Connecting Databases and Ontologies: A Data Quality Perspective

Horacio Tellez Perez and Jef Wijsen

Département d'Informatique, University of Mons, Belgium
{horacio.tellezperez, jef.wijsen}@umons.ac.be

Abstract. Taking a database-theoretic perspective on the problem of mapping relational databases to ontologies, we come up with a new mapping language that is inspired by the semijoin algebra. We illustrate the user friendliness of the mapping language by examples, and prove the decidability of some important reasoning problems by embedding our mapping language into the guarded fragment of first-order logic. We argue that these reasoning problems are relevant in data quality explorations.

Keywords: data quality · guarded fragment · ontology-based data access · OBDA · semijoin algebra

1 Motivation

The literature contains many proposals for mapping relational databases to ontologies. The major motivation for these proposals is probably *ontology-based data access* (OBDA) [20], i.e., the capability of interrogating databases by using an ontological vocabulary. The current study, however, started with a different purpose, which can be coined as *ontology-based database repairing* or *ontology-based database cleaning*. Database repairing and cleaning are approaches for dealing with dirty data, where dirtiness refers to the violation of integrity constraints or, more abstractly, the non-conformity to rules that the data should obey. Ideally, all such data rules should be declared at database design time and subsequently enforced by the database management system. In practice, however, we seldom dispose of an exhaustive declaration of all data rules: some rules were overlooked when the database schema was conceived, while others were hidden in procedural programming code. Moreover, in the course of time, new rules may emerge because of new legislation (e.g., GDPR), while existing rules may be invalidated. Now let us assume that we have access to an ontology that talks about objects and relations that also exist in some presumably dirty database. Our hypothesis is that data quality problems may become more visible when we succeed in connecting or mapping the database to the ontology, enabling us to confront the stored data with the ontological "ground truth." It should be mentioned here that an ontologically based approach to data quality is not a new idea: it already appeared in [19], was formalized in [10], and is mentioned in [20] as an important direction for future research.

In the problem of mapping relational databases to ontologies, we are given a relational database schema (i.e., a set of relation names), a description logic vocabulary (i.e., a set of unary and binary predicate names, called concept names and role names), and a TBox in some description logic. Moreover, we are given a database-to-ontology mapping, which defines a computable function from the set of database instances (over the fixed database schema) to the set of ABoxes in the description logic. If $\mathcal{M}$ denotes such a mapping and $\mathbf{db}$ denotes a database instance that serves as input to $\mathcal{M}$, then we write $\mathcal{M}(\mathbf{db})$ for the resulting ABox. In a data cleaning context, we may be interested to know, for example, whether the knowledge base $(\mathcal{T}, \mathcal{M}(\mathbf{db}))$ is consistent, and if not, what data in $\mathbf{db}$ causes inconsistency.

In this paper, we introduce and study a language for specifying such mappings $\mathcal{M}$, seeking a good balance between expressiveness and complexity. Four design considerations are as follows.

• First, we work in a perspective where columns in relations are not only numbered, as in mathematical logic, but also named with *attributes*. We will not assume that real-world entities have unique identifiers. Instead, we will use tuples with attributes to identify entities. This allows us, for example, to distinguish between the actress $\{\mathit{Lastname} : \texttt{Hilton}, \mathit{Firstname} : \texttt{Paris}\}$ and the entity $\{\mathit{Hotel} : \texttt{Hilton}, \mathit{City} : \texttt{Paris}\}$, which is a hotel in Paris.

• Second, the language for mapping databases to ontologies will be a subset of relational algebra. This leads to a succinct syntax without first-order variables. A major convenience for end-users is that any syntactically correct combination of the algebra operators is allowed in our mapping language. This would not be achievable in predicate logic, where end-users would be troubled with syntactic restrictions like safeness and guardedness.

• Third, like with description logics, a major consideration in the design of our mapping language is the balance between expressiveness and complexity. For expressiveness considerations, we allow negation in our mapping language, which is often considered useful [4]. On the other hand, the full expressive power of predicate logic would result in the undecidability of some basic reasoning problems.

• Fourth, relational database schemas are often obtained from a conceptual schema expressed in the Entity-Relationship model [9] or some variant of it. In such database schemas, most database tables are in 3NF and correspond to either an entity type or a relationship type in the conceptual schema. Intuitively, concept names and role names in description logics also correspond, respectively, to entity types and relationship types. Therefore, a plausible assumption is that in a well-designed database, the same real-world entity will generally not be spread out over multiple database tables, thus reducing the need for arbitrary joins in the mapping rules of $\mathcal{M}$. On the other hand, negation may be commonly needed (for example, to compute foreign students as all students except Belgian citizens).

The above considerations have brought us to the semijoin algebra [11], a fragment of the relational algebra which can be translated in GF, the guarded

fragment of first-order logic. In terms of expressiveness, our mapping language is incomparable with the commonly used language of GLAV mappings.

