On the Data Complexity of Consistent Query Answering over Graph Databases

Pablo Barceló and Gaëlle Fontaine

Department of Computer Science
University of Chile
pbarcelo@dcc.uchile.cl, gaelle@dcc.uchile.cl

Abstract

Areas in which graph databases are applied – such as the semantic web, social networks and scientific databases – are prone to inconsistency, mainly due to interoperability issues. This raises the need for understanding query answering over inconsistent graph databases in a framework that is simple yet general enough to accommodate many of its applications. We follow the well-known approach of consistent query answering (CQA), and study the data complexity of CQA over graph databases for regular path queries (RPQs) and regular path constraints (RPCs), which are frequently used. We concentrate on subset, superset and symmetric difference repairs. Without further restrictions, CQA is undecidable for the semantics based on superset and symmetric difference repairs, and Π_2^{P}-complete for subset repairs. However, we provide several tractable restrictions on both RPCs and the structure of graph databases that lead to decidability, and even tractability of CQA. We also compare our results with those obtained for CQA in the context of relational databases.

1998 ACM Subject Classification H.2.3 Database Management – Query Languages

Keywords and phrases graph databases, regular path queries, consistent query answering, description logics, rewrite systems

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.380

1 Introduction

Query languages for graph databases are typically navigational, in the sense that they allow for recursively traversing the labeled edges while checking for the existence of a path whose label satisfies a particular regular condition (see, e.g., [39, 5]). The basic building block for navigational languages over graph databases is the class of *regular path queries*, or RPQs [14]. Each RPQ is a regular expression L, and its evaluation $L(G)$ over a graph database G corresponds to a binary relation that contains all pairs of nodes in G that are linked by some path whose label matches L. The evaluation problem for RPQs can be solved in NLogspace in *data complexity*; that is, when the RPQ is fixed (cf., [5]).

Although graph databases are schema-less, it is possible to enforce data consistency over them using *path constraints* [1, 9]. These constraints have been used in several scenarios that are based on the graph database paradigm, e.g., to express local knowledge about semi-structured data [1]; to enforce restrictions over object-oriented databases, XML and RDF [36, 12, 22, 3, 30]; and to capture ontological hierarchies in the context of description logics (DLs) [16, 17]. Here we concentrate on a simple class of path constraints based on RPQs that was introduced by Abiteboul and Vianu; namely, the *regular path constraints*, or RPCs [1]. An RPC is an expression of the form $L_1 \sqsubseteq L_2$, where L_1 and L_2 are RPQs. In the graph database and DL contexts, a graph database G satisfies $L_1 \sqsubseteq L_2$ if and only if $L_1(G) \subseteq L_2(G)$ [26, 16, 17] (but we also consider a more restrictive semantics for RPCs,

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 380–397

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

motivated by their application over semi-structured data, following the original proposal of Abiteboul and Vianu). RPCs are a constituent part of the semantics of graph data and can also be used in the optimization process for RPQ evaluation [1].

An important problem when dealing with dependencies is that databases might be *inconsistent*, i.e., the databases might fail to satisfy the integrity constraints. In the case of graph database applications, high levels of inconsistency appear due to interoperability and distribution (e.g., in RDF and social/scientific networks [27, 40]). As an example, inconsistency might arise while integrating several sources into a single RDF graph, or while performing statistical inference on a scientific or social network. This raises the need for developing an *inconsistency-tolerant* semantics for graph databases in a simple yet general framework that abstracts away from its many implementations. In order to tackle this problem, we use the widespread approach of *consistent query answering* (as first introduced in the seminal work of Arenas, Bertossi and Chomicki [2]), which we now describe.

The approach is based on the notion of *repair*, which represents a possible minimal way in which consistency over the data could be restored. More formally, a repair is a database that satisfies the constraints and "minimally differs" from the original database. In general, a database does not admit a unique repair. This leads to an inconsistency-intolerant semantics based on the *consistent answers* of a query, i.e., the answers that hold in every possible repair. The problem of computing consistent answers to a query is known as *consistent query answering* (CQA).

Here we study the data complexity of CQA over graph databases in the scenario in which queries are RPQs and constraints are RPCs. That is, we study the complexity of evaluating consistent answers over an inconsistent graph database G for a *fixed* RPQ L under a *fixed* set Γ of RPCs. We explain next the context and main contributions of our work.

The context. The complexity of CQA has received considerable attention over different data models, notions of repairs and classes of constraints. Under the relational model, for instance, this problem has been studied for set-based [2, 18, 24], cardinality-based [29], and attribute-based repairs [37]; for constraints expressed as traditional functional dependencies, inclusion dependencies and denial constraints (see, e.g., [11, 19, 25, 38, 28, 24]); and for constraints expressed as tuple-generating dependencies (tgds) and equality-generating dependencies (egds) that arise in the context of data integration and data exchange [18].

CQA has also been studied in depth in the DL context, starting from the work of Lembo and Ruzzi [31]. The DL semantics is *open-world* in nature, meaning that the non-presence of a fact is not sufficient to ensure that the negation of the fact holds. This implies that in the DL context the only meaningful set-based repairs are the *subset* repairs; i.e., those that allow to restore consistency with respect to the DL constraints (i.e., the ones in the *TBox*) only by *deleting* facts from the database (i.e., the *ABox*). It is worth mentioning that the notion of DL repair is slightly different to its relational counterpart: A DL repair is not a subinstance that satisfies the constraints in the TBox, but one that does not lead to a contradiction in conjunction with those constraints [31, 32].

Some applications of graph databases, such as RDF, are open-world in nature. However, graph databases are not tied to this interpretation and may also accommodate closed-world applications. Therefore, there should be no a priori restriction on the class of set-based repairs one allows in this context. We thus study CQA for graph databases under the three usual notions of set-based repairs: *subset*, *superset*, and *symmetric difference* repairs [18].

Our contributions. We first look at the data complexity of CQA over graph databases under subset repairs. The problem is in Π_2^{P}, and we prove that it is complete for this class in

restricted cases. Moreover, it remains intractable even under an approximation semantics (based on the intersection of all subset repairs) that is motivated by the DL context [32].

In order to deal with this high complexity, we provide tractable cases by restricting the class of RPCs or the class of graph databases allowed. In the first case, we prove that the problem is tractable if RPCs are in *LAV* form, i.e., if they are of the form $a \sqsubseteq L$, where a is a single letter. This result is, in a sense, tight, since allowing a single RPC of the form $ab \sqsubseteq c$, for symbols a, b and c, leads to intractability. In the case of restrictions on graph databases, by applying a deep result of Courcelle [20] we obtain that the data complexity of CQA under subset repairs is tractable over graph databases of *bounded treewidth*.

We then move to study CQA under superset and symmetric difference repairs, and prove that in both cases the problem is undecidable. However, we obtain decidability by either restricting the class of RPCs allowed or their semantics. Let us consider first the restrictions on RPCs. We prove that if RPCs are in LAV form then the data complexity of CQA is tractable when dealing with symmetric difference repairs. We leave open whether CQA is also decidable under the semantics of superset repairs, but prove that at least tractability in data complexity is not preserved. We then prove that if RPCs are in *GAV* form, i.e., if they are of the form $L \sqsubseteq a$, where a is a single letter, our CQA problem is tractable under the semantics of superset repairs, but intractable under symmetric difference repairs. With respect to restrictions on the semantics of RPCs, we prove that by forcing RPCs to be read from a particular node in the graph database, called the *origin* (which corresponds to the original semantics Abiteboul and Vianu defined for these constraints), the problem can be solved in coNP under superset repairs.

Comparison with previous results. The main difference between the query and constraint languages we study here (RPQs and RPCs) and the ones studied in the relational context, is that our languages allow recursion while CQA has been studied in the relational world mostly in the absence of recursive features. Interestingly, our results show that recursion does not add to the complexity of the problems studied. In fact, almost all of our lower bounds hold in the restricted setting in which RPQs do not mention the Kleene-star and RPCs are *word constraints* of the form $w_1 \sqsubseteq w_2$, for words w_1 and w_2 [1].

