectively), the semantics of $\mathcal{DB}$ with respect to $\mathcal{D}$ and x, denoted $sem_x(\mathcal{DB}, \mathcal{D})$, is the set of database instances $\mathcal{B}$ for $\mathcal{DB}$ such that:

- $\mathcal{B}$ is consistent with $\mathcal{DB}$, i.e., it satisfies the integrity constraints in $\mathcal{I}$ and $\mathcal{K}$;
- $\mathcal{B}$ satisfies the assumptions specified on $\mathcal{D}$, i.e.:
  - $\mathcal{B} \supseteq \mathcal{D}$ when $x = s$ (sound semantics);
  - $\mathcal{B} \subseteq \mathcal{D}$ when $x = c$ (complete semantics);
  - $\mathcal{B} = \mathcal{D}$ when $x = e$ (exact semantics).

It is easy to see that, while $sem_e(\mathcal{DB}, \mathcal{D})$ contains at most a single database instance for $\mathcal{DB}$, in general $sem_s(\mathcal{DB}, \mathcal{D})$ and $sem_c(\mathcal{DB}, \mathcal{D})$ contain several databases for $\mathcal{DB}$. Furthermore, in our setting it always holds that $sem_c(\mathcal{DB}, \mathcal{D})$ is a non-empty set for each $\mathcal{DB}$ and each $\mathcal{D}$, since the empty database instance satisfies every possible set of KDs and IDs, therefore $\emptyset \in sem_c(\mathcal{DB}, \mathcal{D})$.

Finally, we give the semantics of queries. Formally, given a database instance $\mathcal{D}$ for $\mathcal{DB}$ and an assumption x for $\mathcal{D}$, where $x \in \{s, c, e\}$, we call *answers* to a query q of arity n with respect to $\mathcal{DB}$, $\mathcal{D}$ and x, the set $ans_x(q, \mathcal{DB}, \mathcal{D}) = \{\langle c_1, \ldots, c_n \rangle \mid \text{for each } \mathcal{B} \in sem_x(\mathcal{DB}, \mathcal{D}), \langle c_1, \ldots, c_n \rangle \in q^{\mathcal{B}}\}$, where $q^{\mathcal{B}}$ denotes the result of evaluating q over the database $\mathcal{B}$. We recall that $q^{\mathcal{B}}$ is the set of n-tuples of values of Γ $\langle c_1, \ldots, c_n \rangle$, such that, when substituting each c_i for x_i, the formula $\exists \vec{\mathbf{y}}_1 . conj_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1) \vee \cdots \vee \exists \vec{\mathbf{y}}_m . conj_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m)$ evaluates to true in $\mathcal{B}$.

In this paper we address the decision problem associated to query answering, that is, given a database schema $\mathcal{DB}$, a database instance $\mathcal{D}$, a query q of arity n over $\mathcal{DB}$ and a n-tuple $\overline{t}$ of values of Γ, to establish whether $\overline{t} \in ans_x(q, \mathcal{DB}, \mathcal{D})$.

EXAMPLE 2.1. *Consider the database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ where $\mathcal{S}$ contains the two relations*[1] $\mathsf{player}(Pname, Pteam)$ *and* $\mathsf{team}(Tname, Tcity)$*, $\mathcal{I}$ contains the ID* $\mathsf{player}[Pteam] \subseteq \mathsf{team}[Tname]$*, stating that every player is enrolled in a team of a city, and $\mathcal{K} = \emptyset$. Assume to have the database instance*

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

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

where a, b, c, d, e, f are values from Γ. It is easy to see that $sem_e(\mathcal{DB}, \mathcal{D}) = \emptyset$, since there do not exist two tuples in team *having d and f as first component, i.e., $\mathcal{D}$ is not consistent with $\mathcal{DB}$. This in turn implies that query answering is meaningless, since every possible fact is a logical consequence of $\mathcal{DB}$ and $\mathcal{D}$: for instance, the answer to the query that asks for all team names in* team*, i.e., $q(x) \leftarrow \mathsf{team}(x, y)$, is the whole interpretation domain Γ (that is, every possible value belongs to the extension of the query).*

On the other hand,

$$sem_c(\mathcal{DB}, \mathcal{D}) = \{\{\mathsf{player}(a, b), \mathsf{team}(b, c)\}, \{\mathsf{team}(b, c)\}, \emptyset\}$$

while $sem_s(\mathcal{DB}, \mathcal{D})$ contains all databases instance that can be obtained by adding to $\mathcal{D}$ (among others) at least one fact of the form $\mathsf{team}(d, \alpha)$ and one fact of the form $\mathsf{team}(f, \beta)$, where α and β are values of the domain Γ. Notice that, since $\emptyset \in sem_c(\mathcal{DB}, \mathcal{D})$, $ans_c(q, \mathcal{DB}, \mathcal{D}) = \emptyset$, i.e., there is no answer to the query in the complete semantics, whereas $ans_s(q, \mathcal{DB}, \mathcal{D}) = \{b, d, f\}$. ∎

2.3 Complexity classes

Finally, we briefly recall the complexity classes mentioned in the paper, and refer to [22] for further details. 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. In particular, the complexity class 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 is the class of problems that can be solved by a Turing machine that uses a polynomially bounded amount of memory.

3. QUERY ANSWERING

In this section we address the problem of query answering in the presence of integrity constraints, under different assumptions on the data. We consider a database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$, and a database instance $\mathcal{D}$ for $\mathcal{DB}$.

As illustrated in Section 2, when the data are considered complete, then the empty database always belongs to $sem_c(\mathcal{DB}, \mathcal{D})$, independently of $\mathcal{I}$ and $\mathcal{K}$; therefore, for any query q and for any tuple $\overline{t}$ we have immediately $ans_c(q, \mathcal{DB}, \mathcal{D}) = \emptyset$; hence, the answer to the decision problem is always negative. When the data are considered exact, we have two cases:

1. $\mathcal{D}$ satisfies both $\mathcal{I}$ and $\mathcal{K}$, therefore $sem_c(\mathcal{DB}, \mathcal{D}) = \{\mathcal{D}\}$ and $ans_e(q, \mathcal{DB}, \mathcal{D}) = q^{\mathcal{D}}$. So, it is immediate to establish whether $\overline{t} \in ans_e(q, \mathcal{DB}, \mathcal{D})$;

2. $\mathcal{D}$ violates either $\mathcal{I}$ or $\mathcal{K}$, therefore $sem_e(\mathcal{DB}, \mathcal{D}) = \emptyset$ and $ans_e(q, \mathcal{DB}, \mathcal{D})$ consists of all tuples of the same arity as q; the answer to the decision problem is therefore affirmative, independently of q and $\overline{t}$.

The case where the data are considered sound is more interesting: in fact, if the inclusion dependencies in $\mathcal{I}$ 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 $\mathcal{D}$ satisfies $\mathcal{K}$, the semantics of $\mathcal{DB}$ 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.

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

Definition 3.1. *Given a database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$, an inclusion dependency in $\mathcal{I}$ 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 $\mathcal{K}$ if either: (i) no KD is defined on r_2, or (ii) the KD $key(r_2) = \mathbf{K}$ is in $\mathcal{K}$, and $\mathbf{A}_2$ is not a strict superset of $\mathbf{K}$, i.e., $\mathbf{A}_2 \not\supset \mathbf{K}$. Moreover, the schema $\mathcal{DB}$ is* non-key-conflicting *(NKC) if all the IDs in $\mathcal{I}$ are NKCIDs with respect to $\mathcal{K}$.*

Informally, a set of dependencies is NKC if no ID in $\mathcal{I}$ propagates a proper subset of the key of the relation in its right-hand side. We point out that the class of NKC IDs 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 $\mathcal{K}$, $\mathbf{A}_2$ 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 $\mathcal{K}$. A *1-key-conflicting* (1KC) database schema is defined analogously to a NKC schema. We first show undecidability of implication of KDs and 1KCIDs.