This paper is organized as follows. The next section discusses related work. Section 3 illustrates the concepts of this paper by means of a simple example. Section 4 introduces some preliminary definitions. Section 5 introduces *Entity-expressions* and *Relationship-expressions*, which are the building blocks for our mapping rules that are introduced in Section 6. The decidability of some important reasoning problems is established in Section 7. Section 8 concludes the paper.

2 Related Work

Starting with the seminal work by Poggi et al. [13], recent years have seen active research on disclosing relational databases to ontologies or the semantic Web [7, 14–16, 20]. The most commonly used rules used for mapping relational databases to ontologies have the form

$$\forall \boldsymbol{x} \left(\varphi(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y} \psi(\boldsymbol{x}, \boldsymbol{y}) \right), \tag{1}$$

where the *left-hand side* φ is a conjunction of atoms over the database schema, and the *right-hand side* ψ is a conjunction of atoms over the vocabulary (concept names and role names) of the ontology. A closed formula of the form (1) is called a *GLAV mapping* or, in the database literature, a *tuple-generating dependency (tgd)*. A tgd is *full* if no existential quantifier occurs in it. A *GAV tgd* is a full tgd whose right-hand side is a single atom. A *LAV tgd* is a tgd whose left-hand side is a single atom. Bienvenue [4] uses $\text{GAV}^{\neg,\neq}$ tgds, which extend GAV tgds by allowing negated atoms and inequalities in the left-hand side. In [13], the left-hand side is allowed to be an arbitrary SQL query. Most studies in OBDA have adopted the relational database model; a recent notable exception is [5] which also considers NoSQL databases.

As explained in the introduction, our incentive for studying OBDA is that it can provide an ontologically based approach to data quality. This involves identifying inconsistency and redundancy in OBDA mappings, as well as testing for other (un)desirable properties [10, 12, 13]. A recent survey on OBDA [20] mentions data quality as an important research direction.

When mapping relational databases to ontologies, a difficulty is that the relational database model uses value-based primary keys to identify tuples, while description logics use abstract individual names to refer to objects, possibly in combination with the Unique Name Assumption. This difficulty is nicely discussed in [13, p. 149], where a solution is proposed that uses Herbrand terms of the form $f(\boldsymbol{a})$ as individual names, where f is a function symbol and $\boldsymbol{a}$ is a sequence of database values whose length is the arity of f. By allowing multiple function symbols, this solution can distinguish between $f_{person}(\texttt{Hilton}, \texttt{Paris})$ and $f_{hotel}(\texttt{Hilton}, \texttt{Paris})$. Our approach resembles the latter solution, with one significant difference: instead of using function symbols, we use attribute names to distinguish between two sequences that contain the same data values. Such

attribute-based representation has recently appeared in the description logic $\mathcal{DLR}^+$ [2]. We do not address the problem that the same entity may be identified by different identifiers [8, 21].

3 Introductory Example

Before starting the technical development, we introduce our mapping language by means of a simple example. A fact $\mathit{ENROLLED}(c, f, \ell, p, y)$ in our example database means that student (f, ℓ) is currently enrolled in course c and took the prerequisite course p in the year y. A fact $\mathit{TAUGHT\text{-}BY}(c, f, \ell, h, s)$ means that the course c is taught by (f, ℓ) and takes place at every hour h during semester s. The same course can be taught more than once in a week.

ENROLLED	*Course*	*First*	*Last*	*Prerequisite*	*Year*
	CS402	Tom	Jones	CS311	2008
	CS402	Tom	Jones	CS401	2009

TAUGHT-BY	*Course*	*First*	*Family*	*Hour*	*Semester*
	CS402	David	Maier	Mon. 10am	Spring
	CS402	David	Maier	Tue. 10am	Spring

We will identify all persons by their first and last names, using the attributes *First* and *Last*. The operator $\pi_{\mathit{First},\mathit{Last}}$ takes the projection on *First* and *Last*. Since the table *TAUGHT-BY* uses the attribute *Family* for last names, we rename that attribute by means of the renaming operator $\delta_{\mathit{Family}\to\mathit{Last}}$. Let

$$S \coloneqq \pi_{\mathit{First},\mathit{Last}}\mathit{ENROLLED} \quad \text{and} \quad T \coloneqq \pi_{\mathit{First},\mathit{Last}}(\delta_{\mathit{Family}\to\mathit{Last}}\mathit{TAUGHT\text{-}BY}).$$

Thus, S is the set of persons that are students, and T is the set of persons that are teachers. We will identify all courses by the attribute *Course*, which necessitates the use of the renaming operator $\delta_{\mathit{Prerequisite}\to\mathit{Course}}$. Let

$$C \coloneqq \pi_{\mathit{Course}}\mathit{ENROLLED} \cup \pi_{\mathit{Course}}(\delta_{\mathit{Prerequisite}\to\mathit{Course}}\mathit{ENROLLED}) \cup \pi_{\mathit{Course}}\mathit{TAUGHT\text{-}BY}$$