If we interpret graph databases as relational databases, word constraints can be represented as tgds. The tgds corresponding to word constraints have a special restricted structure and are called *chain* tgds. On the other hand, CQA under tgds has been intensively studied [18], and it is tempting to think that lower bounds obtained in such setting could be adapted to work in our scenario. This is not the case, however, as those proofs do not apply to the class of chain tgds. Actually, the proofs of our lower bounds are considerably more involved than the ones for arbitrary tgds. As a side-effect, we obtain that several of our lower bounds extend to CQA in the relational case for queries defined as unions of CQs under chain tgds.

DL databases are (essentially) graph databases. In such scenario, Π_2^{P}-hardness results have been obtained by Rosati for the data complexity of CQA under subset repairs, in particular, for the case when queries are unions of CQs and constraints are expressed in the logic $\mathcal{ALC}$ [35]. However, constraints in this logic cannot be directly expressed as RPCs due to the presence of negation. Furthermore, the notion of repair in [35] is different to ours.

Organisation of the paper. Preliminaries are in Section 2. Results about the subset repair semantics are in Section 3, and those about the semantics of superset and symmetric difference repairs are in Section 4. Comparison with previous work is in Section 5 and conclusions in Section 6. Due to space constraints, complete proofs are in the appendix.

2 Preliminaries

Graph databases and regular path queries. As it is customary in the graph database literature [14, 15, 39, 5], we consider graph databases to be finite, edge-labeled and directed graphs. Formally, let Σ be a finite alphabet. A *graph database* $G = (V, E)$ over Σ consists of a finite set V of nodes and a set of labeled edges $E \subseteq V \times \Sigma \times V$. We interpret each tuple $(u, a, v) \in E$, for $u, v \in V$ and $a \in \Sigma$, as an edge from node u to node v whose label is a. If $G = (V, E)$ and $G' = (V', E')$ are graph databases over Σ, we write $G \subseteq G'$ to denote that $V \subseteq V'$ and $E \subseteq E'$. If in addition it is not the case that $G' = G$, we write $G \subsetneq G'$.

Navigational query languages for graph databases, such as the one of RPQs, express properties of *paths*. Formally, a path π in $G = (V, E)$ of length m (where $m > 0$) from node v_0 to node v_m is a sequence of the form $(v_0, a_1, v_1)(v_1, a_2, v_2) \dots (v_{m-1}, a_m, v_m)$, where (v_{i-1}, a_i, v_i) is an edge in E, for each $1 \leq i \leq m$. The *label* of π, denoted $\lambda(\pi)$, is the string $a_1 a_2 \dots a_m \in \Sigma^*$. A path π of length 0 is simply a node v and its label $\lambda(\pi)$ is the empty string ε.

Here we concentrate on the simplest navigational language for graph databases, namely, *regular path queries*, or RPQs [14]. An RPQ is a regular expression L over Σ. The evaluation $L(G)$ of L over a graph database $G = (V, E)$ is the set of pairs (u, v) of nodes in V for which there is a path π in G from u to v such that $\lambda(\pi)$ satisfies L. If L does not mention the Kleene-star (i.e., if L defines a finite language) then we say that L is *non-recursive*. It is well-known (cf., [5]) that computing $L(G)$, for an RPQ L and a graph database G, can be solved in polynomial time (and in NLogspace if L is fixed, that is, in data complexity).

Regular path constraints. Graph database constraints based on the class of RPQs are known as *regular path constraints* [1, 26], or RPCs. Formally, an RPC over Σ is an expression of the form $L_1 \sqsubseteq L_2$, where L_1 and L_2 are RPQs over Σ. A *word constraint* is an RPC in which both L_1 and L_2 are words.

An RPC $L_1 \sqsubseteq L_2$ expresses that the evaluation of the RPQ L_1 is contained in the evaluation of the RPQ L_2 [26]. Formally, a graph database G *satisfies* $L_1 \sqsubseteq L_2$, denoted $G \models L_1 \sqsubseteq L_2$, if and only if $L_1(G) \subseteq L_2(G)$. If Γ is a finite set of constraints, we write $G \models \Gamma$ to denote that for each RPC $L_1 \sqsubseteq L_2$ in Γ it is the case that $G \models L_1 \sqsubseteq L_2$. It follows from previous remarks that the problem of checking whether $G \models \Gamma$, for a fixed set Γ of RPCs, is in NLogspace.

▶ **Example 1.** Let Γ be the following set of RPCs:

1. $\texttt{child\_of} \sqsubseteq \texttt{son\_of} \cup \texttt{daughter\_of}$.
2. $\texttt{brother\_of} \cdot (\texttt{brother\_of} \cup \texttt{sister\_of}) \sqsubseteq \texttt{brother\_of}$.
3. $\texttt{sister\_of} \cdot (\texttt{brother\_of} \cup \texttt{sister\_of}) \sqsubseteq \texttt{sister\_of}$.
4. $\texttt{child\_of} \cdot (\texttt{brother\_of} \cup \texttt{sister\_of}) \sqsubseteq \texttt{is\_nephew} \cup \texttt{is\_niece}$.

Intuitively, the first RPC expresses that if u is a child of v, then u is a son or a daughter of v. The second and third RPCs express that if u is a brother (resp., sister) of v and v is a sibling of w, then v is a brother (resp., sister) of w. The forth RPC states that each child of a person v is the niece or nephew of every sibling of v. ◀

Repairs. A *repair* of a database D under a set of constraints Γ is a database D' that satisfies Γ but "minimally" differs from D [2]. We formalise this idea for graph databases and RPCs below, following closely its formalisation in the relational context [2].

The *symmetric difference* between two relational databases D and D' is defined as a database $D \oplus D'$ that contains those "facts" that belong to D but not to D' or to D' but

not to D. We can analogously define the symmetric difference between two graph databases $G = (V, E)$ and $G' = (V', E')$ over the same alphabet Σ, as the graph database

$$G \oplus G' \;:=\; \big(V_0,\, (E \oplus E')\big),$$

where $E \oplus E' := (E \setminus E') \cup (E' \setminus E)$ and V_0 is the set of nodes occurring in $E \oplus E'$. That is, the nodes (resp., edges) of $G \oplus G'$ are those nodes (resp., edges) that appear in either G or G' but not in both. Notice that $G \oplus G'$ is also a graph database over Σ.

Three notions of set-based repairs have been studied in the literature [18]: the *subset*, *superset* and *symmetric difference* repairs ($\subseteq$–, $\supseteq$–, and $\oplus$–repairs, respectively). We introduce them below in the context of graph databases. Let $G = (V, E)$ and $G' = (V', E')$ be two graph databases over Σ, and assume that Γ is a finite set of RPCs over Σ. Then:

1. G' is a $\oplus$-repair (i.e., symmetric difference repair) of G under Γ, if (1) $G' \models \Gamma$, and (2) there is no graph database G'' over Σ such that $G'' \models \Gamma$ and $G \oplus G'' \subsetneq G \oplus G'$.
2. G' is a $\subseteq$-repair (i.e., subset repair) of G under Γ, if $G' \subseteq G$ and G' is a $\oplus$-repair of G. Equivalently, if (1) $G' \subseteq G$, (2) $G' \models \Gamma$, and (3) there is no graph database G'' over Σ such that $G'' \models \Gamma$ and $G' \subsetneq G'' \subseteq G$.
3. G' is a $\supseteq$-repair (i.e., superset repair) of G under Γ, if $G \subseteq G'$ and G' is a $\oplus$-repair of G. Equivalently, if (1) $G \subseteq G'$, (2) $G' \models \Gamma$, and (3) there is no graph database G'' over Σ such that $G'' \models \Gamma$ and $G \subseteq G'' \subsetneq G'$.

▶ **Example 2** (Example 1 cont.)**.** Consider a graph database G whose set of edges is

$$\{(a, \texttt{child\_of}, b), (b, \texttt{sister\_of}, c), (c, \texttt{brother\_of}, d)\}.$$

Then G has two $\subseteq$-repairs under Γ: $\{(b, \texttt{sister\_of}, c)\}$ and $\{(c, \texttt{brother\_of}, d)\}$. On the other hand, G has eight $\supseteq$-repairs under Γ. One of them is the one that extends G with edges:

$$\{(a, \texttt{son\_of}, b), (b, \texttt{sister\_of}, d), (a, \texttt{is\_nephew}, c), (a, \texttt{is\_nephew}, d)\}.$$

Finally, G has seven $\oplus$-repairs under Γ that are neither $\subseteq$-repairs nor $\supseteq$-repairs. One of them is $\{(b, \texttt{sister\_of}, c), (c, \texttt{brother\_of}, d), (b, \texttt{sister\_of}, d)\}$. ◀

Repairs might not exist in some situations. Consider an RPC of the form $L \sqsubseteq \varepsilon$, where ε is the empty word, and a graph database G that consists of nodes u and v linked by a path labeled in L. Assume that G has a $\supseteq$-repair H. Then it must be the case that $u = v$ in H, which is impossible. As the the next lemma shows, repairs exist in all other cases.