Theorem 3.2. *The problem of implication[2] for KDs and 1KCIDs is undecidable.*

Proof. The proof is by reduction from the more general problem of implication of functional dependencies (FDs) and inclusion dependencies. Consider a generic instance of this problem, i.e., given a database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{F} \rangle$, where $\mathcal{I}$ is a set of IDs and $\mathcal{F}$ is a set of FDs, and an inclusion dependency δ. We assume that all FDs in $\mathcal{F}$ 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 database schema $\mathcal{DB}_1 = \langle \mathcal{S}_1, \mathcal{I}_1, \mathcal{K}_1 \rangle$, where $\mathcal{I}_1$ is a set of 1KCID with respect to $\mathcal{K}_1$, and the same dependency δ. We will show that the two problems are equivalent, i.e. $(\mathcal{I} \cup \mathcal{F}) \models \delta$ if and only if $(\mathcal{I}_1 \cup \mathcal{K}_1) \models \delta$. The dependencies $\mathcal{I}_1$ and $\mathcal{K}_1$, defined in a new database schema $\mathcal{DB}_1 = \langle \mathcal{S}_1, \mathcal{I}_1, \mathcal{K}_1 \rangle$, are constructed as follows.

- The new set of relations $\mathcal{S}_1$ includes all relations in $\mathcal{S}$ (plus those added as below).
- $\mathcal{I}_1$ includes all IDs in I (plus those added as below).
- Let φ be a FD in $\mathcal{F}$, of the form

  $$r : \mathbf{A} \rightarrow B$$

  We add to the schema an auxiliary relation r_φ of arity $|\mathbf{A}| + 1$, and we add to $\mathcal{I}_1$ the dependencies

  $$\begin{array}{lrcl} \gamma_1 : & r_\varphi[\mathbf{A}, B] & \subseteq & r[\mathbf{A}, B] \\ \gamma_2 : & r[\mathbf{A}, B] & \subseteq & r_\varphi[\mathbf{A}, B] \end{array}$$

[2]For the details about implication of database dependencies, we refer the reader to [1].

plus the key dependency

$$\varkappa : \quad key(r_\varphi) = \mathbf{A}$$

Note that all the IDs in $\mathcal{I}_2$ are 1KCIDs with respect to $\mathcal{K}_1$. The following result, whose proof is straightforward, will be used in the rest of the proof.

LEMMA 3.3. *For any database $\mathcal{B}_1$ for $\mathcal{DB}_1$, we have that $\mathcal{B}_1$ satisfies $\{\varphi, \gamma_1, \gamma_2\}$ if and only if $\mathcal{B}_1$ satisfies $\{\varkappa, \gamma_1, \gamma_2\}$.*

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 $(\mathcal{I} \cup \mathcal{F}) \models \delta$ if and only if $(\mathcal{I}_1 \cup \mathcal{K}_1) \models \delta$.

"$\Rightarrow$" By contradiction, suppose $(\mathcal{I}_1 \cup \mathcal{K}_1) \not\models \delta$; then there exists a database $\mathcal{B}_1$ for $\mathcal{DB}_1$ such that $\mathcal{B}_1$ satisfies $(\mathcal{I}_1 \cup \mathcal{K}_1)$ and violates δ. Consider a database $\mathcal{B}$ for $\mathcal{DB}$ obtained from $\mathcal{B}_1$ by removing the facts associated with the relations of the form r_φ introduced in the reduction. By Lemma 3.3, $\mathcal{B}$ satisfies $(\mathcal{I} \cup \mathcal{F})$; moreover, $\mathcal{B}$ cannot satisfy δ because $\mathcal{B}$ coincides with $\mathcal{B}_1$ on the relations in $\mathcal{S}$.

"$\Leftarrow$" By contradiction, suppose $(\mathcal{I} \cup \mathcal{K}) \not\models \delta$; then there exists a database $\mathcal{B}$ for $\mathcal{DB}$ such that $\mathcal{B}$ satisfies $(\mathcal{I} \cup \mathcal{F})$ and violates δ. We construct a database $\mathcal{B}_1$ for $\mathcal{D}_1$ that coincides with $\mathcal{B}$ on the relations in $\mathcal{S}$, 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 3.3, $\mathcal{B}_1$ satisfies $(\mathcal{I}_1 \cup \mathcal{K}_1)$; moreover, it cannot satisfy δ because $\mathcal{B}_1$ coincides with $\mathcal{B}$ on the relations in $\mathcal{S}$.

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

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

THEOREM 3.4. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a 1KC database schema, $\mathcal{D}$ a database instance for $\mathcal{DB}$, q a query of arity n over $\mathcal{DB}$, and $\overline{t}$ a n-tuple of values of Γ. The problem of establishing whether $\overline{t} \in ans_s(q, \mathcal{DB}, \mathcal{D})$ 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 [18]. 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 $\mathcal{DB}$ 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., $\langle\rangle \in ans_s(q, \mathcal{DB}, \mathcal{D})$) if and only if the fact $s(c_1, \ldots, c_k, d_{k+1}, \ldots, d_m)$ is in all databases in $sem_s(\mathcal{DB}, \mathcal{D})$. It is immediate to see that this is true if and only if $(\mathcal{I} \cup \mathcal{K}) \models \delta$. 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 3.5. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a 1KC database schema, $\mathcal{D}$ a database instance for $\mathcal{DB}$ consistent with $\mathcal{K}$, q a query of arity n over $\mathcal{DB}$, and $\overline{t}$ a n-tuple of values of Γ. The problem of establishing whether $\overline{t} \in ans_s(q, \mathcal{DB}, \mathcal{D})$ is undecidable.*

PROOF. The case where $\mathcal{D}$ does not satisfy $\mathcal{K}$ 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. To this aim, we need some preliminary results, presented in the milestone paper of Johnson and Klug [18], which addresses the problem of conjunctive query containment in a database $\mathcal{DB}$, in the presence of functional and inclusion dependencies. 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 $\mathcal{D}$, and then applying the *chase* procedure to such a database. The resulting (possibly infinite) database, denoted as $chase(\mathcal{DB}, \mathcal{D})$, 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 $r^{chase(\mathcal{DB},\mathcal{D})}$, and there is an ID $\delta \in \mathcal{I}$ of the form $r[\mathbf{X}_r] \subseteq s[\mathbf{X}_s]$. If there is no tuple t' in $s^{chase}(\mathcal{DB}, \mathcal{D})$ such that $t'[\mathbf{X}_s] = t[\mathbf{X}_r]$, then we add a new tuple t_{chase} in $s^{chase(\mathcal{DB},\mathcal{D})}$ such that $t_{chase}[\mathbf{X}_s] = t[\mathbf{X}_r]$, and for any attribute A_i of s, with $1 \leq i \leq m$ and $A_i \notin \mathbf{X}_s$, $t_{chase}[A_i]$ is a fresh value, not appearing elsewhere in the database.

Johnson and Klug have proved that $q_1 \subseteq q_2$ if and only if $q_2^{chase(\mathcal{DB},\mathcal{D})}$ is non-empty. Moreover, they have shown that, to check whether $q_2^{chase(\mathcal{DB},\mathcal{D})}$ is non-empty, only a finite portion of $chase(\mathcal{DB}, \mathcal{D})$ needs to be considered. Based on this property, they have defined a PSPACE algorithm Answer_{JK}, that checks the non-emptiness of $q_2^{chase(\mathcal{DB},\mathcal{D})}$.

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.

LEMMA 3.6. *Consider a database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \emptyset \rangle$ and an instance $\mathcal{D}$ for $\mathcal{DB}$; let q be a conjunctive query of arity n, and $\overline{t}$ a tuple of the same arity. We have that $\overline{t} \in ans_s(q, \mathcal{DB}, \mathcal{D})$ if and only if $\overline{t} \in q^{chase(\mathcal{DB},\mathcal{D})}$.*

PROOF (SKETCH).

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

"$\Leftarrow$" Analogously to [7], it can be proved by induction on the structure of $chase(\mathcal{DB}, \mathcal{D})$ that, for any database instance $\mathcal{B} \in sem_s(\mathcal{DB}, \mathcal{D})$, there exists a homomorphism μ that sends the tuples of $chase(\mathcal{DB}, \mathcal{D})$ to the tuples of $\mathcal{B}$.