Thus, C is the set of all courses. We are now ready to give three mapping rules for populating concept names Student, Teacher, and Course:

$$S : \mathsf{Student}, \qquad T : \mathsf{Teacher}, \qquad C : \mathsf{Course}.$$

Our mapping language captures negation by means of the difference operator $-$. For example, one could declare

$$S - T : \mathsf{PersonWhoDoesNotTeach}.$$

Finally, we show a mapping rule for roles. Assume we are in the spring semester, and are interested in who attends which course in the current semester. We show a mapping rule for the role name *attends*:

$$\big[S,\ C \ltimes \big(\sigma_{\mathit{Semester}=\texttt{Spring}}\mathit{TAUGHT\text{-}BY}\big),\ \mathit{ENROLLED}\big] : \mathit{attends} \qquad (2)$$

S gets all students. Next, $C \ltimes \sigma_{Semester=\texttt{Spring}}(TAUGHT\text{-}BY)$ gets all courses that take place in the spring semester. Technically, $\ltimes$ is the semijoin operator, whose effect is to return those courses in C that join with some tuple in the selection $\sigma_{Semester=\texttt{Spring}} TAUGHT\text{-}BY$. Then, the third argument *ENROLLED* specifies that a student in the first argument has to be related to a course in the second argument if they occur together in a same tuple of *ENROLLED*. The last argument, *attends*, specifies the role name for student-course pairs so obtained.

4 Preliminaries

Preliminaries from database theory. We assume a denumerable set **att** of *attributes*, a denumerable set **dom** of *constants*, and a denumerable set **relname** of *relation names*. We assume a total order $\leq_{\mathbf{att}}$ on **att**. We assume a total function *sort* with domain **relname** that maps every relation name to a finite set of attributes.

Let U be a finite set of attributes. A *tuple over* U is a total mapping from U to **dom**. A *relation over* U is a finite set of tuples over U. An *attribute renaming for* U is a total injective function from U to **att**. We write $A_1, A_2, \ldots, A_n \to B_1, B_2, \ldots, B_n$ for the attribute renaming f such that $f(A_i) = B_i$ for $i \in \{1, \ldots, n\}$ and f is the identity on other attributes. If t is a tuple over U and f is an attribute renaming, then $f(t)$ denotes the tuple s over $\{f(A) \mid A \in U\}$ such that for every $A \in U$, $s(f(A)) = t(A)$. For example, if $t = \{A : a, B : b, C : c\}$ and $f = AB \to BD$, then $f(t) = \{B : a, D : b, C : c\}$.

A *database schema* is a finite set of relation names. The following definitions are relative to a fixed database schema. A *database instance* **db** associates, to each relation name R, a finite relation over $sort(R)$, denoted $R^{\mathbf{db}}$. A database instance is also called a *database*.

Relational algebra. The operations of the relational algebra [1] are selection σ, projection π, (natural) join $\bowtie$, semijoin $\ltimes$, renaming δ, union $\cup$, and difference $-$. Conditions in selections can be equalities between attribute values and constants, that is, $\sigma_{A=B}$ and $\sigma_{A=c}$. A projection $\pi_X E$ takes the projection of E on the set X of attributes. A renaming $\delta_f E$, where f is an attribute renaming, applies f to all tuples in E. A join $E \bowtie F$ returns all tuples that can be constructed by taking the union of two tuples, one from E and one from F, that agree on their common attributes. A semijoin $E \ltimes F$ returns every tuple of E that agrees with some tuple of F on their common attributes. In the full relational algebra, $\ltimes$ is not a primitive operator, because it can be expressed as a projection of a join: $E \ltimes F \equiv \pi_{sort(E)}(E \bowtie F)$. However, semijoin is a primitive operator in the *semijoin algebra*, which allows semijoins but disallows joins. The formal semantics of all operators can be found in [1]; they define $eval(E, \mathbf{db})$, the relation to which an algebra expression E on a database **db** evaluates.

The guarded fragment of first-order logic. We define *GF* as the following restriction of predicate calculus, with equality:

- every quantifier-free formula belongs to GF;
- if $\varphi(\boldsymbol{x}, \boldsymbol{y})$ belongs to GF and $R(\boldsymbol{x}, \boldsymbol{y})$ is a relation atom in which all free variables of φ actually occur, then the formulas $\exists \boldsymbol{y}\,(R(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi(\boldsymbol{x}, \boldsymbol{y}))$ and $\forall \boldsymbol{y}\,(R(\boldsymbol{x}, \boldsymbol{y}) \rightarrow \varphi(\boldsymbol{x}, \boldsymbol{y}))$ belong to GF; and
- GF is closed under $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$.