▶ **Lemma 3.** *Let $G = (V, E)$ be a graph database and Γ a finite set of RPCs over Σ. Then:*

1. *There is a $\subseteq$-repair and a $\oplus$-repair of G under Γ.*
2. *If Γ contains no RPC of the form $L \sqsubseteq \varepsilon$, then there is a $\supseteq$-repair of G under Γ.*

Proof. Let $G = (V, E)$ be a graph database and Γ a finite set of RPCs. The empty database $G_\emptyset = (\emptyset, \emptyset)$ satisfies Γ. Hence, there is an $\subseteq$-repair H of G under Γ such that $G_\emptyset \subseteq H \subseteq G$. By definition, H is also a $\oplus$-repair of G under Γ. Consider now the graph database $G_c = (V, V \times \Sigma \times V)$. It is easy to see that G_c satisfies Γ since Γ contains no RPC of the form $L \sqsubseteq \varepsilon$. Since $G \subseteq G_c$, there is a $\supseteq$-repair H of G under Γ such that $G \subseteq H \subseteq G_c$. ◀

Consistent query answering. We are now ready to define our most important notion, that of a *consistent answer* to an RPQ. The consistent answers are the pairs of nodes that belong to the evaluation of the RPQ over every single repair of the original graph database.

▶ **Definition 4** (Consistent answers). Assume that $\star \in \{\oplus, \subseteq, \supseteq\}$. Let $G = (V, E)$ be a graph database, Γ a set of RPCs and L an RPQ, all of them over Σ. We define the set $\star$-Cons(G, L, Γ) of *$\star$-consistent answers of L over G under Γ* as:

$$\star\text{-Cons}(G, L, \Gamma) \;=\; \bigcap \{L(G') \mid G' \text{ is a } \star\text{-repair of } G \text{ under } \Gamma\}. \quad \blacktriangleleft$$

▶ **Example 5.** (Example 1 cont.) Consider the RPQ $L = \texttt{child\_of} \cdot \texttt{sister\_of}$. The pair (a, d) belongs to $\supseteq$-Cons(G,L,Γ). This pair also belongs to $\supseteq$-Cons(G,L',Γ), for $L' = \texttt{is\_nephew} \cup \texttt{is\_niece}$. On the other hand, the only way in which a pair (u, v) can belong to $\subseteq$-Cons(G,L'',Γ), for an RPQ L'', is when $u = v = c$ and $L'' = \varepsilon$. $\blacktriangleleft$

Here we study the data complexity of the problem of computing certain answers. We formalise this decision problem as follows. Assume that L is an RPQ and Γ is a finite set of RPCs over Σ. We denote by $\star$-CQA(L,Γ) the problem of, given a graph database $G = (V, E)$ over Σ and a pair (u, v) of nodes in V, checking whether $(u, v) \in \star$-Cons(G, L, Γ).

3 CQA under Subset Repairs

We start by proving that under the subset repair semantics our CQA problem is Π_2^{P}-complete. This holds even in the case in which all RPCs are word constraints and the RPQ is non-recursive.

▶ **Theorem 6.**

1. *For each RPQ L and finite set Γ of RPCs over the same alphabet Σ, it is the case that $\subseteq$-CQA(L,Γ) is in Π_2^{P}.*
2. *There exist a finite alphabet Σ, a non-recursive RPQ L and a finite set Γ of word constraints over Σ, such that $\subseteq$-CQA(L,Γ) is Π_2^{P}-complete.*

Proof. We sketch the hardness proof, i.e., that there is a non-recursive RPQ L and a set Γ of word constraints such that $\subseteq$-CQA(L,Γ) is Π_2^{P}-hard. We do so by providing a reduction from the quantified boolean satisfaction problem for Π_2^{P} to $\subseteq$-CQA(L,Γ).

Let ϕ be a quantified boolean formula of the form $\forall X \exists Y \psi$, where X, Y are disjoint sets of variables and ψ is of the form $(z_{11} \vee z_{12} \vee z_{13}) \wedge \cdots \wedge (z_{n1} \vee z_{n2} \vee z_{n3})$, for $z_{ij} \in \{x, \neg x, y, \neg y \mid x \in X, y \in Y\}$ $(1 \leq i \leq n,\ 1 \leq j \leq 3)$. For all i, j, we define u_{ij} as the variable z_{ij} if $z_{ij} \in X \cup Y$, or we define u_{ij} as u if z_{ij} is of the form $\neg u$. Without loss of generality, we may assume that in each clause $C_i := z_{i1} \vee z_{i2} \vee z_{i3}$, there is at least one variable in[1] Y.

We will define the RPQ L, the constraints Γ and associate a graph database G_ϕ with ϕ in such a way that

$$\phi \text{ is satisfiable} \quad \text{iff} \quad (n_s, n_s) \in \subseteq -\text{Cons}(G_\phi, L, \Gamma).$$

where n_s is a node of G_ϕ.

[1] Suppose for example that the clause C_1 of ψ contains only variables in X. We construct a new formula ψ' defined by $(z_{11} \vee z_{12} \vee y) \vee (\neg y \vee z_{13} \vee z_{13}) \vee C_2 \wedge \cdots \wedge C_n$, where y is a new fresh variable. We define Y' as $Y \cup \{y\}$. Then $\forall X \exists Y \psi$ is satisfiable iff $\forall X \exists Y' \psi'$ is satisfiable.

It is convenient to start defining a graph database G'_ϕ over alphabet

$$\Sigma \;=\; \{t, f, t_0, f_0, a, y, z, w, d, s, e, r_{-,-}, r_{-,+}, r_{+,-}, r_{+,+}\}.$$

The nodes of G'_ϕ are the following set:

$$V \;:=\; \{n_{ij} \mid 1 \leq i \leq n,\, 1 \leq j \leq 3\} \;\cup\; \{n_t, n_f, n_s\}.$$

With each variable u_{ij}, we associate a node id n_{ij} in V. Observe that even if $u_{ij} = u_{kl}$ (with $(i,j) \neq (k,l)$), the associated nodes are distinct. We also have three special nodes n_s, n_t and n_f. We let $<$ be an arbitrary irreflexive partial order over $\{1,\ldots,n\} \times \{1,2,3\}$ such that (a) for each $1 \leq i,k \leq n$ and $1 \leq j,l \leq 3$, if $u_{ij} = u_{kl}$, then $(i,j) < (k,l)$ or $(k,l) < (i,j)$, and (b) for each (i,j), where $1 \leq i \leq n$ and $1 \leq j \leq 3$, there is at most one pair (k,l) such that (k,l) is a "successor" of (i,j) with respect to $<$ (where we say that (k,l) is a *successor* of (i,j) if $(i,j) < (k,l)$ and there is no (k',l') such that $(i,j) < (k',l') < (k,l)$).

The set of edges of G'_ϕ is defined as the union of all the sets below:

$$E_t = \{(n_{ij}, t, n_{ij}) \mid 1 \leq i \leq n,\, 1 \leq j \leq 3\} \qquad E_f = \{(n_{ij}, f, n_{ij}) \mid 1 \leq i \leq n,\, 1 \leq j \leq 3\}$$
$$E_{t_0} = \{(n_t, t_0, n_t)\} \qquad E_{f_0} = \{(n_f, f_0, n_f)\}$$
$$E_a = \{(n_{ij}, a, n_s) \mid u_{ij} \in Y\} \qquad E_y = \{(n_s, y, n_s)\}$$
$$E_z = \{(n_{ij}, z, n_t), (n_{ij}, z, n_f) \mid u_{ij} \in Y\} \qquad E_w = \{\{(n_t, w, n_s), (n_f, w, n_s)\}$$
$$E_d = \{(n_{ij}, d, n_{kl}) \mid (i,j) < (k,l)\} \qquad E_s = \{(u, s, v) \mid u, v \in V\}$$
$$E_{r_{\sim\sharp}} = \{(n_{ij}, r_{\sim\sharp}, n_{i(j+1)}) \mid z_{ij} = {\sim} u_{ij},\, z_{i(j+1)} = \sharp u_{i(j+1)},\, 1 \leq i \leq n,\, 1 \leq j \leq 2\}$$
$$E_e = \emptyset$$

where $\sim, \sharp$ belong to $\{+,-\}$ and $+u := u$, $-u := \neg u$.

