Journal of Artificial Intelligence Research 76 (2023) 721-759 Submitted 06/2022; published 03/2023

On the Complexity of Finding Set Repairs for Data-Graphs

Sergio Abriola SABRIOLA@DC.UBA.AR
Maria Vanina Martinez MVMARTINEZ@DC.UBA.AR
Nina Pardal NPARDAL@DC.UBA.AR
ICC CONICET, Universidad de Buenos Aires

Santiago Cifuentes SCIFUENTES@DC.UBA.AR
Edwin Pin Baque EPIN@DC.UBA.AR
Departamento de Computación, Universidad de Buenos Aires

Abstract

In the deeply interconnected world we live in, pieces of information link domains all around us. As graph databases embrace effectively relationships among data and allow processing and querying these connections efficiently, they are rapidly becoming a popular platform for storage that supports a wide range of domains and applications. As in the relational case, it is expected that data preserves a set of integrity constraints that define the semantic structure of the world it represents. When a database does not satisfy its integrity constraints, a possible approach is to search for a 'similar' database that does satisfy the constraints, also known as a repair. In this work, we study the problem of computing subset and superset repairs for graph databases with data values using a notion of consistency based on having a set of Reg-GXPath expressions as integrity constraints. We show that for positive fragments of Reg-GXPath these problems admit a polynomial-time algorithm, while the full expressive power of the language renders them intractable.

1. Introduction

The availability of high volumes of interconnected data allows the possibility of developing applications that go well beyond semantic indexing and search and involve advance reasoning tasks on top of existing data. Alternative data models such as graph databases are becoming popular as they allow to effectively represent and access this type of data. Graph databases are specially useful for applications where the topology of the data is as important as the data itself, such as social networks analysis (Fan, 2012), data provenance (Anand et al., 2010), and the Semantic Web (Arenas & Pérez, 2011). The structure of the database is commonly queried through navigational languages such as *regular path queries* or RPQs (Barceló, 2013) that can capture pairs of nodes connected by some specific kind of path. These query languages can be extended to add more expressiveness, while usually adding extra complexity in the evaluation as well. For example, C2RPQs are a natural extension of RPQs defined by adding to the language the capability of traversing edges backwards and closing the expressions under conjunction (similar to relational CQs).

RPQs and its most common extensions, C2RPQs and NREs (Barceló et al., 2012), can only act upon the edges of the graph, leaving behind any possible interaction with data values in the nodes. This led to the design of query languages for the case of data-graphs

©2023 AI Access Foundation. All rights reserved.

(i.e. graph databases where data lies both in the paths and in the nodes themselves), such as REMs and Reg-GXPath (Libkin et al., 2016).

Advanced computational reasoning tasks usually require the management of inconsistent information. Reasoning with or in the presence of inconsistent knowledge bases has been the focus of a vast amount of research in Artificial Intelligence and theory of relational databases for over 30 years. However, in the last years, the area has flourished focusing on logical knowledge bases and ontologies (Bienvenu et al., 2019; Lembo et al., 2010; Lukasiewicz et al., 2022).

As in the relational case, consistency is related to the notion of *integrity constraints* that express some of the semantic structure the data intends to represent. In the context of graph databases, these constraints can be expressed through *path constraints* (Abiteboul & Vianu, 1999; Buneman et al., 2000). When a database does not satisfy its integrity constraints, a possible approach is to compute a 'similar' database that does satisfy the constraints. In the literature, such a database is called a *repair* (Arenas et al., 1999), and in order to define it properly one has to determine the precise meaning of 'similar'.

Consider for example the data-graph in Figure 1: here the nodes represent people, and the edges model different kinds of relationships between them, such as brotherhood or parenthood. Observe that in this data-graph, every pair of nodes (x, y) connected with a SIBLING_OF edge directed from x to y is also connected with a SIBLING_OF edge directed from y to x. This property seems reasonable, since the brotherhood relation is symmetric. In addition, the nodes (MAURO, JULIETA) are connected through a NIBLING_OF edge directed to JULIETA, which it is also a property we would expect to find considering that MAURO's dad, DIEGO, is the brother of JULIETA.