A first-order formula is called *guarded* if it belongs to GF.

When we use formulas in predicate logic as database queries, we will make sure that these formulas are domain-independent [1, Definition 5.3.7], and we assume that constant symbols occurring in these formulas are interpreted as themselves, which incorporates the Unique Name Assumption.

5 Entity-Expressions and Relationship-Expressions

In this section, we introduce Entity-expressions and Relationship-expressions, which will be used in Section 6 to construct mapping rules. The following definitions are relative to a fixed database schema and description logic vocabulary (i.e., a finite set of concept names and role names).

Definition 1 (Entity-Expression). Entity-expressions (EEs) *are recursively defined as follows:*

1. *Every relation name is an EE.*
2. *If E is an EE and $X \subseteq sort(E)$, then $\pi_X E$ is an EE with $sort(\pi_X E) = X$.*
3. *If E is an EE and f is an attribute renaming for $sort(E)$, then $\delta_f E$ is an EE with $sort(\delta_f E) = \{f(A) \mid A \in sort(E)\}$.*
4. *If E is an EE, $A, B \in sort(E)$, and $c \in \mathbf{dom}$, then $\sigma_{A=c}E$ and $\sigma_{A=B}E$ are EEs with $sort(\sigma_{A=c}E) = sort(\sigma_{A=B}E) = sort(E)$.*
5. *If E_1 and E_2 are EEs such that $sort(E_1) = sort(E_2)$, then $E_1 \cup E_2$ and $E_1 - E_2$ are EEs with $sort(E_1 \cup E_2) = sort(E_1 - E_2) = sort(E_1)$.*
6. *If E_1 and E_2 are EEs, then $E_1 \ltimes E_2$ is an EE with $sort(E_1 \ltimes E_2) = sort(E_1)$.*

Note that Entity-expressions cannot use the join operator $\bowtie$. The fragment of relational algebra that replaces the join operator $\bowtie$ with the semijoin operator $\ltimes$ is known as the *semijoin algebra*. An important result by Leinders et al. [11] states that the semijoin algebra is contained in GF. Our setting slightly differs from this earlier work because we have attribute renamings δ_f and selections of the form $\sigma_{A=c}$, both of which are not present in [11]. The proof of the following Theorem 1 translates Entity-expressions in domain-independent formulas in the guarded fragment. It differs from [11] in that it uses constants and does not use the formulas $\mathbb{G}_k(x_1, \ldots, x_k)$ introduced by Leinders et al. for defining the guarded k-tuples of a structure.

Theorem 1. *For every Entity-expression E with $sort(E) = \{A_1, \ldots, A_n\}$, it is possible to construct a domain-independent formula $\varphi(x_1, \ldots, x_n)$ in GF such that for every database $\mathbf{db}$, for all $a_1, \ldots, a_n \in \mathbf{dom}$, $\{A_1 : a_1, \ldots, A_n : a_n\} \in eval(E, \mathbf{db})$ if and only if $\mathbf{db} \models \varphi(a_1, \ldots, a_n)$. Furthermore, φ can be constructed as a disjunction of formulas in GF, all of the form $\exists \boldsymbol{y}\,(R(\boldsymbol{x}, \boldsymbol{y}) \wedge \psi(\boldsymbol{x}, \boldsymbol{y}))$.*

The join operator $\bowtie$ can be used in Relationship-expressions, which captures a common intuition that relationships are places where entities "join."

Definition 2 (Relationship-Expression). Relationship-expressions (REs) *are recursively defined as follows:*
- *Every Entity-expression is an RE.*
- *If E_1 and E_2 are REs, then $E_1 \bowtie E_2$ is an RE.*
- *The set of REs is closed under the operators $\sigma_{A=c}$, $\sigma_{A=B}$, δ_f, $\cup$, and $-$.*

Note that the set of Relationship-expressions is not closed under projection or semijoin; for example, $T \ltimes (R \bowtie S)$ and $\pi_{AB}(R \bowtie S)$ are not Relationship-expressions. In this way, Theorem 1 remains valid if we replace "Entity-expression" with "Relationship-expression" in its statement.

6 The Mapping Language

We will now introduce the notion of *Database-to-ABox Dependency (DAD).* From here on, all definitions are relative to a database schema $\mathbb{S}$, a description logic vocabulary $\mathbb{C} \cup \mathbb{R}$ (i.e., a set of concept names and role names), and a description logic $\mathcal{DL}$. A DAD can be of two sorts: a *Concept DAD (CDAD)* takes as input a database and returns a set of concept assertions; a *Role DAD (RDAD)* takes as input a database and returns a set of role assertions.

CDAD The following definition introduces the syntax and semantics for a Concept DAD. The semantics of CDAD relies on a function ι from the set of all tuples to $\mathbb{I}$, the set of individual names.