The intuition is as follows. There is no edge with label e and there is an edge with label s between any two nodes. For the labels t and f, each node n_{ij} associated with a variable admits loops with labels t and f. The $\subseteq$-repairs will be such that each such node admit at most one loop, either with label t or f. This allows us to define a partial map associating a truth value with each node n_{ij} ($\top$ if the loop has label t and $\bot$ if the loop has label f). The special node n_t has a loop with label t_0 and the special node n_f has a loop with label f_0.

The labels r_{++}, r_{-+}, r_{+-} and r_{--} express which variables occur in the same clause and whether each variable occurs positively or negatively. The label d specifies which nodes corresponds to the same variable. The constraints will be such that in a $\subseteq$-repair, two nodes linked by d (corresponding to the same variables) admit loops with the same label (t or f). This implies that the partial mapping associating a truth value to each node ($\top$ if the loop has label t and $\bot$ if the loop has label f), corresponds to a partial valuation over the variables in $X \cup Y$.

The special node n_s admits a loop with label y. That loop will act as a "witness". If that loop appears in a repair, we will say that the node n_s gets *activated*. When n_s is activated in a repair, the idea is that each node n_{ij} associated with a variable in Y admits a loop with label t or f. That is, each such a node receives a truth value ($\top$ if the loop has label t and $\bot$ otherwise). Conversely, if in a repair one node associated with a variable in Y admits a loop with label t or f, then the presence of that loop "activates" the node n_s (i.e. n_s admits a loop with label y). As a consequence, all the nodes associated with variables in Y will admit a loop. Basically, the node n_s guarantees that either all the nodes associated with variables in Y admit a loop, or none of those nodes admits a loop.

In order to encode in the graph database which variables belong to Y, we add an edge with label a between the source n_s and each node of the form n_{ij} with $u_{ij} \in Y$. We also add an edge with label z from n_{ij} to the special node n_t and from n_{ij} to n_f.

We use the nodes n_t and n_f in the following way. Recall that in a given $\subseteq$-repair, the (possible) truth value of a node is given by the label (either t or f) of the loop of the node. Now, for the nodes associated with variables in Y, we have an extra way of encoding the truth value. If the truth value of a node n_{ij} (with $u_{ij} \in Y$) is true, this will also be witnessed by an arrow with label z from n_{ij} to the special node n_t (and the other arrow with label z from n_{ij} to n_f is deleted). If the truth value is false, this will be witnessed by an arrow with label z from n_{ij} to n_f.

We now define the set Γ of constraints. We have six different sets of constraints (C1), (C2), (C3), (C4), (C5) and (C6) in Γ. The set (C2) contains the constraint

$$tf \sqsubseteq e.$$

Since e is empty, it says that a node cannot admit both a loop with label t and f. The set (C3) contains the two constraints

$$td \sqsubseteq dt$$
$$fd \sqsubseteq df.$$

It expresses that if two nodes are linked by d (hence, they are associated with the same variable), then they must admit a loop with the same label (in case they admit a loop). This guarantees that the partial map associating a truth value to each node (depending on the label of the loop), can be transformed into a partial valuation over the variables. The set of constraints (C4) is given by

$$F(j)r_{jk}F(k)r_{kl}F(l) \sqsubseteq e,$$

where $j, k, l \in \{+, -\}$, $F(+) = f$ and $F(-) = t$. These constraints express that the formula ψ is not false under the partial valuation associated with the repair.

The constraints (C5) are given by

$$zt_0 \sqsubseteq tz$$
$$zf_0 \sqsubseteq fz.$$

They express that if a node associated with a variable in Y is linked to the node n_t, then it admits a loop with label t. Similarly, if such a node is linked to the node n_f, then it admits a node with label f.

Indeed, let us look for example at the constraint $zt_0 \subseteq tz$. Suppose that there is a path with label zt_0 from a node u to a node v. By definition of t_0 and z, this can only happen if u is a node of the form n_{ij} (with $u_{ij} \in Y$) and v is the node n_t. Now since $zt_0 \subseteq tz$, this implies that n_{ij} admits a loop with label t.

The set (C6a) contains the two constraints

$$ta \sqsubseteq ay$$
$$fa \sqsubseteq ay$$

while the constraint (C6b) is given by $ay \subseteq zw$. The constraints (C6a) express that if one node associated with a variable in Y admits a loop with label t or f, then the node n_s is activated (i.e. admits a loop with label y). Indeed, if there is a path with label ta from a node u to a node v, then, u must be a node of the form n_{ij} admitting a loop with label t and v is the node n_s. Moreover, since n_{ij} admits an outgoing edge with label a, n_{ij} must be associated with a variable in Y. Since $ta \subseteq ay$, the presence of the loop with label t and the fact that u_{ij} belongs to Y imply that there is an edge with label y. That is, n_s is activated.

The constraint (C6b) expresses that if the node n_s is activated (admits a loop with label y), then each node n_{ij} associated with a variable in Y is linked either to the node n_t or to the node n_f. Together with the contraints (C5), this means that if n_s is activated, then each node n_{ij} associated with a variable in Y admits a loop with label t or f.

We define an RPQ L' as sys. This RPQ evaluates to $V \times V$ in the $\subseteq$-repairs admitting a y-labeled edge, i.e., the repairs in which n_s is activated. Recall that this means that the partial valuation associated with the repair assigns a truth value to all the variables in Y.

We would like to prove that ϕ is satisfiable iff (n_s, n_s) belongs to $\subseteq$-Cons(G'_ϕ, L', Γ). Now the implication from left to right is not true. The problem is that since we are allowed to delete edges in order to obtain a repair, we may loose some "relevant information". For example, in a repair, we may loose an edge with label a encoding the fact that a node is associated with a variable in Y. In such repairs, it might not be the case that (n_s, n_s) belongs to the answer of L'.

The solution is to extend the graph database G'_ϕ with a set of extra edges E, define a new graph G_ϕ, and add a set of constraints (C1). If in a repair H of G_ϕ we lost some "relevant information", the constraints (C1) will be such that at least one extra edge in E occurs in H. We then modify the RPQ L' into an RPQ L, in such a way that if a graph database contains a new edge in E, then the evaluation of L consists of all the pairs of nodes. Hence, in all the repairs in which we lose some relevant information, the answer of the RPQ contains any pair of nodes.

As an example, we show how to modify the graph G'_ϕ into a graph G_a, add new constraints (C1d) and modify the RPQ L into a RPQ L_a in such a way that if in a repair H of G_a, we "lost" an edge with label a, then H trivially satisfies[2] L_d. We define G_a by adding the following edges

$$E_{a'} = \{(n_{ij}, a', n_{ij}) \mid (n_{ij}, a, n_s) \in E_a\}.$$

That is, we add a loop with label a' to all the nodes who must admit an outgoing edge with label a. We define (C1d) as the following constraint

$$a'a \subseteq e.$$

Using the maximality property of the repairs, we can show that this constraint ensure that in each repair H, exactly one of the following properties holds: (a) we did not loose any edge with label a, and there is no edge with label a' or e, (b) there is an edge with label a' or e. Case (a) means that we did not lost any "relevant information" with respect to the label a. In case (b), the fact that we lost that information is witnessed by the fact that in the repair, there is an edge with label a' or e.

Finally we let L_a be the RPQ $s(y + a' + e)s$. That is, L_a is equivalent to $sys + s(a' + e)s$. There are two possibilities for a repair to be such that (n_s, n_s) belongs to L_a: either (n_s, n_s) belongs to L' or there is an edge with label a' or e. Those two possibilities corresponds to cases (a) and (b) of the previous paragraph. ◀

Semantics based on the intersection of subset repairs. In order to obtain good complexity bounds for CQA in the DL context, Lembo et al. introduced a sound approximation of the CQA semantics based on the idea of evaluating queries over the intersection of all

[2] For a complete definition of G, (C1) and L, we should make such modifications for the labels d, r_{++}, r_{-+}, r_{+-}, r_{--}, t_0, f_0 and w.

subset repairs [32]. This approximation leads to tractability for inconsistency-tolerant query answering over some DLs of interest (e.g., for *DL-Lite*$_{\mathcal{A}}$). Although the repairs studied in the DL context are different to ours, it makes sense to study the pertinence of this semantics as a tool for establishing tractability results also in the graph database context.