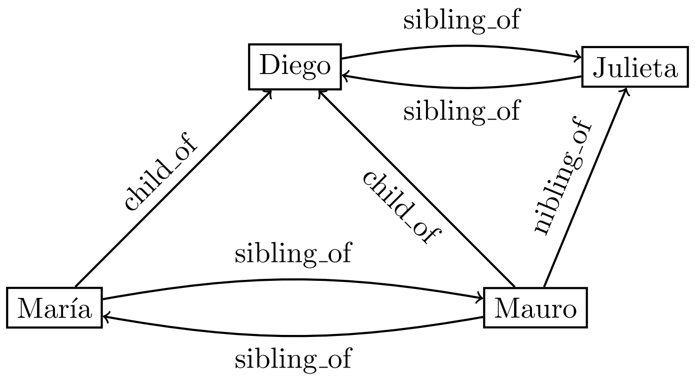

Figure 1: An example of a data-graph where the nodes represent people and the edges family relationships.

The structure that this particular data-graph has to preserve in order to properly capture our notions of SIBLING_OF and NIBLING_OF can be stated through path expressions that capture those pairs of nodes that satisfy the semantic constraints. In particular, the pair (MARÍA, JULIETA) does not satisfy the semantics of the NIBLING_OF relation, since DIEGO is also MARÍA's parent but there is no NIBLING_OF edge from MARÍA to JULIETA. This could be 'fixed' by adding the NIBLING_OF edge from MARIA to JULIETA, or by rather deleting the edge child_of directed from María to Diego. Naturally, we want to preserve as much data as possible from our original data-graph.

There are different notions of repairs in the literature, among others, set-based repairs (Ten Cate et al., 2012), attribute-based repairs (Wijsen, 2002), and cardinality based repairs (Lopatenko & Bertossi, 2007). In this work, we study two restrictions of the problem of finding a set-based repair G' of a graph database G under a set of Reg-GXPath expressions R considered as *path constraints*: when G' is a subgraph of G and when G' is a super-graph of G. These kinds of repairs are usually called *subset* and *superset* repairs, respectively (Barceló & Fontaine, 2017; Ten Cate et al., 2012). Since repairs may not be unique, it is possible to impose an order over the set of repairs and look for an 'optimum' repair over such order (Flesca et al., 2007; Staworko et al., 2012).

Reg-GXPath is a query language developed for graph databases with data values in the nodes (Libkin et al., 2016) largely inspired by *XPath* (Benedikt & Koch, 2009), a language for traversing data trees (i.e. XML documents). Reg-GXPath's expressiveness is quite understood in relation to other query languages (Libkin et al., 2016), but there are still some open problems related to it, such as query containment and query equivalence. Two other candidates for this work were the RQMs and RDQs (Libkin & Vrgoč, 2012), and our notion of consistency can be defined based upon those languages. RQMs are more expressive than Reg-GXPath regarding its capabilities of interacting with data values, and its evaluation belongs to PSpace, while RDQs are less expressive than Reg-GXPath (taking into account both topological and data aspects), and its complexity is lower. The main advantage of Reg-GXPath in comparison with other query languages is its great balance between expressiveness and complexity of evaluation (being polynomial on the graph and query size), as well as the fact that it allows to interact with the topology of the graph and, at the same time, to reason about the data that it contains.

Regarding its topological component, Reg-GXPath is as expressive as most of the languages that can be evaluated in polynomial time, such as 2RPQs and NREs (Barceló, 2013), allowing the description of regular and nested paths. It also allows negation, which is a powerful and uncommon tool among navigational languages. As we can see from the results, the inclusion of such feature makes reasoning problems way harder (see Table 1). Regarding the interaction with data values, most query languages usually allow for some form of data tests (as RDQ does), but Reg-GXPath also admits data comparison between nodes. Nonetheless, features from RQMs, such as memory and variable usage and replacement in the formula (i.e. the capability of instantiating variables in a path formula with data values found alongside the navigation), cannot be simulated in this context.

The specific contributions of this work are the following:

- We define a model for graph databases with data values and introduce a notion of consistency over this model, based on a set of Reg-GXPath expressions that capture a significant group of common integrity constraints that appear in the literature.
- Given a graph database G and a set of constraints R, we study the problem of computing a subset repair (respectively, a superset repair) G' of G.
- We show that depending on the expressiveness of the restrictions (whether they use the full power of Reg-GXPath or only the fragment without negation known as

Reg-GXPath^{pos}), these problems admit a polynomial-time algorithm or rather turn out to be intractable, even undecidable.

The rest of this work is organized as follows. In Section 2 we introduce the necessary preliminaries and notation for the syntax and semantics for our data-graph model as well as the definitions of consistency and different types of repairs. We also show, by means of examples, that the proposed language can capture a group of integrity constraints that are common in the database literature. In Section 3 we study the complexity of the repair computing problem for both subset and superset repairs. Finally, in Section 4 and Section 5 we discuss related work and conclusions, respectively.