Definition 3 (Concept DAD). *A* Concept DAD (CDAD) *is an expression $E : C$ where E is an Entity-expression (over the database schema $\mathbb{S}$) and C is a concept name in $\mathbb{C}$.*

We assume a denumerable set $\mathbb{I}$ of individual names. *We assume an injective function ι from the set of all tuples (taken over all finite subsets of* **att***) to the set of individual names. Let* **db** *be a database. The* set of concept assertions generated by $E : C$ from **db** *is the following:*

$$\{\ \iota(t) : C \ \mid t \in \mathit{eval}(E, \mathbf{db})\ \}.$$

Note that if $t_1 = \{\mathit{Lastname} : \texttt{Hilton}, \mathit{Firstname} : \texttt{Paris}\}$ and $t_2 = \{\mathit{Hotel} : \texttt{Hilton}, \mathit{City} : \texttt{Paris}\}$, then $\iota(t_1) \neq \iota(t_2)$, because ι is injective and $t_1 \neq t_2$.

RDAD The syntax for a Role DAD is slightly more complex: it is a sequence of two Entity-expressions, one Relationship-expression, and a role name r in $\mathbb{R}$. Informally, given a database, such a Role DAD generates a role assertion $(a, b) : r$ whenever a and b belong, respectively, to the result of the first and the second Entity-expression, and together fit the Relationship-expression. We give an example, and then provide a formal definition.

Example 1. Consider the following data from the Mathematics Genealogy Project at http://www.genealogy.ams.org.

PHD	*First*	*Last*	*Year*	*AdvisorFirst*	*AdvisorLast*
	Jan	Chomicki	1990	Tomasz	Imielinski
	Tomasz	Imielinski	1981	Witold	Lipski
	Witold	Lipski	1968	Wiktor	Marek

Assume that no two distinct persons in this database agree on their first and last names. Let f be a renaming such that $f(\mathit{First}) = \mathit{AdvisorFirst}$ and $f(\mathit{Last}) = \mathit{AdvisorLast}$. Thus, the inverse of f, denoted f^{-1}, maps *AdvisorFirst* and *AdvisorLast* to, respectively, *First* and *Last*. The following Entity-expression P gets first and last names of all persons in the database:

$$P \coloneqq \pi_{\{\mathit{First},\mathit{Last}\}}\mathit{PHD} \cup \delta_{f^{-1}}\big(\pi_{\{\mathit{AdvisorFirst},\mathit{AdvisorLast}\}}\mathit{PHD}\big).$$

It is significant to note that Wiktor Marek will be added with attributes *First* and *Last*, even though he does not appear with these attributes in the *PHD* table. The following RDAD populates the role name *SupervisedBy*:

$$[P, P/f, \mathit{PHD}] : \mathit{SupervisedBy}. \tag{3}$$

Given a database $\mathbf{db}$, this rule will add a role assertion $(\iota(s), \iota(t)) : \mathit{SupervisedBy}$ to the ABox whenever $s, t \in \mathit{eval}(P, \mathbf{db})$ such that some tuple of $\mathit{eval}(\mathit{PHD}, \mathbf{db})$ includes both s and $f(t)$. For example, if $s_0 = \{\mathit{First} : \texttt{Witold}, \mathit{Last} : \texttt{Lipski}\}$ and $t_0 = \{\mathit{First} : \texttt{Wiktor}, \mathit{Last} : \texttt{Marek}\}$, then $(\iota(s_0), \iota(t_0)) : \mathit{SupervisedBy}$ is added to the ABox because $s_0 \cup f(t_0) = \{\mathit{First} : \texttt{Witold}, \mathit{Last} : \texttt{Lipski}, \mathit{AdvisorFirst} : \texttt{Wiktor}, \mathit{AdvisorLast} : \texttt{Marek}\}$ is included in the last tuple of the *PHD* table.

Definition 4 (Role DAD). *A* Role DAD (RDAD) *is an expression of the form*

$$[E_1/f_1, E_2/f_2, E] : r$$

where E_1 and E_2 are Entity-expressions, f_1 and f_2 are attribute renamings, E is a Relationship-expression such that $\mathit{sort}(\delta_{f_1}E_1) \cup \mathit{sort}(\delta_{f_2}E_2) \subseteq \mathit{sort}(E)$, and r is a role name in $\mathbb{R}$.

If f_1 or f_2 is the identity, it can be omitted. Such an RDAD is called join-free *if $\bowtie$ does not occur in it (but $\ltimes$ can occur). For example, the RDADs (2) and (3) are both join-free. As for the semantics, the* set of role assertions generated by $[E_1/f_1, E_2/f_2, E] : r$ from a database $\mathbf{db}$ *is the following:*

$$\{\ (\iota(t_1), \iota(t_2)) : r \ \mid\ t_1 \in \mathit{eval}(E_1, \mathbf{db}),\ t_2 \in \mathit{eval}(E_2, \mathbf{db}),\ \text{and } f_1(t_1) \cup f_2(t_2) \subseteq t \text{ for some } t \in \mathit{eval}(E, \mathbf{db})\ \}.$$

7 Reasoning Problems

We now move from a single CDAD or a single RDAD to sets of CDADs and RDADs, and introduce some reasoning problems.