Formally, let L be an RPQ and Γ a finite set of RPCs over Σ. We define $\cap$-CQA(L,Γ) as the problem of, given a graph database $G = (V, E)$ over Σ and a pair (u, v) of nodes in V, checking whether (u, v) belongs to the evaluation of L over the intersection of all $\subseteq$-repairs of G under Γ. (The intersection of graph databases $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ over Σ is the graph database $(V_1 \cap V_2, E_1 \cap E_2)$).

We prove below that the approximation semantics is not easier to evaluate than the original one. Therefore, in order to obtain tractability for our CQA problem it is necessary to look for restrictions that are proper to the scenario of graph databases.

▶ **Proposition 7.** *There exist a finite alphabet Σ, a non-recursive RPQ L and a finite set Γ of word constraints over Σ, such that $\cap$-CQA(L,Γ) is Π_2^{P}-hard.*

3.1 Tractable restrictions

Due to the inherent high complexity of our CQA problem under the subset repair semantics, it is important to look for meaningful restrictions leading to tractability. We provide two such restrictions in this section. The first one is based on the class of LAV RPCs, and the second one on the class of graph databases of bounded treewidth.

Restricting RPCs. In the relational case, the data complexity of CQA for unions of CQs under the class of *LAV tgds* (i.e., tgds with a single atom in the left-hand side [33]) is tractable. This actually holds for any of the three repair semantics [18]. The direct analogue of LAV tgds in our setting is the class of *LAV* RPCs, which are RPCs with a single symbol in the left-hand side. Formally, a LAV RPC over Σ is an RPC of the form $a \sqsubseteq L$, where $a \in \Sigma$ and L is an RPQ over Σ. We can leverage the techniques used to study CQA under LAV tgds to prove tractability in data complexity for our CQA problem under LAV RPCs.

▶ **Theorem 8.** *For each RPQ L and finite set Γ of LAV RPCs over the same alphabet Σ, it is the case that $\subseteq$-CQA(L,Γ) is in* NLogspace.

It is interesting to also consider RPCs based on the class of *GAV* tgds [33], that only allow for one symbol on the right-hand side. That is, a GAV RPC over Σ is of the form $L \sqsubseteq a$, for L an RPQ over Σ and $a \in \Sigma$. While this restriction improves the complexity of the CQA problem, it does not lead to tractability.

▶ **Proposition 9.**

1. *For each RPQ L and finite set Γ of GAV RPCs over the same alphabet Σ, it is the case that $\subseteq$-CQA(L,Γ) is in* coNP.
2. *There exist a finite alphabet Σ, a non-recursive RPQ L over Σ and a single GAV RPC of the form $ab \sqsubseteq c$, where $a, b, c \in \Sigma$, such that $\subseteq$-CQA(L,Γ) is* coNP*-complete.*

Note that in the setting of relational databases, the data complexity of consistent query answering for unions of conjunctive queries with respect to GAV constraints was also shown to be complete for the class coNP [18]. It is worth mentioning that we obtained the upper bound of Proposition 9 using techniques from the relational case.

The second part of the previous proposition shows that, in a sense, the tractability result for LAV RPCs in Theorem 8 is optimal: Allowing two-letter words on the left-hand side of RPCs leads to intractability, even if the right-hand side consists of a single letter.

Restricting graph databases. Our CQA problem under the subset repair semantics can be reformulated as a *monadic second-order logic* (MSO) evaluation problem over a relational representation of graph databases. This allows us to apply results establishing the tractability of MSO over structures of bounded treewidth [20]. From those results, we obtain tractability for the CQA problem over graph databases of bounded treewidth.

Formally, let $G = (V, E)$ be a graph database. A *tree decomposition* of G is a pair (T, λ), where T is a tree and $\lambda : T \to 2^V$ maps each node t in T to a nonempty set $\lambda(t)$ of nodes in V, that satisfies the following conditions:

- The set $\{t \in T \mid v \in \lambda(t)\}$ is a connected subset of T.
- For each edge $(u, a, v) \in E$, it is the case that $\{u, v\} \subseteq \lambda(t)$, for some $t \in T$.

The *width* of the tree decomposition (T, λ) is $\max\{|\lambda(t)| - 1 \mid t \in T\}$. The *treewidth* of G is the minimum width of a tree decomposition of G. For instance, the treewidth of G is one if and only if the underlying undirected graph of G is a tree.

We then obtain the following:

▶ **Theorem 10.** *Let L be an RPQ and Γ a set of RPCs. Then $\subseteq$-CQA(L,Γ) can be solved in linear time over graph databases of treewidth $\leq k$, for each $k \geq 1$.*

4 CQA under Superset and Symmetric Difference Repairs

We prove in this section that our CQA problem is undecidable under the semantics of $\supseteq$- and $\oplus$-repairs. This holds even if RPQs are non-recursive and RPCs are word constraints:

▶ **Theorem 11.** *Assume $\star \in \{\supseteq, \oplus\}$. There exist a finite alphabet Σ, a non-recursive RPQ L and a set Γ of word constraints over Σ, such that $\star$-CQA(L,Γ) is undecidable.*

In order to prove this theorem we establish a connection with the *implication problem* for RPCs [1, 26]. Recall that this is the problem of, given a finite set Γ of RPCs and an RPC $L_1 \sqsubseteq L_2$, checking whether $\Gamma \models L_1 \sqsubseteq L_2$, i.e., if $G \models \Gamma$ implies $G \models L_1 \sqsubseteq L_2$, for every graph database G. Grahne and Thomo proved this problem to be undecidable, even for word constraints, using a reduction from the *word rewriting* problem [26]. We develop nontrivial adaptations of such reduction to prove Theorem 11. The reason why we have to develop such adaptations is that there exist differences in nature between CQA and the implication problem for constraints. First, in the CQA problem we do not reason about all graph databases that satisfy the constraints (as in the case of the implication problem), but only about those that minimally differ from the original graph database. Note that in the special case of the superset semantics, this first problem does not apply. Indeed, given a graph database G, the intersection of the answers of a *monotone* query L over all the $\supseteq$-repairs is equal to the intersection of the answers of L over all the databases containing G and satisfying the constraints.

Second, we study the data complexity of the CQA problem, and, therefore, our goal is to prove undecidability of CQA for a *fixed* set of RPCs and a *fixed* RPQ. This is different to the case of the implication problem in which RPCs and RPQs define the input, and, therefore, cannot be fixed.

Proof of Theorem 11. We only prove the case when $\star$ is $\supseteq$. We start by recalling the basic notions of rewrite systems. Let Δ be a finite alphabet. A *semi-Thue rewrite system* $\mathcal{R}$ over Δ is a finite subset of $\Delta^* \times \Delta^*$. A rewrite system $\mathcal{R}$ induces a single-step reduction relation $\to_{\mathcal{R}}$ over Δ^* defined as:

$$\to_{\mathcal{R}} = \{(v, w) \mid v = xty, w = xuy, \text{ for some } (t, u) \in \mathcal{R} \text{ and } x, y \in \Delta^*\}.$$

We let $\rightarrow_{\mathcal{R}}^*$ be the reflexive transitive closure of $\rightarrow_{\mathcal{R}}$. The problem of testing whether a pair (u, v) belongs to $\rightarrow_{\mathcal{R}}^*$ is called the *rewrite problem* for $\mathcal{R}$. It is well known that there is a semi-Thue rewrite system $\mathcal{R}_0$ over Δ such that the rewrite problem for $\mathcal{R}_0$ is undecidable (see e.g., [8]).