2. Definitions

Let us fix a finite set of edge labels Σ_e and a countable (either finite or infinite enumerable) set of data values Σ_n, sometimes referred to as data labels, both of which we assume non-empty, and such that $\Sigma_e \cap \Sigma_n = \emptyset$. A *data-graph* G is a tuple (V, L_e, D) where V is a finite set of nodes, L_e is a mapping from $V \times V$ to $\mathcal{P}(\Sigma_e)$ defining the edges of the graph, and D is a function mapping the nodes from V to data values from Σ_n. Given a data-graph $G = (V, L_e, D)$, we will sometimes consider the data-graph $G_{V'}$, with $V' \subseteq V$, defined as $G_{V'} = (V', L|_{V' \times V'}, D|'_V)$. Intuitively, $G_{V'}$ is the subdata-graph of G induced by the set V'.

Path expressions of Reg-GXPath are given by:

$$\alpha, \beta = \epsilon \mid \_ \mid \text{A} \mid \text{A}^- \mid [\varphi] \mid \alpha \,.\, \beta \mid \alpha \cup \beta \mid \alpha \cap \beta \mid \alpha^* \mid \overline{\alpha} \mid \alpha^{n,m}$$

where A iterates over every label of Σ_e, n and m over all possible natural numbers and φ is a *node expression* defined by the following grammar:

$$\varphi, \psi = \neg\varphi \mid \varphi \wedge \psi \mid \langle \alpha \rangle \mid \text{c}^= \mid \text{c}^{\neq} \mid \langle \alpha = \beta \rangle \mid \langle \alpha \neq \beta \rangle \mid \varphi \vee \psi$$

where α and β are path expressions (i.e. path and node expressions are defined by mutual recursion) and c iterates over Σ_n. If we only allow the Kleene star to be applied to labels and their inverses (the production A$^-$), then we obtain a subset of Reg-GXPath called Core-GXPath. The semantics of these languages are defined in (Libkin et al., 2016) in a similar fashion as the usual regular languages for navigating graphs (Barceló, 2013), while adding some extra capabilities such as the complement of a path expression $\overline{\alpha}$ and data tests. The $\langle \alpha \rangle$ operator is the usual one for *nested regular expressions* (NREs) used in (Barceló et al., 2012): intuitively, for a node expression $\langle \alpha \rangle$ to be satisfied in a node v of a data-graph G there must exist in G a path α starting at v. Given a data-graph $G = (V, L, D)$, the semantics of Reg-GXPath expressions are:

$$[\![\epsilon]\!]_G = \{(v, v) \mid v \in V\}$$
$$[\![\_]\!]_G = \{(v, w) \mid v, w \in V, L(v, w) \neq \emptyset\}$$
$$[\![\text{A}]\!]_G = \{(v, w) \mid \text{A} \in L(v, w)\}$$
$$[\![\text{A}^-]\!]_G = \{(w, v) \mid \text{A} \in L(v, w)\}$$
$$[\![\alpha^*]\!]_G = \text{the reflexive transitive closure of } [\![\alpha]\!]_G$$

$$
\begin{aligned}
&[\![\alpha.\beta]\!]_G = [\![\alpha]\!]_G[\![\beta]\!]_G\\
&[\![\alpha \cup \beta]\!]_G = [\![\alpha]\!]_G \cup [\![\beta]\!]_G\\
&[\![\alpha \cap \beta]\!]_G = [\![\alpha]\!]_G \cap [\![\beta]\!]_G\\
&[\![\overline{\alpha}]\!]_G = V \times V \setminus [\![\alpha]\!]_G\\
&[\![[\varphi]]\!]_G = \{(v,v) \mid v \in [\![\varphi]\!]_G\}\\
&[\![\alpha^{n,m}]\!]_G = \bigcup_{k=n}^{m} ([\![\alpha]\!]_G)^k\\
&[\![\langle\alpha\rangle]\!]_G = \{v \mid \exists w \in V,\, (v,w) \in [\![\alpha]\!]_G\}\\
&[\![\neg\varphi]\!]_G = V \setminus [\![\varphi]\!]_G\\
&[\![\varphi \wedge \psi]\!]_G = [\![\varphi]\!]_G \cap [\![\psi]\!]_G\\
&[\![\varphi \vee \psi]\!]_G = [\![\varphi]\!]_G \cup [\![\psi]\!]_G\\
&[\![\mathrm{c}^{=}]\!]_G = \{v \in V \mid D(v) = \mathrm{c}\}\\
&[\![\mathrm{c}^{\neq}]\!]_G = \{v \in V \mid D(v) \neq \mathrm{c}\}\\
&[\![\langle\alpha = \beta\rangle]\!]_G = \{v \in V \mid \exists v', v'' \in V,\, (v,v') \in [\![\alpha]\!]_G,\, (v,v'') \in [\![\beta]\!]_G,\, D(v') = D(v'')\}\\
&[\![\langle\alpha \neq \beta\rangle]\!]_G = \{v \in V \mid \exists v', v'' \in V,\, (v,v') \in [\![\alpha]\!]_G,\, (v,v'') \in [\![\beta]\!]_G,\, D(v') \neq D(v'')\}
\end{aligned}
$$