Definition 5. *Let* **db** *be a database. Let* $\mathcal{M}$ *be a set of CDADs and RDADs. We write* $\mathcal{M}(\mathbf{db})$ *for the smallest ABox that contains all concept and role assertions generated from* **db** *by the CDADs and RDADs in* $\mathcal{M}$*.*

A CDAD is said to be active on **db** *if it generates at least one concept assertion from* **db***; an RDAD is* active on **db** *if it generates at least one role assertion from* **db***.*

In the following definition, one may think of Σ as a set of database constraints. However, when studying problems like SATISFIABILITY (see below), we may add to Σ some desirable properties, like $\exists \boldsymbol{x} R(\boldsymbol{x})$ if we are asking for a satisfying database in which R is nonempty.

Definition 6. *A* DB2KB *(or* OBDA specification*) is a triple* $(\Sigma, \mathcal{M}, \mathcal{T})$ *where*
- Σ *is a set of closed first-order formulas over the database schema;*
- $\mathcal{M}$ *is a set of CDADs and RDADs; and*
- $\mathcal{T}$ *is a* $\mathcal{DL}$ *TBox.*

Console and Lenzerini [10] introduced the notions of *faithfulness* and *protection* for characterizing data quality in OBDA. These notions are recalled next, together with the well-known notion of *satisfiability*. For aesthetic reasons, we state all questions in the form "Is there a database such that...?". Therefore, we are asking for the complement of faitfulness and protection as defined in [10]. Finally, the notion of *global-consistency* appeared in [12].

INPUT: A DB2KB $(\Sigma, \mathcal{M}, \mathcal{T})$.
QUESTIONS:
- SATISFIABILITY: Is there a database **db** such that $\mathbf{db} \models \Sigma$ and the knowledge base $(\mathcal{T}, \mathcal{M}(\mathbf{db}))$ is consistent?
- NON-FAITHFULNESS: Is there a database **db** such that the knowledge base $(\mathcal{T}, \mathcal{M}(\mathbf{db}))$ is consistent but $\mathbf{db} \not\models \Sigma$?
- NON-PROTECTION: Is there a database **db** such that $\mathbf{db} \models \Sigma$ but the knowledge base $(\mathcal{T}, \mathcal{M}(\mathbf{db}))$ is inconsistent?
- GLOBAL-CONSISTENCY: Is there a database **db** such that $\mathbf{db} \models \Sigma$, all CDADs and RDADs of $\mathcal{M}$ are active on **db**, and the knowledge base $(\mathcal{T}, \mathcal{M}(\mathbf{db}))$ is consistent?

Informally, a "yes"-answer to NON-PROTECTION tells us that the ontology has some constraints not implied by Σ. Recall from Section 1 that the discovery of such constraints may be significant in data quality assessments. As mentioned just in front of Definition 6, Σ may contain desirable properties in addition to database constraints. For example, for SATISFIABILITY, we can use Σ to express that the database **db** must be nonempty.

The above problems can be shown to be undecidable in general [10, Theorem 1]. The following theorem shows their decidability under some restrictions on the input, which will be discussed after the theorem. A technical crux in the proof of Theorem 2 concerns the switch from database tuples to $\mathcal{DL}$ individual names.

Theorem 2. SATISFIABILITY, NON-FAITHFULNESS, NON-PROTECTION, *and* GLOBAL-CONSISTENCY *are decidable problems if their inputs are restricted to DB2KBs* $(\Sigma, \mathcal{M}, \mathcal{T})$ *with the following properties:*
- $\mathcal{T}$ *can be effectively expressed in GF;*
- *every formula in* Σ *is in GF; and*
- *all RDADs in* $\mathcal{M}$ *are join-free.*