We prove that $\supseteq$-CQA(L,Γ) is undecidable using a reduction from the rewrite problem for $\mathcal{R}_0$. Let $\hat{\Delta} = \Delta \cup \{\hat{a} \mid a \in \Delta\} \cup \{\$\}$, where the $\hat{a}$'s and \$ are fresh symbols. We define a set Γ of RPCs and an RPQ L over $\hat{\Delta}$, such that there is an algorithm that takes as input two words w_1 and w_2 over Δ and constructs a graph database G over $\hat{\Delta}$ with a node n_0 such that $w_1 \rightarrow_{\mathcal{R}_0}^* w_2$ iff $(n_0, n_0) \in \supseteq$-Cons(G,L,Γ). The set Γ consists of all RPCs of the form:

$$\begin{aligned} u &\sqsubseteq v, \\ a\$\hat{a} &\sqsubseteq \$, \end{aligned}$$

for $(u, v) \in \mathcal{R}_0$ and $a \in \Delta$. We define the RPQ L as the letter \$.

Let w_1 and w_2 be two words in Δ^*. We assume that

$$\begin{aligned} w_1 &= w_{11} w_{12} \dots w_{1k}, \\ w_2 &= w_{21} w_{22} \dots w_{2l}, \end{aligned}$$

where $w_{1i}, w_{2j} \in \Delta$ $(1 \leq i \leq k,\ 1 \leq j \leq l)$. We define the graph database $G = (V, E)$ over $\hat{\Delta}$ as follows. The set V of nodes of G is $\{n_i : 0 \leq i \leq k\} \cup \{m_i : 0 < i < l\}$. Let us define $m_0 = n_0$ and $m_l = n_k$. Then the set E of edges of G is defined as $E_1 \cup E_2 \cup E_3$, where E_1, E_2 and E_3 are as follows:

$$\begin{aligned} E_1 &= \{(n_{i-1}, a, n_i) \mid w_{1i} = a,\ 1 \leq i \leq k\}, \\ E_2 &= \{(m_i, \hat{a}, m_{(i-1)}) \mid w_{2i} = a,\ 1 \leq i \leq l\}, \\ E_3 &= \{(n_k, \$, n_k)\}. \end{aligned}$$

Notice that G consists of a path with label w_1 from n_0 to n_k and a path with label $\hat{w}_{2l}\hat{w}_{2(l-1)} \dots \hat{w}_{21}$ from n_k to n_0. The node n_k admits a loop with label \$.

The intuition is as follows. We start with the path with label w_1 in G_0. The idea is that if $w_1 \rightarrow_{\mathcal{R}_0}^* w_2$, then applying the constraints of the form $u \sqsubseteq v$ in Γ, we will construct a path with label w_2. The query is \$, and thus we have to check whether n_0 admits a loop with label \$ in every repair.

Intuitively, the presence of a path with label w_2 from n_0 to n_k is witnessed by a loop with label \$ at n_0. This is due to the presence of the constraints of the form $a\$\hat{a} \sqsubseteq \$$ in Γ. Indeed, suppose that $s_0 w_{21} s_1 w_{22} \dots s_l$ is a path with label w_2 from n_0 to n_k (where each s_i is a node, and thus $s_0 = n_0$ and $s_l = n_k$). Then, by induction on $0 \leq i \leq l$, using the constraints $a\$\hat{a} \sqsubseteq \$$ and the fact that n_k admits a loop with label \$, we can prove that there is an edge with label \$ from s_{l-i} to m_{l-i}. In particular, there is an edge with label \$ from s_0 to m_0. Since $s_0 = m_0 = n_0$, this implies that n_0 admits a loop with label \$.

We have to prove that $w_1 \rightarrow_{\mathcal{R}_0}^* w_2$ iff $(n_0, n_0) \in \supseteq$-Cons(G,L,Γ). The proof of the implication from left to right follows basically from the intuition that we gave in the previous two paragraphs. For the implication from right to left, suppose that $(n_0, n_0) \in \supseteq$-Cons(G,L,Γ). We have to prove that $w_1 \rightarrow_{\mathcal{R}_0}^* w_2$. The strategy is as follows.

(a) We construct a graph database H such that $G \subseteq H$ and $H \models \Gamma$. Moreover, if there is a path with label w_2 from n_0 to n_k in H, then $w_1 \rightarrow_{\mathcal{R}_0}^* w_2$.

(b) Since $H_0 \models \Gamma$, there is a repair H' of G under Γ such that $G \subseteq H' \subseteq H$. As the consistent answer of $L = \$$ contains the pair (n_0, n_0), this implies that n_0 admits a loop with label \$ in H'. In particular, n_0 admits a loop with label \$ in H.

(c) We prove that if n_0 admits a loop with label \$ in H, then there is a path with label w_2 from n_0 to n_k in H. Together with (a), this finishes the proof that $w_1 \to^* w_2$.

The construction of the graph database H uses ideas from the construction in the undecidability proof of [26]. In fact, H is an extension of the graph database used in such construction. Let k_0 be the maximum of $\{k, l\}$. We define the set of nodes V' of H as:

$$\{[u] \mid u \in \Delta^*, |u| \leq k_0\} \cup \{m_i : 0 < i < l\}.$$

We identify n_0 with $[\epsilon]$. For all $1 \leq i \leq k$, we identify n_i with $[w_{11} \dots w_{1i}]$. In particular, n_k is $[w_1]$. Note that this implies that the set V of nodes of G is a subset of V'.

Let E_1' be the relation:

$$E_1' = \{([u], a, [v]) \mid v \to^*_{\mathcal{R}_0} ua\}.$$

The graph $H_0 := (V', E_1')$ is precisely the graph constructed in the undecidability proof of [26]. We now define H as the graph $(V', E_1' \cup E_2' \cup E_3')$, where E_2' and E_3' are as follows:

$$\begin{aligned} E_2' &= \{(m_i, \hat{a}, m_{(i-1)}) \mid w_{2i} = a, 1 \leq i \leq l\}, \\ E_3' &= \{([u], \$, m_{(l-i)}) \mid \text{ there is a path with label } w_{2(l-i+1)} \dots w_{2l} \\ &\qquad \text{from } [u] \text{ to } n_k \text{ in } H_0\}. \end{aligned}$$

If $i = l$, by convention, we define $w_{2(l-i+1)} \dots w_{2l}$ as the empty word ϵ. Notice that $G \subseteq H$.

We skip the details of the proof that (a), (b) and (c) hold. An important part of this proof of is to show that if there is a path with label w_2 from n_0 to n_k in H, then $w_1 \to^*_{\mathcal{R}_0} w_2$. This can be obtained as an immediate consequence of Lemma 2 in the proof of Theorem 2 in [26]. Notice that although it might be impossible to construct the graph database H (since E_1' is defined in terms of the rewrite problem for $\mathcal{R}_0$), our proof only requires the existence of such graph database. ◀

4.1 Decidable restrictions

Since the CQA problem in this context is undecidable, it is crucial to look for decidable (and, ideally, tractable) restrictions of it. We provide three such restrictions in this section: The first one is based on the class of LAV RPCs, while the second one is based on the class of GAV RPCs. The third one is obtained by modifying RPC interpretation to be *from the origin*. It is worth noticing that the restriction to classes of graph databases of bounded treewidth, which leads to tractability under the semantics of subset repairs, is not useful in this context: The undecidability result in Theorem 11 holds even over graph databases of treewidth two.

Restriction to the class of LAV RPCs. As mentioned before, in the relational scenario the CQA problem for unions of CQs under LAV tgds is tractable, no matter which repair semantics is used. We already stated a similar result for CQA over graph databases under LAV RPCs and the subset repair semantics (Theorem 8). We can further extend those techniques to obtain tractability for our CQA problem under the semantics of $\oplus$-repairs.

▶ **Theorem 12.** *For each RPQ L and finite set Γ of LAV RPCs over the same alphabet Σ, it is the case that $\oplus$-CQA(L,Γ) is in* NLogspace.

The case of the $\supseteq$-repair semantics is different: We do not know whether LAV RPCs yield decidability in this context, but we prove next that at least they do not yield tractability in data complexity. This establishes a first difference in complexity between CQA under LAV tgds in the relational context and under LAV RPCs over graph databases.

▶ **Proposition 13.** *There exist a finite alphabet Σ, a non-recursive RPQ L, and a finite set Γ of LAV RPCs (without Kleene-star) over Σ, such that $\supseteq$-CQA(L,Γ) is* coNP*-hard.*

Notice that, unlike all previous lower bounds, the one in Proposition 13 is not stated in terms of the class of word constraints. In fact, the techniques developed for studying CQA under tgds in the relational case [18] can be adapted to show that under a set Γ of LAV word constraints the problem $\supseteq$-CQA(L,Γ) is tractable.