By hypothesis $\overline{t} \in q^{chase(\mathcal{DB},\mathcal{D})}$, so there exists a homomorphism λ from the atoms of q to the facts of $chase(\mathcal{DB},\mathcal{D})$; the composition $\lambda \circ \mu$ witnesses that $\overline{t} \in ans_s(q,\mathcal{DB},\mathcal{D})$. □

Based on the above property, we can apply the algorithm Answer_{JK} for query answering, to check whether a tuple $\overline{t}$ belongs to $ans_s(q,\mathcal{DB},\mathcal{D})$.

We now go back to NKCIDs. 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. This property is expressed by the following result.

THEOREM 3.7 (SEPARATION). *Let $\mathcal{DB} = \langle \mathcal{S},\mathcal{I},\mathcal{K}\rangle$ be a NKC database schema, and let $\mathcal{DB}_1 = \langle \mathcal{S},\mathcal{I},\emptyset\rangle$ be the database schema obtained from $\mathcal{DB}$ by removing the KDs. Let $\mathcal{D}$ be a database instance for $\mathcal{DB}$ and $\mathcal{DB}_1$, q a query of arity n over $\mathcal{DB}$, and $\overline{t}$ a n-tuple of values of Γ. We have that $\overline{t} \notin ans_s(q,\mathcal{DB},\mathcal{D})$ iff $\mathcal{D}$ is consistent with $\mathcal{K}$ and $\overline{t} \notin ans_s(q,\mathcal{DB}_1,\mathcal{D})$.*

PROOF. "⇒" By hypothesis $\overline{t} \notin ans_s(q,\mathcal{DB},\mathcal{D})$; this means that there exists a database instance $\mathcal{B}$ for $\mathcal{DB}$ that satisfies $\mathcal{I}$ and $\mathcal{K}$, and such that $\overline{t} \notin q^{\mathcal{B}}$. It is immediate to verify that $\mathcal{B}$ is also an instance for $\mathcal{DB}_1$ that satisfies $\mathcal{K}$, and therefore $\overline{t} \notin ans_s(q,\mathcal{DB}_1,\mathcal{D})$. This proves the claim.

"⇐" By hypothesis $\overline{t} \notin ans_s(q,\mathcal{DB}_1,\mathcal{D})$ and $\mathcal{D}$ satisfies $\mathcal{K}$. Before we proceed further, we need to prove the following result.

LEMMA 3.8. *Let $\mathcal{DB} = \langle \mathcal{S},\mathcal{I},\mathcal{K}\rangle$ be a database schema, with KDs and NKCIDs, and $\mathcal{D}$ is an instance for $\mathcal{DB}$. Then $chase(\mathcal{DB},\mathcal{D})$ satisfies $\mathcal{I}$ and $\mathcal{K}$ if and only if $\mathcal{D}$ is consistent with $\mathcal{K}$.*

PROOF. "⇒" If $\mathcal{D}$ violates any of the key dependencies, since facts are only added (and never removed) in the construction of the canonical database $chase(\mathcal{DB},\mathcal{D})$, then also $chase(\mathcal{DB},\mathcal{D})$ violates the key dependencies in $\mathcal{DB}$.

"⇐" The proof is by induction on the structure of $chase(\mathcal{DB},\mathcal{D})$. First, by hypothesis $\mathcal{D}$ is consistent with $\mathcal{K}$. For the induction step, suppose we insert in $chase(\mathcal{DB},\mathcal{D})$ 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 $\overline{t}$ in $r^{chase^*(\mathcal{DB},\mathcal{D})}$, where $chase^*(\mathcal{DB},\mathcal{D})$ is the portion of $chase(\mathcal{DB},\mathcal{D})$ 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 $\overline{t}$ 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 values 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 $\overline{t}$ differ in the values in $\mathbf{K} - \mathbf{B}$, where t has fresh values, thus they differ *a fortiori* on $\mathbf{K}$.

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

□

With this result in place, we are able to extend the result of [18] to the case of NKCIDs and KDs. In fact, in this case $chase(\mathcal{DB}_1,\mathcal{D})$ (which is identical to $chase(\mathcal{DB},\mathcal{D})$ by construction) satisfies both $\mathcal{I}$ and $\mathcal{K}$. Therefore, it is also a representative of all databases in $sem_s(\mathcal{DB},\mathcal{D})$, since $sem_s(\mathcal{DB},\mathcal{D}) \subseteq sem_s(\mathcal{DB}_1,\mathcal{D})$: hence, from Lemma 3.6 it follows that $ans_s(q,\mathcal{DB},\mathcal{D}) = q^{chase(\mathcal{DB}_1,\mathcal{D})}$. The claim follows immediately. □

Based on the above theorem, we define the algorithm $\mathsf{Answer}_{\mathsf{S}}$, that solves query answering in the case of non-key-conflicting database schemata.

Algorithm $\mathsf{Answer}_{\mathsf{S}}(\mathcal{DB},\mathcal{D},q,t)$
Input: NKC database schema $\mathcal{DB} = \langle \mathcal{S},\mathcal{I},\mathcal{K}\rangle$,
database instance $\mathcal{D}$,
query q of arity n over $\mathcal{DB}$,
n-tuple t of values of Γ;
Output: *true* if $t \in ans_s(q,\mathcal{DB},\mathcal{D})$, *false* otherwise;
if $\mathcal{D}$ is not consistent with $\mathcal{K}$
then return *true*
else return $\mathsf{Answer}_{JK}(\mathcal{DB},\mathcal{D},q,t)$

To conclude the section, we present a complexity result for query answering in the presence of NKCIDs and KDs.

THEOREM 3.9. *Let $\mathcal{DB} = \langle \mathcal{S},\mathcal{I},\mathcal{K}\rangle$ be a NKC database schema, $\mathcal{D}$ a database instance for $\mathcal{DB}$, q a query of arity n over $\mathcal{DB}$, and $\overline{t}$ a n-tuple of values of Γ. The problem of establishing whether $\overline{t} \in ans_s(q,\mathcal{DB},\mathcal{D})$ is PSPACE-complete with respect to combined complexity. Moreover, it is in PTIME in data complexity.*

PROOF. From the results in [18], it follows directly that the problem in the case of IDs alone is PSPACE-complete; being such a case a particular case of NKCIDs and KDs (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 $\mathsf{Answer}_{\mathsf{S}}$ runs in PSPACE. Consider a database schema $\mathcal{DB} = \langle \mathcal{S},\mathcal{I},\mathcal{K}\rangle$ where $\mathcal{I}$ and $\mathcal{K}$ are sets of NKCIDs and KDs respectively. Given a database $\mathcal{D}$ for $\mathcal{DB}$, a query q of arity n over $\mathcal{G}$, and a n-tuple t of values of Γ, we want to establish whether $t \in ans_s(q,\mathcal{DB},\mathcal{D})$. Our algorithm $\mathsf{Answer}_{\mathsf{S}}$ proceeds as follows. The first step, clearly feasible in PTIME (and *a fortiori* in PSPACE), checks whether $\mathcal{D}$ satisfies $\mathcal{K}$. If it does not, the answer is trivial; if it does, we can apply Theorem 3.7, disregarding $\mathcal{K}$, and apply the PSPACE algorithm Answer_{JK} of [18]. All steps of $\mathsf{Answer}_{\mathsf{S}}$ are computable in PSPACE. Soundness and completeness of the algorithm follow immediately from Theorem 3.7 and from soundness and completeness of Answer_{JK} [18].

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

The above complexity characterization of the problem holds even if we restrict to instances consistent with the key dependencies.