Proof (Sketch). The proof shows that the four mentioned problems can be effectively reduced to satisfiability in *GF*. We first explain how the reduction deals with the function ι in Definition 3 and with $\mathcal{M}$. Let $U \coloneqq \{A_1, \ldots, A_m\}$ be the set of attributes, totally ordered, such that U includes $sort(E)$ for every $E : C$ in $\mathcal{M}$, and U includes $sort(E_1) \cup sort(E_2)$ for every $[E_1/f_1, E_2/f_2, E] : r$ in $\mathcal{M}$. Let ε be a fresh constant. For each $S \subseteq U$, we encode each tuple t over S as $(a_1, a_2, \ldots, a_m)$ where for $1 \leq i \leq m$, $a_i \coloneqq t(A_i)$ if $A_i \in S$, and $a_i \coloneqq \varepsilon$ otherwise. For example, if $U = \{\mathit{Lastname}, \mathit{Firstname}, \mathit{Hotel}, \mathit{City}\}$, then $(\varepsilon, \varepsilon, \texttt{Hilton}, \texttt{Paris})$ and $(\texttt{Hilton}, \texttt{Paris}, \varepsilon, \varepsilon)$ encode, respectively, a hotel in Paris and an actress. The reduction first uses the construction of Theorem 1 to translate Entity- and Relationship-expressions into *GF*. Next, the syntactic form in Theorem 1 allows us to translate CDADs and RDADs in *GF*. For example, for a CDAD $E : C$, every disjunct $\exists \boldsymbol{y}\,(R(\boldsymbol{x}, \boldsymbol{y}) \wedge \psi(\boldsymbol{x}, \boldsymbol{y}))$ in E's translation further translates into the guarded formula $\forall \boldsymbol{x} \forall \boldsymbol{y}\,(R(\boldsymbol{x}, \boldsymbol{y}) \rightarrow (\psi(\boldsymbol{x}, \boldsymbol{y}) \rightarrow C(t_1, \ldots, t_m)))$. Here, C is a predicate symbol of arity $m \coloneqq |U|$ that uses the encoding explained and illustrated before: for $1 \leq i \leq m$, $t_i \coloneqq x_i$ if $A_i \in sort(E)$, and $t_i \coloneqq \varepsilon$ otherwise. We next show how the reduction deals with $\mathcal{T}$. By the hypothesis of the theorem, $\mathcal{T}$ can be expressed by a formula $\varphi_{\mathcal{T}}$ in *GF*. Of course, in this formula, concept names are unary, and role names are binary. In our reduction, predicates for concept names and role names become, respectively, m-ary and $2m$-ary. To this extent, the reduction replaces in $\varphi_{\mathcal{T}}$ every occurrence of every variable x by $x_1, \ldots, x_m$. For example, the guarded formula $\forall x\,(C(x) \rightarrow D(x))$ translates into $\forall x_1 \cdots \forall x_m\,(C(x_1, \ldots, x_m) \rightarrow D(x_1, \ldots, x_m))$, a formula that is also guarded. □

The satisfiability problem for *GF* with constants is EXPTIME-complete when the arities of all relation names are fixed [17]. The EXPTIME-hard lower bound obviously carries over to the problems in Theorem 2. The EXPTIME-upper bound does not, insofar as Theorems 1 and 2 use exponential translations of CDADs and RDADs in *GF*. However, it is a plausible conjecture that membership in EXPTIME can be obtained along the lines of the proof of [11, Theorem 9].

We briefly discuss the restrictions in the statement of Theorem 2. The restriction that the input TBoxes $\mathcal{T}$ must be expressible in *GF* may be automatically fulfilled by the description logic $\mathcal{DL}$ under consideration. Indeed, many expressive description logics can be expressed in *GF* [18], an example being $\mathcal{ALC}$ [3, p. 46]. The requirement that Σ is in *GF* still allows expressing many interesting properties and database constraints, like non-emptiness of relations and inclusion dependencies, as well as all Boolean combinations of these (because *GF* is closed under Boolean combinations). On the other hand, *GF* does not include common database constraints like primary keys or functional dependencies.

Theorem 2 imposes no restrictions on CDADs, but RDADs are restricted to be join-free. This restriction is unfortunate, because it means that all Relationship-expression that occur in RDADs must actually be Entity-expressions. Informally, join-freeness demands that whenever an RDAD puts two entities together in a role, then these entities should already occur together in some database relation. This restriction is plausibly satisfied for database schemas that are obtained from Entity-Relationship diagrams that already capture such roles by relationships (as is actually the case for our example RDADs (2) and (3)). The restriction can be prohibitive though if one wants to combine in a role two entities that are unrelated in the Entity-Relationship diagram. Anyway, our proof of Theorem 2 fails if we relax one of its hypotheses, because such relaxation would take us outside GF.

Finally, we show a result telling us that database constraints can be obtained from an ontology, given a mapping $\mathcal{M}$. As we argued in Section 1, this may be of interest in data cleaning applications to infer missing database constraints.