Notice that we are using the standard composition of relations in the definition of $[\![\alpha.\beta]\!]_G$ and $[\![\alpha^{n,m}]\!]_G$. More precisely, given two binary relations R_1, R_2 over V_G, we define R_1R_2 as $\{(x,z) : \exists y \in V_G, (x,y) \in R_1 \text{ and } (y,z) \in R_2\}$, and R^k as $R^k \equiv R^{k-1}R$ for $k > 0$, where $R^0 = [\![\epsilon]\!]_G = \{(x,x) : x \in V_G\}$.

As in (Libkin et al., 2016), we denote the sizes of path expressions α and node expressions φ by $|\alpha|$ and $|\varphi|$, respectively, and define them to be the number of symbols in α and φ, respectively (equivalently, we could define them as the sizes of the parse trees of those expressions). The size of a counter is the number of bits representing it, so $|\alpha^{n,m}| = |\alpha| + \log n + \log m$.

We denote by $\alpha \Rrightarrow \beta$ the path expression $\beta \cup \overline{\alpha}$, and by $\varphi \Rightarrow \psi$ the node expression $\psi \vee \neg\varphi$. The expression $\alpha \Rrightarrow \beta$ works in an analogous way as the logical implication operator: a pair of nodes (v,w) satisfy $\alpha \Rrightarrow \beta$ over a data-graph G if $(v,w) \in [\![\beta]\!]_G$ whenever $(v,w) \in [\![\alpha]\!]_G$. Or, equivalently, if (v,w) belongs to $[\![\beta \cup \overline{\alpha}]\!]_G$.

We also note an expression $\textsc{a}$ as $\downarrow_{\textsc{a}}$ in order to easily distinguish the 'path' fragment of the expressions. For example, the expression $\textsc{child\_of}\ [\textsc{Maria}^{=}]\ \textsc{sister\_of}$, which would capture pair of nodes (v,w) such that there exists another node z with $D(z) = \textsc{Maria}$ and edges vz and zw with labels $\textsc{child\_of}$ and $\textsc{sister\_of}$ respectively, will be noted as $\downarrow_{\textsc{child\_of}}\ [\textsc{Maria}^{=}]\ \downarrow_{\textsc{sister\_of}}$. Also, we will sometimes denote an expression $\textsc{a}^{-}$ as $\downarrow_{\textsc{a}}^{-}$.

Naturally, the expression $\alpha \cap \beta$ can be rewritten as $\overline{\overline{\alpha} \cup \overline{\beta}}$ while preserving the semantics, and something similar happens with the operators $\wedge$ and $\vee$ for the case of node expressions using the $\neg$ operator. However, we still include the definitions of all these operators for this grammar, since we will be interested in the sequel in a fragment of Reg-GXPath called Reg-GXPath^{pos}. This fragment shares the same grammar with the exception of the $\overline{\alpha}$ and $\neg\varphi$ expressions. Thus, we will not be able to 'simulate' the $\cap$ operator in Reg-GXPath^{pos} unless it is present in the original Reg-GXPath grammar.

Example 1. Using Core-GXPath, we could write an expression capturing every "transitive friend" in a social network using $\alpha = \downarrow^{+}_{\text{FRIEND\_OF}}$. This is a common example of regular expressions, and it does not require most of Core-GXPath capacities. We can enhance our example by asking for an expression that captures those pairs of nodes who are "transitive friends" and also follow some other specific node with data value x.

$$\alpha = [\langle \downarrow_{\text{FOLLOWS}} [x^{=}] \rangle] \downarrow^{+}_{\text{FRIEND\_OF}} [\langle \downarrow_{\text{FOLLOWS}} [x^{=}] \rangle]$$

In this case, we are already using the data tests and the *nesting* operator.