COROLLARY 3.10. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a NKC database schema, $\mathcal{D}$ a database instance for $\mathcal{DB}$ consistent with $\mathcal{K}$, q a query of arity n over $\mathcal{DB}$, and $\bar{t}$ a n-tuple of values of Γ. The problem of establishing whether $\bar{t} \in ans_s(q, \mathcal{DB}, \mathcal{D})$ is PSPACE-complete.*

PROOF. Membership follows immediately from the general case treated in Theorem 3.9. With regard to hardness, observe that in the case where $\mathcal{D}$ does not satisfy $\mathcal{K}$ the above algorithm solves query answering in PTIME. The claim follows immediately. □

4. SEMANTICS FOR INCONSISTENT DATA

In the cases we have addressed so far, the violation of a single dependency (under the sound and exact semantics) may lead to the non-interesting case in which $sem_x(\mathcal{DB}, \mathcal{D}) = \emptyset$ ($x \in \{e, s\}$). This does not seem reasonable when the violations are due to a small set of facts. According to a common approach in the literature on inconsistent databases [15, 21, 3, 19], we now introduce less strict assumptions on data, under which we can get *consistent answers* from inconsistent database instances.

EXAMPLE 2.1 (CONTD.). As we have already shown, $sem_e(\mathcal{DB}, \mathcal{D}) = \emptyset$ since $\mathcal{D}$ does not satisfy $\mathcal{I}$, and as a consequence query processing is trivial in the exact semantics. Assume now to add the key dependency $key[\mathsf{player}] = \{Pname\}$ to $\mathcal{K}$, stating that a player cannot be enrolled in more than one team. It is easy to see that now it is also $sem_s(\mathcal{DB}, \mathcal{D}) = \emptyset$, since the facts $\mathsf{player}(a, b)$ and $\mathsf{player}(a, d)$ are not consistent with $\mathcal{K}$, and it is not possible to make $\mathcal{D}$ satisfy $\mathcal{K}$ by adding other facts to $\mathcal{D}$. On the other hand, $\mathsf{team}(b, c)$ is consistent with the dependencies in the schema, whereas the inconsistency caused by $\mathsf{player}(e, f)$ can be resolved under the sound semantics by adding a suitable fact to $\mathcal{D}$ of the form $\mathsf{player}(f, \alpha)$. Therefore, rather than the whole domain Γ, the query $q(x) \leftarrow \mathsf{team}(x, y)$ should return the answer set $\{b\}$ under the exact semantics and $\{b, f\}$ under the sound semantics. ■

A possible solution to this problem is to characterize the semantics of a database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ with respect to a database instance $\mathcal{D}$ in terms of those databases that *(i)* satisfy the integrity constraints on $\mathcal{DB}$, and *(ii)* approximate "at best" the satisfaction of the assumptions on $\mathcal{D}$. In other words, the integrity constraints of $\mathcal{DB}$ are considered "hard", whereas the assumptions are considered "soft".

According to the main approaches to inconsistent databases, we now propose a modified definition of the semantics that reflects the above idea. Given a possibly inconsistent database $\mathcal{D}$ for $\mathcal{DB}$, we define an ordering on the set of all databases consistent with $\mathcal{DB}$. If $\mathcal{B}_1$ and $\mathcal{B}_2$ are two such databases, we say that $\mathcal{B}_1$ is *better* than $\mathcal{B}_2$ with respect to $\mathcal{D}$, denoted as $\mathcal{B}_1 \gg_{\mathcal{D}} \mathcal{B}_2$, if:

- $\mathcal{B}_1 \cap \mathcal{D} \supset \mathcal{B}_2 \cap \mathcal{D}$ for the sound assumption
- $\mathcal{B}_1 - \mathcal{D} \subset \mathcal{B}_2 - \mathcal{D}$ for the complete assumption
- at least one of the two following conditions holds for the exact assumption:
  (i) $\mathcal{B}_1 \cap \mathcal{D} \supset \mathcal{B}_2 \cap \mathcal{D}$ and $\mathcal{B}_1 - \mathcal{D} \subseteq \mathcal{B}_2 - \mathcal{D}$;
  (ii) $\mathcal{B}_1 \cap \mathcal{D} \supseteq \mathcal{B}_2 \cap \mathcal{D}$ and $\mathcal{B}_1 - \mathcal{D} \subset \mathcal{B}_2 - \mathcal{D}$.

With this notion in place, we can modify the notion of semantics of a schema $\mathcal{DB}$ with respect to a database instance $\mathcal{D}$ and an assumption x, where $x \in \{s, c, e\}$ as usual. In order to distinguish between the semantics used so far and their modified version, in the following we refer to the former as *strict* semantics, while we call the latter *loose* semantics, and denote it with $sem_{l_x}(\mathcal{DB}, \mathcal{D})$. 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 loose semantics. More specifically, a database $\mathcal{B}$ consistent with $\mathcal{DB}$ is in $sem_{l_x}(\mathcal{DB}, \mathcal{D})$ if $\mathcal{B}$ is maximal with respect to $\gg_{\mathcal{D}}$, i.e., for no other database $\mathcal{B}'$ consistent with $\mathcal{DB}$, we have that $\mathcal{B}' \gg_{\mathcal{D}} \mathcal{B}$. It is also immediate to verify the following lemma:

LEMMA 4.1. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a database schema and $\mathcal{D}$ be a database instance for $\mathcal{DB}$. Then, $sem_{ls}(\mathcal{DB}, \mathcal{D}) = \bigcup_{\mathcal{D}'} sem_s(\mathcal{DB}, \mathcal{D}')$ for each $\mathcal{D}'$ maximal subset of $\mathcal{D}$ consistent with $\mathcal{K}$.*

PROOF. For each $\mathcal{B} \in sem_{ls}(\mathcal{DB}, \mathcal{D})$ consider $\mathcal{D}' = \mathcal{B} \cap \mathcal{D}$. It is easy to see that $\mathcal{D}'$ is a maximal subset of $\mathcal{D}$ consistent with $\mathcal{K}$, and that $\mathcal{B} \in sem_s(\mathcal{DB}, \mathcal{D}')$. Furthermore, for each $\mathcal{D}'$ maximal subset of $\mathcal{D}$ consistent with $\mathcal{K}$, if $\mathcal{B}' \in sem_s(\mathcal{DB}, \mathcal{D}')$, then $\mathcal{B}' \cap \mathcal{D}' = \mathcal{D}'$, hence $\mathcal{B}' \in sem_{ls}(\mathcal{DB}, \mathcal{D})$. □

With regard to answers, we indicate the set of answers to queries under the loose semantics with $ans_{l_x}(q, \mathcal{DB}, \mathcal{D})$, where $x \in \{s, c, e\}$ as usual. It is immediate to verify that, if $sem_x(\mathcal{DB}, \mathcal{D}) \neq \emptyset$ for any $x \in \{s, c, e\}$, then the strict semantics and the loose one coincide, in the sense that, for each query q $ans_x(q, \mathcal{DB}, \mathcal{D}) = ans_{l_x}(q, \mathcal{DB}, \mathcal{D})$. Consequently, since (as illustrated in Section 2) $sem_c(\mathcal{DB}, \mathcal{D}) \neq \emptyset$ for each $\mathcal{DB}$ 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 $sem_{l_x}(\mathcal{DB}, \mathcal{D}) \neq \emptyset$ for any $x \in \{s, c, e\}$, even if $sem_x(\mathcal{DB}, \mathcal{D}) = \emptyset$.

EXAMPLE 2.1 (CONTD.). With regard to our ongoing example we have that:

1. $sem_{le}(\mathcal{DB}, \mathcal{D})$ contains the database $\mathcal{B}_1 = \{\mathsf{player}(a, b), \mathsf{team}(b, c)\}$, and all the databases of the form $\mathcal{B}_2 = \{\mathsf{player}(a, d), \mathsf{team}(b, c), \mathsf{team}(d, \alpha)\}$ for each $\alpha \in \Gamma$, $\mathcal{B}_3 = \{\mathsf{player}(a, b), \mathsf{player}(e, f), \mathsf{team}(b, c), \mathsf{team}(f, \alpha)\}$ for each $\alpha \in \Gamma$, and $\mathcal{B}_4 = \{\mathsf{player}(a, d), \mathsf{player}(e, f), \mathsf{team}(b, c), \mathsf{team}(d, \alpha), \mathsf{team}(f, \beta)\}$ for each $\alpha, \beta \in \Gamma$;
2. $sem_{ls}(\mathcal{DB}, \mathcal{D})$ contains the databases of the form $\mathcal{B}_3$ and $\mathcal{B}_4$, and each database consistent with $\mathcal{DB}$ that can be obtained by adding facts to a database of the form $\mathcal{B}_3$ or $\mathcal{B}_4$;
3. $sem_{lc}(\mathcal{DB}, \mathcal{D}) = sem_c(\mathcal{DB}, \mathcal{D})$.