Restriction to the class of GAV RPCs. In the case of the symmetric difference semantics, it is easy to adapt the proof of Proposition 9 in order to show that when restricting to GAV RPCs, our CQA problem is coNP-hard. Note that in the relational case, a similar result holds.

▶ **Proposition 14.**

1. *For each RPQ L and finite set Γ of GAV RPCs over the same alphabet Σ, it is the case that $\oplus$-CQA(L,Γ) is in* coNP*.*
2. *There exist a finite alphabet Σ, a non-recursive RPQ L over Σ and a single GAV RPC of the form $ab \sqsubseteq c$, where $a, b, c \in \Sigma$, such that $\oplus$-CQA(L,Γ) is* coNP*-complete.*

In the case of the superset semantics, the restriction to GAV RPCs leads to tractability. Given a graph database G and a set of GAV RPCs Γ, using a classical chase argument, we can easily compute in Logspace the *unique* superset repair of G with respect to Γ. This is identical to what happens in the setting of relational databases.

▶ **Proposition 15.** For each RPQ L and finite set Γ of GAV RPCs over the same alphabet Σ, it is the case that $\supseteq$-CQA*(L,Γ)* is in NLogspace.

Modifying the interpretation of RPCs. In the original proposal of Abiteboul and Vianu, RPQs, and therefore RPCs, are only evaluated from a particular node called the *origin*. This is motivated by their application over semi-structured data. Formally, let o be a fixed node id that we identify as the origin. A graph database $G = (V, E)$, satisfies $L_1 \sqsubseteq L_2$ *from the origin*, denoted $G \models_o L_1 \sqsubseteq L_2$, if and only if the origin o belongs to V and $\{v \in V \mid (o, v) \in L_1(G)\} \subseteq \{v \in V \mid (o, v) \in L_2(G)\}$. If Γ is a set of RPCs, we write $G \models_o \Gamma$ if $G \models_o L_1 \sqsubseteq L_2$, for each RPC $L_1 \sqsubseteq L_2$ in Γ.

We can now modify the definition of repairs and consistent answers with respect to the restricted $\models_o$ interpretation of RPCs. Assume $\star \in \{\subseteq, \supseteq, \oplus\}$. An $\{o, \star\}$*-repair* of a graph database G under a set of RPCs Γ is defined exactly as an $\star$-repair of G under Γ, except that now the satisfaction of the RPCs in Γ is defined with respect to the relation $\models_o$. (For safety reasons, we assume that G contains the origin o in this case). For instance, G' is a $\{o, \supseteq\}$-repair of G under Γ, if (1) $G \subseteq G'$, (2) $G' \models_o \Gamma$, and (3) there is no graph database G'' such that $G \subseteq G'' \subsetneq G'$ and $G'' \models_o \Gamma$.

Furthermore, if L is an RPQ and Γ is a finite set of RPCs, we define $\{o, \star\}$-CQA(L,Γ) as the problem of, given a graph database $G = (V, E)$ and a pair (u, v) of nodes in V, checking whether (u, v) is an $\{o, \star\}$*-consistent answer of L over G under* Γ, i.e., if $(u, v) \in L(G')$, for each $\{o, \star\}$-repair G' of G under Γ.

By interpreting RPCs under the relation $\models_o$, we obtain decidability for our CQA problem under the semantics of $\supseteq$-repairs. We do not know whether this can be extended to the semantics of $\oplus$-repairs. Notice the difference with the restriction to LAV RPCs we studied before: For the latter we could only obtain decidability under the $\oplus$-repairs semantics.

▶ **Theorem 16.** *For each RPQ L and finite set Γ of RPCs over the same alphabet Σ, it is the case that $\{o, \supseteq\}$-CQA(L,Γ) is in* coNP.

The difference here is that the implication problem for RPCs becomes decidable if RPCs are interpreted under the relation $\models_o$ [1]. We adapt the techniques used to prove this fact in order to obtain Theorem 16.

We do not know whether the bound in Theorem 16 is tight. Interestingly, we can show that a slight extension on the query language leads to intractability. We reduce from the problem of 3-colorability. Assume we are given an undirected graph H. From H we construct a graph database $G = (V, E)$, such that (1) V corresponds to the set of nodes of H plus the origin o, and (2) E contains edges (o, a, v), for each node v in H, and (u, e, v) and (v, e, u), for each edge $\{u, v\}$ in H. Assume that Γ consists of the single constraint $a \sqsubseteq c_1 \cup c_2 \cup c_3$. Intuitively, this tells us that a $\supseteq$-repair of G contains, for each node $v \neq o$, one, and only one, edge of the form (o, c_i, v), for $1 \leq i \leq 3$. This edge represents the color assigned to node v by an assignment of three colors to the nodes of H.

It is not hard to prove that H is 3-colorable if and only if there exists a $\supseteq$-repair G' of G such that no two nodes linked by an edge labeled e in G' are assigned the same color. This is equivalent to checking that it is not the case that there are paths labeled $c_i e$ and c_i in G', for $1 \leq i \leq 3$, that start in the origin o and reach the same node v. While this cannot be expressed as an RPQ, it can be easily expressed as an RPQ with *inverses* [14]. The query is $Q := \bigcup_{1 \leq i \leq 3} c_i e c_i^-$, where c_i^- represents a backward traversal of an edge labeled c_i. We then have that H is 3-colorable if and only if (o, o) is not an $\{o, \supseteq\}$-consistent answer of Q over G under Γ. This query can also be expressed as a *conjunction* of two RPQs [13].

▶ Remark. The restriction presented in this section does not help reducing the complexity under the semantics of $\subseteq$-repairs. In fact, it can be proved that all lower bounds obtained in Section 3 continue to hold for the semantics of $\{o, \subseteq\}$-repairs. This is done in the appendix.

5 Comparison with CQA in the relational context

We compare our results with previous results on CQA obtained in the relational context. We assume familiarity with relational schemas and CQs. Tuple-generating dependencies, or tgds, define one of the most important classes of relational database constraints. They subsume several other classes of interest, such as inclusion dependencies. In addition, they have important applications in data integration, data exchange and ontological query answering [33, 23, 10]. Formally, a tgd over a relational schema σ is a formula of the form $\forall \bar{x}(\phi(\bar{x}) \rightarrow \psi(\bar{x}))$, where both $\phi(\bar{x})$ and $\psi(\bar{x})$ are CQs over σ and each variable in $\bar{x}$ is mentioned in $\phi(\bar{x})$. A relational database D over σ satisfies this tgd if $D \models \phi(\bar{a})$ implies $D \models \psi(\bar{a})$, for each tuple $\bar{a}$ of elements in D of the same length than $\bar{x}$.

As mentioned in the introduction, each word constraint can be naturally seen as a tgd over the standard relational representation of graph databases. However, lower bounds for CQA under tgds in the relational setting, such as the ones obtained by ten Cate et al. [18], cannot be used to obtain lower bounds for CQA under word constraints (or even RPCs) in the graph database context. This is because word constraints correspond to a restricted class of tgds defined by *chain* CQs (which we call *chain tgds*). However, none of the lower bounds developed for the data complexity of CQA under tgds applies to this class.

We formalise the class of chain tgds as follows. Let σ be a relational schema that contains only binary relation symbols. A chain CQ over σ is a CQ of the form

$$\phi(x, y) \;:=\; \exists u_1 u_2 \dots u_{m-1} \big(R_1(x, u_1) \wedge R_2(u_1, u_2) \wedge \dots \wedge R_{m-1}(u_{m-1}, y)\big),$$

where each R_i is a relation symbol in σ [21]. That is, the underlying *directed* graph of a chain CQ is a path. A chain tgd is one of the form $\forall x \forall y(\phi(x,y) \rightarrow \psi(x,y))$, where both $\phi(x,y)$ and $\psi(x,y)$ are chain CQs. It is easy to see that each word constraint can be represented as a chain tgd over the standard relational representation of graph databases (in which, for each $a \in \Sigma$, there is a binary relation symbol E_a that contains all pairs of nodes that are linked by an a-labeled edge in the graph database). Conversely, each chain tgd is the representation of a word constraint.