Another interesting difference between most regular languages and Core-GXPath is that we can easily obtain the complement of a path expression. Since regular languages are closed under complement, most navigation languages can express the complement of any expression, but this is not generally built into the grammar.

Example 2. Given a social database where we have family links, we can capture those nodes that do not have any ancestor node with the same name. We do so by using the following node expression:

$$\varphi = \neg \langle \epsilon = \downarrow^{+}_{\text{FATHER\_OF}} \rangle$$

Note that this cannot be expressed by most navigational languages, since they do not have a way of comparing data values.

Consistency Given a specific database, we may want a node or path expression to capture all the nodes from the data-graph, since it could represent some structure we expect to find in our data. This kind of Core-GXPath or Reg-GXPath expression would work as an *integrity constraint* defining semantic relations among our data. A data-graph in which an expression α, used as a constraint, does not capture all nodes may be called *inconsistent* with respect to α. In general, we define the notion of consistency in the following way:

Definition 3 (Consistency)**.** Let G be a data-graph and $R = P \cup N$ a finite set of restrictions, where P and N consist of path and node expressions, respectively. We say that (G, R) is **consistent**, noted as $G \models R$, if the following conditions hold:

- $\forall x \in V_G$ and $\varphi \in N$, we have that $x \in [\![\varphi]\!]$
- $\forall x, y \in V_G$ and $\alpha \in P$, we have that $(x, y) \in [\![\alpha]\!]$

Otherwise we say that G is inconsistent w.r.t. R.

In the remainder of this work we will simply say that G is (in)consistent, whenever R is clear from the context.

Example 4. Consider a film database (see Figure 2), where some nodes represent people from the film industry such as actors or directors, and others represent movies or documentaries. If we want to make a cut from that graph that preserves only actors who have worked with Philip Seymour Hoffman in a film from Paul Thomas Anderson, then we want the following formula to be satisfied:

$$\varphi = \langle \downarrow_{\text{TYPE}} [actor^{=}] \rangle \Rightarrow \langle \downarrow_{\text{ACTS\_IN}} \langle \downarrow_{\text{DIRECTED\_BY}} [Anderson^{=}] \rangle \downarrow^{-}_{\text{ACTS\_IN}} [Hoffman^{=}] \rangle$$

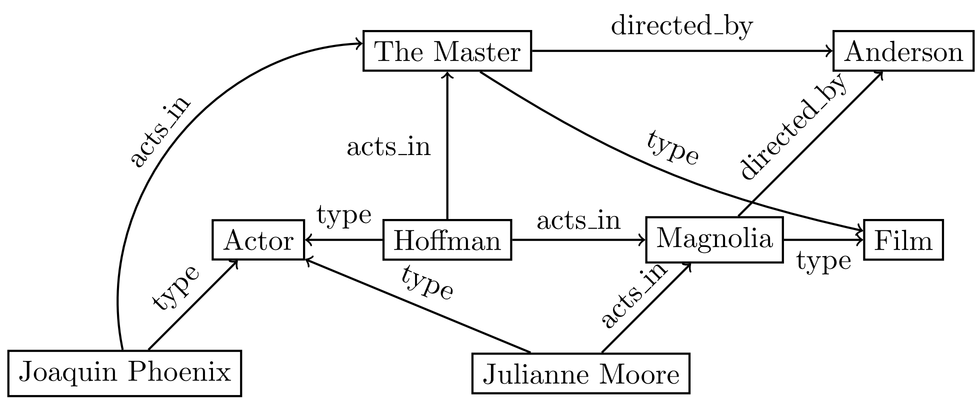

Figure 2: In this data-graph φ is satisfied, since both Phoenix and Moore have worked with Hoffman in a film from Anderson (respectively through 'The Master' and 'Magnolia'). Note that the restriction also applies to Hoffman, so it is required that he participates in any film from Anderson in order to satisfy the constraint.