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

5. 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 4) the loosely-complete and the strictly-complete semantics coincide, we study query answering under the loosely-sound semantics and the loosely-exact semantics.

5.1 Query answering under the loosely-sound semantics

We now study the query answering problem under the loosely-sound semantics. As we already said, differently from the strictly-sound semantics, given a database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ and a database instance $\mathcal{D}$, it always holds that $sem_{ls}(\mathcal{DB}, \mathcal{D}) \neq \emptyset$, because we are now allowed to also eliminate facts from $\mathcal{D}$ in order to satisfy integrity constraints. Notice that, while to satisfy key dependencies we are forced to delete facts from $\mathcal{D}$, inclusion dependencies must be satisfied by adding new facts, since databases in $sem_{ls}(\mathcal{DB}, \mathcal{D})$ 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 database schemata.

THEOREM 5.1. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a 1KC database schema, $\mathcal{D}$ a database instance for $\mathcal{DB}$ consistent with $\mathcal{K}$, q a query of arity n over $\mathcal{DB}$, and $\overline{t}$ a n-tuple of values from Γ. The problem of establishing whether $\overline{t} \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ is undecidable.*

PROOF. Undecidability follows from Corollary 3.5 since, in the case in which $\mathcal{D}$ is consistent with $\mathcal{K}$, $\overline{t} \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ iff $\overline{t} \in ans_s(q, \mathcal{DB}, \mathcal{D})$. □

As for the class of non-key-conflicting database schemata, 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 $\mathcal{K}$, then for each such database $\mathcal{D}'$ we make use of the algorithm $\mathsf{Answer_S}$ presented in Section 3. Indeed, it can be shown that a tuple t is a consistent answer to a query q with respect to $\mathcal{DB}$ and $\mathcal{D}$, i.e., $t \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$, iff $\mathsf{Answer_S}(\mathcal{DB}, \mathcal{D}', q, t)$ returns true for each such database $\mathcal{D}'$. More specifically, we define the following algorithm:

Algorithm $\mathsf{Answer_{LS}}(\mathcal{DB}, \mathcal{D}, q, t)$
Input: non-key-conflicting database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$,
database instance $\mathcal{D}$,
query q of arity n over $\mathcal{DB}$, n-tuple t of values from Γ;
Output: *true* if $t \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$, *false* otherwise;
if there exists $\mathcal{D}_1 \subseteq \mathcal{D}$
such that
(1) $\mathcal{D}_1$ is consistent with $\mathcal{K}$;
(2) **for each** $r(\overline{t}) \in \mathcal{D} - \mathcal{D}_1$,
$\mathcal{D}_1 \cup \{r(\overline{t})\}$ is not consistent with $\mathcal{K}$;
(3) $\mathsf{Answer_S}(\mathcal{DB}, \mathcal{D}_1, q, t)$ 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 $\mathcal{K}$; this implies the existence of a database $\mathcal{B} \in sem_{ls}(\mathcal{DB}, \mathcal{D})$ such that $\mathcal{B} \cap \mathcal{D} = \mathcal{D}_1$. Then, condition (3) verifies that $\overline{t} \notin q^{\mathcal{B}}$.

THEOREM 5.2. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a NKC database schema, $\mathcal{D}$ be a database instance for $\mathcal{DB}$, q be a query of arity n over $\mathcal{DB}$, and $\overline{t}$ be a n-tuple of values of Γ. Then, $\overline{t} \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ iff $\mathsf{Answer_{LS}}(\mathcal{DB}, \mathcal{D}, q, \overline{t})$ returns true.*

PROOF. "⇒" If $\overline{t} \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ then $\overline{t} \in q^{\mathcal{B}}$ for each $\mathcal{B} \in sem_{ls}(\mathcal{DB}, \mathcal{D})$. From Lemma 4.1 it follows that $\overline{t} \in ans_s(q, \mathcal{DB}, \mathcal{D}_1)$ for each $\mathcal{D}_1$ maximal subset of $\mathcal{D}$ consistent with $\mathcal{K}$, and from soundness and completeness of algorithm $\mathsf{Answer_S}$, it follows that $\mathsf{Answer_S}(\mathcal{DB}, \mathcal{D}_1, q, \overline{t})$ returns *true* for each such database $\mathcal{D}_1$. Hence, $\mathsf{Answer_{LS}}(\mathcal{DB}, \mathcal{D}, q, \overline{t})$ returns *true*.

"⇐" Suppose by contradiction that $\overline{t} \notin ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ and $\mathsf{Answer_{LS}}(\mathcal{DB}, \mathcal{D}, q, \overline{t})$ returns *true*. This implies that for each $\mathcal{D}_1$ maximal subset of $\mathcal{D}$ consistent with $\mathcal{K}$, $\mathsf{Answer_S}(\mathcal{DB}, \mathcal{D}_1, q, \overline{t})$ returns *true*. From soundness and completeness of algorithm $\mathsf{Answer_S}$, it follows that $\overline{t} \in ans_s(q, \mathcal{DB}, \mathcal{D}_1)$ for each such database $\mathcal{D}_1$, i.e., $\overline{t} \in q^{\mathcal{B}}$ for each $\mathcal{B} \in sem_s(\mathcal{DB}, \mathcal{D}_1)$. From Lemma 4.1 it follows that $\overline{t} \in q^{\mathcal{B}}$ for each $\mathcal{B} \in sem_{ls}(\mathcal{DB}, \mathcal{D})$, 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 with respect to $\mathcal{K}$.

THEOREM 5.3. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a NKC database schema, $\mathcal{D}$ be a database instance for $\mathcal{DB}$, q be a query of arity n over $\mathcal{DB}$, and $\overline{t}$ be a n-tuple of values of Γ. The problem of establishing whether $\overline{t} \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ is coNP-complete with respect to data complexity.*

PROOF. Membership in coNP follows from the algorithm $\mathsf{Answer_{LS}}(\mathcal{DB}, \mathcal{D}, q, \overline{t})$ and from Theorem 3.9. Indeed, in the algorithm the problem of establishing whether $\overline{t} \notin ans_{ls}(q, \mathcal{DB}, \mathcal{D})$, 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.

We prove coNP-hardness of the problem even if we restrict to database schemata without IDs. Actually, this hardness result can be immediately derived from the results reported in [10] (although obtained under a different semantics): however, in the following we provide an alternative proof, in which we use a reduction of the 3-colorability problem to our problem. Consider a graph $G = (V, E)$ with a set of vertices V and edges E. We define a database schema $\mathcal{DB} = \langle \mathcal{S}, \emptyset, \mathcal{K} \rangle$ where $\mathcal{S}$ consists of the two binary relations *edge* and *col*, and $\mathcal{K}$ contains the dependency $key(col) = \{1\}$. The instance $\mathcal{D}$ is defined as follows:

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

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 $\langle\rangle \notin ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ (i.e., the boolean query q has an affirmative answer). 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 $\mathcal{B} \in sem_{ls}(\mathcal{DB}, \mathcal{D})$ 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 $\mathcal{B} \in sem_{ls}(\mathcal{DB}, \mathcal{D})$, consequently $\langle\rangle \notin ans_{ls}(q, \mathcal{DB}, \mathcal{D})$. Conversely, it is immediate to verify that, for each $\mathcal{B} \in sem_{ls}(\mathcal{DB}, \mathcal{D})$, $col^{\mathcal{B} \cap \mathcal{D}}$ 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 $\langle\rangle \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$. □

THEOREM 5.4. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a NKC database schema, $\mathcal{D}$ be a database instance for $\mathcal{DB}$, q be a query of arity n over $\mathcal{DB}$, and $\overline{t}$ be a n-tuple of values of Γ. The problem of establishing whether $\overline{t} \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ is PSPACE-complete with respect to combined complexity.*

PROOF. Hardness follows from Corollary 3.10 and from the fact that, when $\mathcal{D}$ is consistent with $\mathcal{K}$, $\overline{t} \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ if and only if $\overline{t} \in ans_s(q, \mathcal{DB}, \mathcal{D})$.