Theorem 3. *Let $(\Sigma, \mathcal{M}, \mathcal{T})$ be a DB2KB such that $\Sigma = \emptyset$ and $\mathcal{T}$ is a DL-Lite$_{core}$ TBox. It is possible to construct a finite set Σ' of closed first-order formulas such that for every database $\mathbf{db}$, $\mathbf{db} \models \Sigma'$ if and only if $(\mathcal{T}, \mathcal{M}(\mathbf{db}))$ is a consistent knowledge base. Moreover, if every RDAD in $\mathcal{M}$ is join-free, then Σ' is in GF.*

Proof (Sketch). From [6], it follows that unsatisfiability in *DL-Lite*$_{core}$ can only arise due to some negative inclusion $C \sqsubseteq \neg D$ implied by the TBox that is violated in the ABox. The negative inclusion $C \sqsubseteq \neg D$ can be of four different forms, where A, B denote concept names, and r, s role names: $A \sqsubseteq \neg B$, $A \sqsubseteq \neg\exists r$, $\exists r \sqsubseteq \neg A$, or $\exists r \sqsubseteq \neg\exists s$. We show here how to deal with $A \sqsubseteq \neg B$ (the other cases are similar): for all CDADs $E : A$ and $F : B$ in $\mathcal{M}$ such that $sort(E) = sort(F)$, Σ' contains a formula stating emptiness of $\pi_{\{\}}(E \ltimes F)$. The latter algebra expression is an Entity-expression, and thus, by Theorem 1, can be expressed in GF. □

8 Conclusion

The language of CDADs and RDADs allows expressing database-to-ontology mappings in a user-friendly way. The language is based on the semijoin algebra, which is embedded in the guarded fragment of first-order logic. This results in decidability of some important reasoning problems. Since CDADs and RDADs allow full negation, they can express mappings that are not GLAV mappings. On the other hand, the GLAV mapping $\forall x \forall y \forall z\,(R(x, y) \wedge R(y, z) \wedge R(z, x) \rightarrow C(x))$ is not guarded and cannot be expressed as a CDAD. In future research, we plan to explore in more depth the practice of using ontological knowledge in database repairing, database cleaning, and consistent query answering. We also plan to investigate how our framework can be reconciled with $\mathcal{DLR}^+$ [2], a description logic tailored towards relational databases.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. A. Artale, E. Franconi, R. Peñaloza, and F. Sportelli. A decidable very expressive description logic for databases. In *International Semantic Web Conference 2017*, pages 37–52, 2017.
3. F. Baader, I. Horrocks, C. Lutz, and U. Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.
4. M. Bienvenu. Inconsistency-tolerant ontology-based data access revisited: Taking mappings into account. In *IJCAI 2018*, pages 1721–1729, 2018.
5. E. Botoeva, D. Calvanese, B. Cogrel, J. Corman, and G. Xiao. A generalized framework for ontology-based data access. In *AI*IA 2018*, pages 166–180, 2018.
6. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
7. D. Calvanese and E. Franconi. First-order ontology mediated database querying via query reformulation. In S. Flesca, S. Greco, E. Masciari, and D. Saccà, editors, *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years.*, volume 31 of *Studies in Big Data*, pages 169–185. Springer International Publishing, 2018.
8. D. Calvanese, M. Giese, D. Hovland, and M. Rezk. Ontology-based integration of cross-linked datasets. In *International Semantic Web Conference 2015*, pages 199–216, 2015.
9. P. P. Chen. The Entity-Relationship Model – Toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.
10. M. Console and M. Lenzerini. Data quality in ontology-based data access: The case of consistency. In *AAAI 2014*, 2014.
11. D. Leinders, M. Marx, J. Tyszkiewicz, and J. V. den Bussche. The semijoin algebra and the guarded fragment. *Journal of Logic, Language and Information*, 14(3):331–343, 2005.
12. D. Lembo, J. Mora, R. Rosati, D. F. Savo, and E. Thorstensen. Mapping analysis in ontology-based data access: Algorithms and complexity. In *International Semantic Web Conference 2015*, pages 217–234, 2015.
13. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.
14. J. F. Sequeda. Integrating relational databases with the semantic web: A reflection. In *Reasoning Web 2017*, pages 68–120, 2017.
15. J. F. Sequeda, S. H. Tirmizi, Ó. Corcho, and D. P. Miranker. Survey of directly mapping SQL databases to the semantic web. *Knowledge Eng. Review*, 26(4):445–486, 2011.
16. D. Spanos, P. Stavrou, and N. Mitrou. Bringing relational databases into the semantic web: A survey. *Semantic Web*, 3(2):169–209, 2012.
17. B. ten Cate and M. Franceschet. Guarded fragments with constants. *Journal of Logic, Language and Information*, 14(3):281–288, 2005.
18. C. Thorne, R. Bernardi, and D. Calvanese. Designing efficient controlled languages for ontologies. In *Computing Meaning: Volume 4*, pages 149–173. Springer Netherlands, Dordrecht, 2014.
19. Y. Wand and R. Y. Wang. Anchoring data quality dimensions in ontological foundations. *Commun. ACM*, 39(11):86–95, 1996.

20. G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati, and M. Zakharyaschev. Ontology-based data access: A survey. In *IJCAI 2018*, pages 5511–5519, 2018.
21. G. Xiao, D. Hovland, D. Bilidas, M. Rezk, M. Giese, and D. Calvanese. Efficient ontology-based data integration with canonical IRIs. In *ESWC 2018*, pages 697–713, 2018.