Example 5. All *Regular Path Constraints* (RPCs) considered in (Barceló & Fontaine, 2017) can be written as Reg-GXPath expressions, since the language is capable to express implications of the form '$\alpha_1 \Rrightarrow \alpha_2$' where α_i is a 2RPQ for $i \in \{1, 2\}$. In particular, considering that sibling_of might represent either being brother_of or sister_of indistinctly, the constraints from Example 1 in (Barceló & Fontaine, 2017) can be written as Core-GXPath expressions in a straightforward way: the symmetry of the sibling_of relation can be stated as $\alpha = \downarrow_{\text{sibling\_of}} \Rrightarrow \downarrow^{-}_{\text{sibling\_of}}$, and the nibling condition as $\beta = \downarrow_{\text{child\_of}} \downarrow_{\text{sibling\_of}} \Rrightarrow \downarrow_{\text{nibling\_of}}$. These are exactly the constraints we wanted to express in Figure 1. Now we can formally state that the data-graph is inconsistent, since (María, Julieta) $\notin [\![\beta]\!]$. Meanwhile, every pair (x, y) belongs to $[\![\alpha]\!]$.

Deleting the edge (María, child_of, Diego) would make the graph consistent, since María would no longer be Julieta's nibling and thus β would be satisfied. On the other hand, deleting the edge (Diego, sibling_of, Julieta) would also fix β, but it would cause α to not be satisfied anymore, since the brotherhood relation would not be symmetric. Note that adding the edge (Maria, nibling_of, Julieta) would also make the data-graph consistent.

Example 6. We can express restrictions that are similar to *foreign keys* using Core-GXPath, relating data values from different entities. Consider a database where we have information about citizens and cities. We may like to ensure that a person's nationality coincides with the country of the city he was born in. This could be captured with the expression:

$$\varphi = \langle \downarrow_{\text{type}} [person^{=}] \rangle \Rightarrow \langle \downarrow_{\text{bornInCity}} \downarrow_{\text{nation}} = \downarrow_{\text{nationality}} \rangle$$

Example 7. NREs $\subseteq$ Reg-GXPath (Libkin et al., 2016) can express restrictions that take into account typing and inheritance properties common to the *Resource Description Framework* (RDF), such as those considered in (Pérez et al., 2010).

Repairs When a graph database G is inconsistent with respect to a set of restrictions R (i.e. there is a path expression or node expression in R that is not satisfied) we would like to compute a new graph database G' consistent with respect to R and that differs minimally from G. This new database G' is usually called a *repair* of G with respect to R following some formal definition for the semantics of 'minimal difference'.

We consider *set repairs*, in which the notion of minimality is based on the difference between the sets of nodes and edges. While we could provide a notion of distance between arbitrary data-graphs via an adequate definition of symmetric difference, it has been the case that the complexity of finding such repairs is quite high. Thus, it is common to consider only those set repairs in which one graph is obtained from the other by only adding or only deleting information (Barceló & Fontaine, 2017; Lukasiewicz et al., 2013; Ten Cate et al., 2012). This gives raise to subset and superset repairs.

We say that a data-graph $G = (V, L_e, D)$ is a **subset** of a data-graph $G' = (V', L'_e, D')$ (noted as $G \subseteq G'$) if and only if $V \subseteq V'$, and for every pair $v, v' \in V$ holds that $L_e(v, v') \subseteq L'_e(v, v')$ and $D(v) = D'(v)$. In this case, we also say that G' is a **superset** of G.

Definition 8 (Subset and superset repairs)**.** Let R be a set of restrictions and G a data-graph. We say that G' is a **subset repair** (respectively, **superset repair**) or $\subseteq$-repair (respectively, $\supseteq$-repair) of G if:

- (G', R) is **consistent** (i.e. $G' \models R$)
- $G' \subseteq G$ (respectively, $G' \supseteq G$)
- There is no data-graph G'' such that (G'', R) is consistent and $G' \subset G'' \subseteq G$ (respectively, $G' \supset G'' \supseteq G$)

We denote the set of subset repairs of G with respect to R as $\subseteq$-*Rep*(G, R) (respectively, the set of superset repairs as $\supseteq$-*Rep*(G, R)).

Example 9. In Figure 1, deleting the edge (Maria, child_of, Diego) results in a $\subseteq$-repair, while adding the edge (Maria, nibling_of, Julieta) results in a $\supseteq$-repair. Deleting both (Diego, sibling_of, Julieta) and (Julieta, sibling_of, Diego) also creates a $\subseteq$-repair, since this new graph database is consistent with respect to R and is also maximal.

3. Computing Repairs

In this section, we study the computational complexity of the repair computing problem for a given data-graph and a set of constraints. We start by mentioning a useful fact related to Reg-GXPath expressions that we use later in this section.