Membership in PSPACE follows from algorithm $\mathsf{Answer}_{\mathsf{LS}}(\mathcal{DB}, \mathcal{D}, q, t)$ and Theorem 3.9. Indeed, it is easy to see that conditions (1), (2), and (3) can be verified in polynomial space, and furthermore NPSPACE=PSPACE [22]. □

5.2 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 satisfied by either adding or deleting facts. Hence, $sem_{le}(\mathcal{DB}, \mathcal{D})$ 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 and KDs.

THEOREM 5.5. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a database schema, $\mathcal{D}$ a database instance for $\mathcal{DB}$, q a query of arity n over $\mathcal{DB}$, and $\overline{t}$ a n-tuple of values of Γ. The problem of establishing whether $\overline{t} \in ans_{le}(q, \mathcal{DB}, \mathcal{D})$ is undecidable.*

PROOF. We reduce query answering in the loosely-sound semantics to query answering in the loosely-exact semantics. We can restrict to instances $\mathcal{D}$ consistent with $\mathcal{K}$, since by Theorem 5.1 for this class of instances the problem of establishing whether $\overline{t} \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ is undecidable. Starting from such a problem instance $(\mathcal{DB}, \mathcal{D}, q, \overline{t})$, we define a new problem instance $(\mathcal{DB}', \mathcal{D}', q', \overline{t}')$ such that $\overline{t} \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ iff $\overline{t}' \in ans_{le}(q', \mathcal{DB}', \mathcal{D}')$. Precisely:

- $\mathcal{DB}' = \langle \mathcal{S}', \mathcal{I}', \mathcal{K}' \rangle$ is obtained from $\mathcal{DB}$ by:
  - defining $\mathcal{S}'$ as the schema obtained from $\mathcal{S}$ by adding an attribute to each relation in $\mathcal{S}$ (in the last position);
  - changing each inclusion in order to propagate such a new attribute from r to s, i.e., $\mathcal{I}'$ is obtained from $\mathcal{I}$ 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 $\mathcal{S}'$ and m is the arity of s in $\mathcal{S}'$;
- $\mathcal{D}'$ 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

$$\mathcal{D}'_2 = \{\, r(\overline{u}, t_1) \mid r \in \mathcal{S} \text{ and } \overline{u} \text{ is a tuple of values of } \Gamma_{\mathcal{D}} \cup \{t_1\} \,\}$$

  where $\Gamma_{\mathcal{D}}$ denotes the set of symbols from Γ appearing in $\mathcal{D}$, and t_0, t_1 are values not belonging to $\Gamma_{\mathcal{D}}$. 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

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

- $\overline{t}'$ is obtained from $\overline{t}$ by adding the value t_0 at the end of the tuple $\overline{t}$.

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

1. $\mathcal{B} \cap \mathcal{D} = \mathcal{D}$. In this case, due to the key dependencies $\mathcal{K}$, $\mathcal{B}$ 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 $\mathcal{K}$. Consequently, $\overline{t}' \in q'^{\mathcal{B}}$ iff $\overline{t} \in q^{\mathcal{B}}$. Moreover, it is immediate to verify that there exists at least one such $\mathcal{B}$ in $sem_{le}(\mathcal{DB}', \mathcal{D}')$;
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 $\mathcal{K}$, consequently $\overline{t}' \in q'^{\mathcal{B}}$ for each such $\mathcal{B}$. In other words, this kind of databases does not affect $ans_{le}(q', \mathcal{DB}', \mathcal{D}')$, since in $\mathcal{B}$ every possible tuple is in the answer of q'.

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

Finally, since the above reduction is effectively computable and since, by Theorem 5.1, establishing whether $\overline{t} \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ is undecidable, the thesis follows. □

Differently from the previous semantics, in the case when the instance $\mathcal{D}$ is consistent with $\mathcal{K}$, 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 5.6. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a database schema, $\mathcal{D}$ a database instance consistent with $\mathcal{K}$, q a query of arity n over $\mathcal{DB}$, and $\overline{t}$ be a n-tuple of values of Γ. The problem of establishing whether $\overline{t} \in ans_{le}(q, \mathcal{DB}, \mathcal{D})$ 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 $\mathsf{AnswerCons}_{\mathsf{LE}}(\mathcal{DB}, \mathcal{D}, q, \overline{t})$
Input: database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$,
instance $\mathcal{D}$ consistent with $\mathcal{K}$,
conjunctive query q of arity n, n-tuple $\overline{t}$
Output: *true* if $\overline{t} \in ans_{le}(q, \mathcal{DB}, \mathcal{D})$, *false* otherwise
$\mathcal{D}_1 = \mathcal{D}$;
repeat
  $\mathcal{D}_0 = \mathcal{D}_1$;
  for each $r(\overline{t}') \in \mathcal{D}_1$

if there exists $r[i_1,\ldots,i_k] \subseteq s[j_1,\ldots,j_k] \in \mathcal{I}$
such that
for each $s(\overline{t}'') \in \mathcal{D}_1$, $\overline{t}''[j_1,\ldots,j_k] \neq \overline{t}'[i_1,\ldots,i_k]$
then $\mathcal{D}_1 = \mathcal{D}_1 - \{r(\overline{t}')\}$
until $\mathcal{D}_1 = \mathcal{D}_0$;
if $\overline{t} \in q^{\mathcal{D}_1}$
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)* $\mathcal{D}_1 \in sem_{le}(\mathcal{DB},\mathcal{D})$; *(ii)* for each $\mathcal{B} \in sem_{le}(\mathcal{DB},\mathcal{D})$, $\mathcal{B} \supseteq \mathcal{D}_1$. Therefore, $\overline{t} \in ans_{le}(q,\mathcal{DB},\mathcal{D})$ if and only if $\overline{t} \in q^{\mathcal{D}_1}$. 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 database schemata. To this aim, we first define a particular query $Q(I,\overline{t})$ associated with a tuple $\overline{t}$ and an inclusion dependency I.

DEFINITION 5.7. *Let 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 $\overline{t}$ be an n-tuple. We denote as $Q(I,\overline{t})$ 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 $Q(\mathcal{I},\overline{t})$ is used in order to verify whether a database schema $\mathcal{DB}$ and an instance $\mathcal{D}$ imply the existence in all databases $\mathcal{B} \in sem_s(\mathcal{DB},\mathcal{D})$ of a fact of the form $s(\overline{t}')$ such that $\overline{t}[i_1,\ldots,i_k] = \overline{t}'[j_1,\ldots,j_k]$.

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

Algorithm $\mathsf{Answer_{LE}}(\mathcal{DB},\mathcal{D},q,\overline{t})$
Input: NKC database schema $\mathcal{DB} = \langle \mathcal{S},\mathcal{I},\mathcal{K}\rangle$, instance $\mathcal{D}$, query q of arity n over $\mathcal{DB}$, n-tuple $\overline{t}$ of values of Γ
Output: *true* if $\overline{t} \in ans_{le}(q,\mathcal{DB},\mathcal{D})$, *false* otherwise
if there exists $\mathcal{D}' \subseteq \mathcal{D}$ **such that**
(a) $\mathcal{D}'$ is consistent with $\mathcal{K}$ **and**
(b) $\mathsf{Answer_S}(\langle \mathcal{S},\mathcal{I},\emptyset\rangle,\mathcal{D}',q,\overline{t})$ 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 $\mathcal{K}$ **or**
(c2) **there exists** $I \in \mathcal{I}$ **and** $r(\overline{t}_1) \in \mathcal{D}''$
such that
$\mathsf{Answer_S}(\langle \mathcal{S},\mathcal{I},\emptyset\rangle,\mathcal{D}',Q(I,\overline{t}_1),\langle\rangle)$ returns *false* **and**
$\mathsf{Answer_S}(\langle \mathcal{S},\emptyset,\emptyset\rangle,\mathcal{D}'',Q(I,\overline{t}_1),\langle\rangle)$ returns *false*
then return *false*
else return *true*