This allows us to use our proof techniques to obtain lower bounds for CQA under the restricted class of chain tgds in the relational context:

▶ **Proposition 17.**

1. *Consider a semantics based on subset repairs of relational databases. There is a relational schema σ that contains only binary relation symbols, a finite set $\mathcal{T}$ of chain tgds and a union Q of CQs over σ, such that the problem of evaluating certain answers for Q under $\mathcal{T}$ is Π_2^{P}-hard. The same holds for the semantics based on the intersection of all subset repairs.*
2. *Consider a semantics based on superset repairs of relational databases. There is a relational schema σ that contains only binary relation symbols, a finite set $\mathcal{T}$ of chain tgds and a union Q of CQs over σ, such that evaluating certain answers for Q under $\mathcal{T}$ is undecidable. The same holds for the semantics of symmetric difference repairs.*

6 Conclusions and Future Work

In this work we initiated the study of CQA over graph databases. The data complexity of the problem is in general undecidable or highly intractable, which motivated our search for decidable, and even tractable restrictions. In the case of subset repair semantics we obtain tractability by either restricting to the class of LAV RPCs or to the class of graph databases of bounded treewidth. The class of LAV RPCs also yields tractability for our problem under the semantics of $\oplus$-repairs. On the other hand, for the semantics of superset repairs we obtain decidability if RPCs are interpreted from the origin or if we restrict to the class of GAV RPCs.

Several questions regarding CQA under the semantics of $\supseteq$- and $\oplus$-repairs remain open. For instance, we do not know if CQA under $\supseteq$-repairs is decidable when RPCs are in LAV form. Neither do we know whether CQA under $\oplus$-repairs is decidable when RPCs are interpreted from the origin. We plan to study this in the future.

It would also be interesting to look for different kinds restrictions that yield decidability for our CQA problem. For instance, in the relational scenario it is possible to obtain tractability in data complexity for CQA under $\supseteq$- and $\oplus$-repair semantics if the set Γ of tgds is *weakly acyclic* [18]. The reason is that in this case there is a polynomial that bounds the size of each repair of a database D under Γ. We would like to develop a meaningful adaptation of this notion to the scenario of RPCs in search for similar positive results. This is more difficult than in the relational case, however, since the notion of acyclicity will have to consider how regular expressions interact with each other. Another way in which positive results for our CQA problem under $\supseteq$- and $\oplus$-repair semantics could be obtained, is by restricting to classes of RPCs for which the implication problem is decidable. This includes, for instance, classes of word constraints for which the associated rewrite problem is decidable [26].

While our work concentrates on queries defined as RPQs, all upper bounds presented in the paper continue to hold in the extended scenario in which queries are defined as *conjunctive* RPQs (CRPQs) [13]. In turn, CRPQs might lead to an expressive class of *conjunctive* RPCs,

which are expressions of the form $\phi(\bar{x}) \sqsubseteq \psi(\bar{x})$, for CRPQs ϕ and ψ. It is interesting to study how the positive results presented in this paper can be extended to be applied over this class of constraints.

Acknowledgements. We are grateful to Aidan Hogan and Leonid Libkin for their helpful comments in earlier versions of the paper. Carsten Lutz and Meghyn Bienvenu also provided us with important insights on the nature of DL repairs. Barceló and Fontaine are funded by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004. Fontaine is also funded by Fondecyt postdoctoral grant 3130491.

References

1 S. Abiteboul, V. Vianu. Regular path queries with constraints. *JCSS*, 58(3), pages 428–452, 1999.
2 M. Arenas, L. E. Bertossi, Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS* 1999, pages 68–79.
3 M. Arenas, W. Fan, L. Libkin. On the complexity of verifying consistency of XML specifications. *SIAM J. Comput.* 38(3), pages 841–880, 2008.
4 F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.). The description logic handbook: Theory, implementation, and applications. *Cambridge University Press*, 2003.
5 P. Barceló. Querying graph databases. In *PODS* 2013, pages 175–188.
6 M. Bienvenu. On the complexity of consistent query answering in the presence of simple ontologies. In *AAAI* 2012.
7 M. Bienvenu, R. Rosati. Tractable approximations of consistent query answering for robust ontology-based data access. In *IJCAI* 2013.
8 R. Book, F. Otto String. *Rewriting Systems.* Springer Verlag, 1993.
9 P. Buneman, W. Fan, S. Weinstein. Path constraints in semi-structured databases. *J. Comput. Syst. Sci.* 61(2), pages 146–193, 2000.
10 A. Calì, G. Gottlob, M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.* (JAIR) 48, pages 115–174, 2013.
11 Andrea Calì, D. Lembo, R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS* 2003, pages 260–271.
12 D. Calvanese, G. de Giacomo, M. Lenzerini. Structured objects: Modeling and reasoning. In *DOOD* 1995, pages 229–246.
13 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR* 2000, pages 176–185.
14 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Rewriting of regular expressions and regular path queries. *JCSS*, 64(3), pages 443–465, 2002.
15 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Reasoning on regular path queries. *SIGMOD Record* 32(4), pages 83–92, 2003.
16 D. Calvanese, G. de Giacomo, M. Lenzerini. Conjunctive query containment and answering under description logic constraints. *ACM Trans. Comput. Log.* 9(3), 2008.
17 D. Calvanese, M. Ortiz, M. Simkus. Containment of regular path queries under description logic constraints. In *IJCAI* 2011, pages 805–812.
18 B. ten Cate, G. Fontaine, Ph. G. Kolaitis. On the data complexity of consistent query answering. In *ICDT* 2012, pages 22–33.
19 J. Chomicki, J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.* 197(1–2), pages 90–121, 2005.

20 B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite hraphs. *Inf. Comput.* 85(1), pages 12–75, 1990.

21 G. Dong. On datalog linearization of chain queries. *Theoretical Studies in Computer Science* 1992, pages 181–206.

22 W. Fan, J. Siméon: Integrity constraints for XML. *J. Comput. Syst. Sci.*, 66(1), pages 254–291, 2003.

23 R. Fagin, Ph. G. Kolaitis, R. J. Miller, L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.* 336(1), pages 89–124, 2005.

24 G. Fontaine. Why is it hard to obtain a dichotomy for consistent query answering? In *LICS* 2013, pages 550–559.

25 A. Fuxman, R. J. Miller. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.* 73(4), pages 610–635, 2007.

26 G. Grahne, A. Thomo. Query containment and rewriting using views for regular path queries under constraints. In *PODS* 2003, pages 111–122.

27 A. Hogan, A. Harth, A. Passant, S. Decker, A. Polleres. Weaving the pedantic web. In *LDOW* 2010.

28 Ph. G. Kolaitis, Enela Pema. A dichotomy in the complexity of consistent query answering for queries with two atoms. *Inf. Process. Lett.* 112(3), pages 77–85, 2012.

29 A. Lopatenko, L. E. Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *ICDT* 2007, pages 179–193.

30 G. Lausen, M. Meier, M. Schmidt. SPARQLing constraints for RDF. In *EDBT* 2008, pages 499–509.

31 D. Lembo, M. Ruzzi. Consistent query answering over description logic ontologies. In *DL* 2007.

32 D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, D. Fabio Savo. Inconsistency-tolerant semantics for description logics. In *RR* 2010, pages 103–117.

33 M. Lenzerini. Data integration: A theoretical perspective. In *PODS* 2002, pages 233–246.

34 Th. Lukasiewicz, M. V. Martinez, G. I. Simari. Complexity of inconsistency-tolerant query answering in Datalog+/-. In *OTM Conferences* 2013, pages 488–500.

35 R. Rosati. On the complexity of dealing with inconsistency in description logic ontologies. In *IJCAI* 2011, pages 1057–1062.

36 K.-D. Schewe, B. Thalheim, J. W. Schmidt, I. Wetzel. Integrity enforcement in object-oriented databases. In *FMLDO* 1992, pages 174–195.

37 J. Wijsen. Condensed representation of database repairs for consistent query answering. In *ICDT* 2003, pages 375–390.

38 J. Wijsen. Certain conjunctive query answering in first-order logic. *ACM Trans. Database Syst.* 37(2), 9, 2012.

39 P. T. Wood. Query languages for graph databases. *SIGMOD Record* 41(1), pages 50–60, 2012.

40 Y. Yuan, G. Wang, L. Chen, H. Wang. Efficient subgraph similarity search on large probabilistic graph databases. In *PVLDB* 5(9), pages 800–811, 2012.