Theorem 10. *(Libkin et al., 2016, Theorem 4.3) There is an algorithm such that, given a Reg-GXPath expression α and a data-graph G, computes the set $[\![\alpha]\!]_G$ in polynomial time in the size of G and α.*

It follows from this result that, given a set of Reg-GXPath expressions R where $|R| = \Sigma_{\alpha \in R}|\alpha|$ and a data-graph G, it is possible to check if G is consistent with respect to R in polynomial time.

In the sequel, we study the complexity of the subset and superset repair-computing problems, which naturally depend on the fragment of Reg-GXPath used to express the constraints. For some results we consider R fixed, which is a common simplification of the repair-computing problem usually called the **data complexity** of the problem (Vardi, 1982). We obtain bounds for both the combined and data complexity of the problems. For most proofs, we require Σ_e and Σ_n to fulfill simple properties, such as having cardinality bigger than a constant c. We consider these subsets fixed, unless stated otherwise.

3.1 Subset Repairs

Let us consider the *empty graph* defined as $G = (\emptyset, L, D)$, from now on denoted by $\emptyset$. Since the empty graph satisfies every set of restrictions, we conclude that every graph G has a subset repair given any set R of restrictions. In order to understand the complexity of finding such repairs, we define the following decision problem:

Problem: ∃SUBSET-REPAIR
Input: A data-graph G and a set R of expressions from $\mathcal{L}$.
Output: Decide if G has a subset repair $G' \neq \emptyset$ with respect to R.

The difficulty of this problem depends on the set of expressions $\mathcal{L}$. Throughout this work, we will assume that $\mathcal{L}$ is always a set of expressions included in Reg-GXPath. Observe that this problem can be reduced to the problem of finding a subset repair of G with respect to R and, therefore, by studying it we can derive lower bounds for the problem of computing subset repairs.

∃SUBSET-REPAIR is in NP, since we can ask for a non-empty subset of G that satisfies R as a positive certificate. In what follows, we prove that this problem is NP-complete even if considering only path expressions from Reg-GXPath^{pos}.

Theorem 11. *There exists a set R of Reg-GXPath^{pos} path expressions such that the problem* ∃SUBSET-REPAIR *is* NP-complete.

Proof. We reduce 3-SAT to ∃SUBSET-REPAIR, considering a fixed set R of restrictions. Given a 3CNF formula ϕ of n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$ we want to construct a data-graph $G = (V, L, D)$ such that G has a non-trivial subset repair with respect to R if and only if ϕ is satisfiable. We give a description of R once we outline the structure of the data-graph G, and it will not depend on ϕ.

First, let us define the nodes of the graph. We have two 'boolean nodes' for each variable, representing each possible assignment. In addition, we have one node for every clause.

$$V_G = \{ \perp_i \mid 1 \leq i \leq n \} \cup \{ \top_i \mid 1 \leq i \leq n \} \cup \{ c_j \mid 1 \leq j \leq m \}.$$

In a valid subset repair, only one of the nodes $\perp_i$ or $\top_i$ will remain, and that will implicitly define a truth assignment on the variable x_i of ϕ.

For convenience, we will now define the edges of the data-graph by describing its set of edges E_G, instead of defining the function L_G. Nonetheless, both structures contain the same information, since $\text{E} \in L_G(v, w) \iff (v, \text{E}, w) \in E_G$. Also, we will need Σ_e to contain at least 4 elements, so that $\Sigma_e \supseteq \{\text{NEEDS}, \text{EXISTS}, \text{UNIQUE}, \text{VALID}\}$.[1]

1. It is not necessary that Σ_e contains an edge label named asNEEDS, but we rather denote by the name NEEDS one of the data values in Σ_e (and do so analogously for the rest of the labels).

We will split the edges conceptually in 4 groups as follows:

$$E_G = E_{needs} \cup E_{exists} \cup E_{unique} \cup E_{valid}$$

where

$$\begin{aligned}
E_{needs} =& \{(c_j, \textsc{needs}, \top_i) : x_i \text{ is a literal of } c_j\} \\
& \cup \{(c_j, \textsc{needs}, \bot_i) : \neg x_i \text{ is a literal of } c_j\} \\
E_{exists} =& \{(c_j, \textsc{exists}, c_{j+1}) : 1 \leq j \leq m-1\} \\
& \cup \{(c_m, \textsc{exists}, \bot_1), (c_m, \textsc{exists}, \top_1)\} \\
& \cup \{((\star^1)_i, \textsc{exists}, (\star^2)_{i+1}) : \star^1, \star^2 \in \{\bot, \top\}\} \\
& \cup \{(\bot_n, \textsc{exists}, c_1), (\top_n, \textsc{exists}, c_1)\} \\
E_{unique} =& \{(v, \textsc{unique}, \textsc{w}) : (v, w) \neq (\top_i, \bot_i) \text{ for } 1 \leq i \leq n\} \\
E_{valid} =& \{(v, \textsc{valid}, w) : (v, w) \neq (c_j, \star_i) \text{ for any } 1 \leq j \leq m, 1 \leq i \leq n, \star \in \bot, \top\}
\end{aligned}$$