Intuitively, to return *false* the algorithm looks for the existence of a database $\mathcal{B}'$ in $sem_{le}(\mathcal{DB},\mathcal{D})$ such that $\overline{t} \notin q^{\mathcal{B}'}$. 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 $\overline{t} \notin q^{\mathcal{B}'}$ is verified by condition (b), while the fact that $\mathcal{B}' \in sem_{le}(\mathcal{DB},\mathcal{D})$ 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 $\mathcal{B}'' \gg_{\mathcal{D}} \mathcal{B}'$. 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 5.8. *Let $\mathcal{DB} = \langle \mathcal{S},\mathcal{I},\mathcal{K}\rangle$ be a NKC database schema, $\mathcal{D}$ be a database instance, q be a query of arity n over $\mathcal{S}$, and $\overline{t}$ be a n-tuple of values from Γ. Then, $\overline{t} \in ans_{le}(q,\mathcal{DB},\mathcal{D})$ iff $\mathsf{Answer_{LE}}(\mathcal{DB},\mathcal{D},q,\overline{t})$ 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 database schema $\mathcal{DB} = \langle \mathcal{S},\mathcal{I},\emptyset\rangle$, we denote as $chase_1(\mathcal{DB},\mathcal{D})$ 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 $s(\overline{t}_2)$ such that there exist $I = r[i_1,\ldots,i_k] \subseteq s[j_1,\ldots,j_k] \in \mathcal{I}$ and $r(\overline{t}_1) \in \mathcal{D}$ such that $\overline{t}_1[i_1,\ldots,i_k] = \overline{t}_2[j_1,\ldots,j_k]$ and there exists no $s(\overline{t}_3) \in \mathcal{D}$ such that $\overline{t}_1[i_1,\ldots,i_k] = \overline{t}_3[j_1,\ldots,j_k]$.

LEMMA 5.9. *Let $\mathcal{D}',\mathcal{D}''$ be instances of a database schema $\mathcal{DB} = \langle \mathcal{S},\mathcal{I},\emptyset\rangle$ such that $\mathcal{D}' \subset \mathcal{D}''$ and, for each $I \in \mathcal{I}$ of the form $I = r[i_1,\ldots,i_k] \subseteq s[j_1,\ldots,j_k]$ and for each $r(\overline{t}_1) \in \mathcal{D}''$, either $\mathsf{Answer_S}(\langle \mathcal{S},\mathcal{I},\emptyset\rangle,\mathcal{D}',Q(I,\overline{t}_1),\langle\rangle)$ returns true or $\mathsf{Answer_S}(\langle \mathcal{S},\emptyset,\emptyset\rangle,\mathcal{D}'',Q(I,\overline{t}_1),\langle\rangle)$ returns true. Then, $chase(\mathcal{DB},\mathcal{D}'') - \mathcal{D}'' \subseteq chase(\mathcal{DB},\mathcal{D}') - \mathcal{D}'$.*

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

We now prove the theorem.

"⇒" Suppose $\mathsf{Answer_{LE}}(\mathcal{DB},\mathcal{D},q,\overline{t})$ 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 $B' = chase(\mathcal{DB},\mathcal{D}')$. Now, suppose $\mathcal{B}' \notin sem_{le}(\mathcal{DB},\mathcal{D})$: hence, there exists a database instance $\mathcal{B}''$ such that $\mathcal{B}''$ is consistent with $\mathcal{K}$ and $\mathcal{B}'' \gg_{\mathcal{D}} \mathcal{B}'$, 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 $r(\overline{t}_1) \in \mathcal{D}''$ and an inclusion $I = r[i_1,\ldots,i_k] \subseteq s[j_1,\ldots,j_k] \in \mathcal{I}$ such that:

1. $\mathsf{Answer_S}(\langle \mathcal{S},\mathcal{I},\emptyset\rangle,\mathcal{D}',Q(I,\overline{t}_1),\langle\rangle)$ returns *false*, which implies that there is no fact in $\mathcal{B}'$ of the form $s(\overline{t}_2)$ such that $\overline{t}_1[i_1,\ldots,i_k] = \overline{t}_2[j_1,\ldots,j_k]$;
2. $\mathsf{Answer_S}(\langle \mathcal{S},\emptyset,\emptyset\rangle,\mathcal{D}'',Q(I,\overline{t}_1),\langle\rangle)$ returns *false*, which implies that there is no fact in $\mathcal{D}''$ of the form $s(\overline{t}_2)$ such that $\overline{t}_1[i_1,\ldots,i_k] = \overline{t}_2[j_1,\ldots,j_k]$. On the other hand, a fact of the form $s(\overline{t}_2)$ such that $\overline{t}_1[i_1,\ldots,i_k] = \overline{t}_2[j_1,\ldots,j_k]$ must be present in $\mathcal{B}''$, due to the presence of $r(\overline{t}_1)$ in $\mathcal{D}''$ and to the inclusion I.

The two above conditions imply that there exists a fact of the form $s(\overline{t}_2)$ 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 $\mathcal{B}'' \gg_{\mathcal{D}} \mathcal{B}'$. Therefore, $\mathcal{B}' \in sem_{le}(\mathcal{DB},\mathcal{D})$, and since conditions (a) and (b) hold for $\mathcal{D}'$, it follows that $\overline{t} \notin q^{\mathcal{B}'}$, hence $\overline{t} \notin ans_{le}(q,\mathcal{DB},\mathcal{D})$.

"⇐" Suppose $\overline{t} \notin ans_{le}(q,\mathcal{DB},\mathcal{D})$. Therefore, there exists $\mathcal{B}' \in sem_{le}(\mathcal{DB},\mathcal{D})$ such that $\overline{t} \notin q^{\mathcal{B}'}$. Let $\mathcal{D}' = \mathcal{D} \cap \mathcal{B}'$. Since $\mathcal{B}' \in sem_{le}(\mathcal{DB},\mathcal{D})$, condition (a) of the algorithm holds for $\mathcal{B}'$, and since $\mathcal{D}' \subseteq \mathcal{B}'$, condition (a) holds for $\mathcal{D}'$ as well. From $\overline{t} \notin q^{\mathcal{B}'}$ 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 $\mathcal{K}$ and, for each $I \in \mathcal{I}$ of the form $I = r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k]$ and for each $r(\overline{t}_1) \in \mathcal{D}''$, either $\mathsf{Answers}(\langle \mathcal{S}, \mathcal{I}, \emptyset \rangle, \mathcal{D}', Q(I, \overline{t}_1), \langle\rangle)$ returns *true* or $\mathsf{Answers}(\langle \mathcal{S}, \emptyset, \emptyset \rangle, \mathcal{D}'', Q(I, \overline{t}_1), \langle\rangle)$ returns *true*. By Lemma 5.9, it follows that $chase(\mathcal{DB}, \mathcal{D}'') - \mathcal{D}'' \subseteq chase(\mathcal{DB}, \mathcal{D}') - \mathcal{D}'$. Now let $\mathcal{B}'' = chase(\mathcal{DB}, \mathcal{D}'')$: since $\mathcal{B}' \supseteq chase(\mathcal{DB}, \mathcal{D}')$, 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 $\mathcal{B}'' \gg_{\mathcal{D}} \mathcal{B}'$. Moreover, since $\mathcal{D}''$ is consistent with $\mathcal{K}$, $\mathcal{B}''$ is consistent with $\mathcal{K}$ and $\mathcal{I}$, consequently $\mathcal{B}' \notin sem_{le}(\mathcal{DB}, \mathcal{D})$, thus contradicting the hypothesis. Therefore, condition (c) holds for $\mathcal{D}'$, which implies that $\mathsf{Answer}_{\mathsf{LE}}(\mathcal{DB}, \mathcal{D}, q, \overline{t})$ returns *false*. □

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

THEOREM 5.10. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a NKC database schema, $\mathcal{D}$ a database instance, q a query of arity n over $\mathcal{DB}$, and $\overline{t}$ a n-tuple of values of Γ. The problem of establishing whether $\overline{t} \in ans_{le}(q, \mathcal{DB}, \mathcal{D})$ is Π_2^p-complete with respect to data complexity and PSPACE-complete with respect to combined complexity.*

Proof sketch. The analysis of the algorithm $\mathsf{Answer}_{\mathsf{LE}}$ shows that the problem is in Π_2^p with respect to 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 3;
- 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 Π_2^p with respect to data complexity. Hardness with respect to Π_2^p can be proved by a reduction from 2-QBF validity, i.e., the validity problem for quantified boolean formulae having the form $\forall \overline{x} \exists \overline{y} f(\overline{x}, \overline{y})$ where $f(\overline{x}, \overline{y})$ is a 3-CNF, i.e., a propositional formula in 3-conjunctive normal form. The reduction generalizes the scheme employed in the proof of Theorem 5.3.

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 with respect to 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 5.5, and observing that, if the original problem instance is such that, for each $I = r[\vec{\mathbf{A}}] \subseteq s[\vec{\mathbf{B}}] \in \mathcal{I}$, $\vec{\mathbf{B}}$ does not cover the set of all the attributes of s, then the derived database schema $\mathcal{DB}'$ 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 with respect to data complexity.

6. DISCUSSION

6.1 Summary of results

The summary of the results we have obtained is reported in Figure 1[3], 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. Each column (with the exception of the first two) corresponds to a different semantics, while each row corresponds to a different class of dependencies (specified in the first two columns). 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. We have marked with the symbol ♠ the cells corresponding either to already known results or to results straightforwardly implied by known results.

We point out that, due to the correspondence between query answering and query containment illustrated in Section 3, all the complexity results established for the problem of query answering also hold for the conjunctive query containment problem.

6.2 Related work

The problem of reasoning with inconsistent databases is closely related to the studies in *belief revision and update* [2]. 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 $\mathcal{I} \cup \mathcal{K}$ 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 $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ and the instance $\mathcal{D}$ corresponds to the belief revision problem $\mathcal{D} \circ (\mathcal{I} \cup \mathcal{K}) \models q(t)$. 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

[3]Due to space limitations, in the present version of the paper we have not been able to include the proofs of all the results reported in Figure 1.

Data complexity/combined complexity for *general* database instances:

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

Data complexity/combined complexity for *key-consistent* database instances:

KDs	IDs	strictly-sound	loosely-sound	loosely-exact
no	GEN	PTIME/PSPACE♠	PTIME/PSPACE	PTIME/NP
yes	no	PTIME/NP♠	PTIME/NP♠	PTIME/NP♠
yes	FK	PTIME/PSPACE	PTIME/PSPACE	PTIME/NP
yes	FK,UN	PTIME/PSPACE	PTIME/PSPACE	PTIME/NP
yes	NKCID	PTIME/PSPACE	PTIME/PSPACE	PTIME/NP
yes	1KCID	undecidable	undecidable	PTIME/NP
yes	GEN	undecidable♠	undecidable	PTIME/NP

Legenda: FK = foreign key dependencies, GEN = general IDs, UN = unary IDs; ♠ = already known result.

Figure 1: Complexity of query answering under KDs and IDs (decision problem)

results mostly concern a setting in which knowledge is specified in terms of propositional formulae of classical logic [12, 13]. 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 [5]. However, the approach in [5] is completely proof-theoretic, and no computational technique for obtaining consistent answers from inconsistent database is provided.

In [21] 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 [15] 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, which is similar to the loosely-sound semantics above presented.

In [3] 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 [4] 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 [3] and in [4] 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 [10, 9]. 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 [10] the authors establish complexity results for query answering under such a semantics in the presence of *denial constraints*, a generalization of key dependencies and functional dependencies, while in [9] also inclusion dependencies are considered. Such a "maximal complete" semantics is different from the complete semantics considered in the present paper.

Finally, [16] 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.

We point out that none of the above mentioned works provides a general solution for the case of cyclic inclusion dependencies under the semantics (both strict and loose) considered in this paper.

6.3 Future work

Although obtained in a single database context, many of the techniques and results presented here are directly applicable to data integration, where multiple information sources may provide data that are inconsistent with respect to the global view of the sources. Indeed, we believe that one important development of the research presented in this paper is towards both the computational analysis of query answering in data integration systems and the definition of effective query processing techniques in such a setting.

Moreover, we are currently working on the extension of the present framework with more complex forms of dependencies, e.g., functional dependencies and exclusion dependencies.

7. ACKNOWLEDGMENTS

This research has been supported by the Projects INFOMIX (IST-2001-33570) and SEWASIE (IST-2001-34825) funded by the EU, and by the Project D2I funded by MIUR (Ministero per l'Istruzione, l'Università e la Ricerca). We would like to thank Maurizio Lenzerini and Giuseppe De Giacomo for their insightful comments about this material.

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., Reading, Massachussetts, 1995.

[2] C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *J. of Symbolic Logic*, 50:510–530, 1985.

[3] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 68–79, 1999.

[4] M. Arenas, L. E. Bertossi, and J. Chomicki. Specifying and querying database repairs using logic programs with exceptions. In *Proc. of the 4th Int. Conf. on Flexible Query Answering Systems (FQAS 2000)*, pages 27–41. Springer, 2000.

[5] F. Bry. Query answering in information systems with integrity constraints. In *IFIP WG 11.5 Working Conf. on Integrity and Control in Information System*. Chapman & Hall, 1997.

[6] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Accessing data integration systems through conceptual schemas. In *Proc. of the 20th Int. Conf. on Conceptual Modeling (ER 2001)*, pages 270–284, 2001.

[7] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Data integration under integrity constraints. *Information Systems*, 2003. To appear.

[8] M. A. Casanova, R. Fagin, and C. H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. *J. of Computer and System Sciences*, 28(1):29–59, 1984.

[9] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. Technical Report `arXiv:cs.DB/0212004v1`, 2002.

[10] J. Chomicki and J. Marcinkowski. On the computational complexity of consistent query answers. Technical Report `arXiv:cs.DB/0204010v1`, 2002.

[11] P. M. Dung. Integrating data from possibly inconsistent databases. In *Proc. of the 4th Int. Conf. on Cooperative Information Systems (CoopIS'96)*, pages 58–65, 1996.

[12] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates and counterfactuals. *Artificial Intelligence*, 57:227–270, 1992.

[13] T. Eiter and G. Gottlob. The complexity of nested counterfactuals and iterated knowledge base revisions. *J. of Computer and System Sciences*, 53(3):497–512, 1996.

[14] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *Proc. of the 9th Int. Conf. on Database Theory (ICDT 2003)*, pages 207–224, 2003.

[15] R. Fagin, J. D. Ullman, and M. Y. Vardi. On the semantics of updates in databases. In *Proc. of the 2nd ACM SIGACT SIGMOD Symp. on Principles of Database Systems (PODS'83)*, pages 352–365, 1983.

[16] G. Greco, S. Greco, and E. Zumpano. A logic programming approach to the integration, repairing and querying of inconsistent databases. In *Proc. of the 17th Int. Conf. on Logic Programming (ICLP'01)*, volume 2237 of *Lecture Notes in Artificial Intelligence*, pages 348–364. Springer, 2001.

[17] A. Y. Halevy. Answering queries using views: A survey. *Very Large Database J.*, 10(4):270–294, 2001.

[18] D. S. Johnson and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. of Computer and System Sciences*, 28(1):167–189, 1984.

[19] D. Lembo, M. Lenzerini, and R. Rosati. Source inconsistency and incompleteness in data integration. In *Proc. of the 9th Int. Workshop on Knowledge Representation meets Databases (KRDB 2002)*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-54/`, 2002.

[20] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002)*, pages 233–246, 2002.

[21] J. Lin and A. O. Mendelzon. Merging databases under constraints. *Int. J. of Cooperative Information Systems*, 7(1):55–76, 1998.

[22] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley Publ. Co., Reading, Massachussetts, 1994.

[23] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 119–140. Plenum Publ. Co., New York, 1978.