We will not need the data values, so we set $D_G(v) = 0$ for all $v \in V$.
And, finally, the fixed set of path expressions is $R = \{\beta_1, \beta_2, \beta_3\}$ where:

$$\begin{aligned}
\beta_1 =& \downarrow^{+}_{\textsc{exists}} \\
\beta_2 =& \downarrow_{\textsc{unique}} \\
\beta_3 =& \downarrow_{\textsc{valid}} \cup \downarrow_{\textsc{needs}}\downarrow_{\textsc{valid}}
\end{aligned}$$

Now we explain the intuition behind this construction. First, we encode the information of the formula ϕ into the NEEDS edges of our data-graph. For every clause node c_j there is an edge to a boolean node $\bot_i$ (respectively, $\top_i$) if and only if the literal $\neg x_i$ appears in c_j (respectively without $\neg$). In a valid subset repair, we want every clause c_j to keep at least one of these edges, since that would imply that c_j is satisfied.

A technical problem arises from this approach: it could be the case that not every node is present in a repair, and thus this might result in the corresponding clause node c_j to be removed. To avoid this scenario, we rely on the EXISTS edges, that form a certain 'hamiltonian cycle' along the nodes of the data-graph: any node v can reach any other node w in G by using EXISTS edges. Note, however, that if any clause node c_j is removed from G, then this property will not hold anymore, and therefore β_1 will not be satisfied. Therefore, every clause node will be present in any non trivial repair.

Also notice that, because of the particular structure of E_{exists}, in a non trivial repair a node $\bot_i$ or $\top_i$ can be missing, as long as the other one is present. Since we want only one to remain, we added the UNIQUE edges and β_2. Observe that all pair of nodes satisfy β_2, except for those of the form $(\top_i, \bot_i)$ for some i. Therefore, in any non trivial repair of G either $\top_i$ or $\bot_i$ will be missing (but not both, since otherwise β_1 would break) for all $1 \leq i \leq n$.

Finally, we need to ensure that the assignment defined by any non-trivial repair satisfies ϕ. To do this, we use the edges E_{valid} and β_3.

Every pair of nodes belongs to the VALID relation, except for those of the form $(c_j, \star_i)$, where $\star \in \{\bot, \top\}$. Moreover, those pairs of nodes will satisfy β_3 only if at least one of the

NEEDS edges remains for every c_j. This means that in a non-trivial repair, every clause c_j must have a NEEDS edge directed to one of its variables valuations that evaluates c_j to $\top$. Thus, the valuation implicitly defined by the $\bot_i$,$\top_i$ nodes that remain in the repair will be a satisfying one.

Intuitively, if there is a non-trivial repair of G with respect to $R = \{\beta_1, \beta_2, \beta_3\}$, then such a repair consists of a selection of boolean nodes for each i such that every clause node keeps a NEEDS edge. Notice that R is independent of the 3-SAT formula ϕ.

Let us prove that ϕ is satisfiable if and only if G has a non-trivial repair with respect to R.

$\Longrightarrow$) If ϕ is satisfiable then there is an assignment f of its variables such that ϕ evaluates to true. Consider the graph G, and delete from G every node $\star_i$ for $\star \in \{\bot, \top\}$ such that $f(x_i) \neq \star_i$.

Since we deleted only one boolean node for each i, β_1 is satisfied. β_2 is satisfied, since there is no i for which both $\top_i$ and $\bot_i$ remain in the graph. Finally, since f is a valid assignment, we know that for every clause node c_j one of its NEEDS edges remains, hence β_3 is satisfied.

This implies that the subgraph satisfies R, which means that there exists a non-trivial subset repair of G with respect to R.

$\Longleftarrow$) Let G' be a non-trivial repair of G with respect to R. Since both β_1 and β_2 are satisfied, we know that all clause nodes c_j belong to G' and that, for every i, one of the Boolean nodes belongs to G'. Hence, we define an assignment f on the variables x_i as $f(x_i) = \top \iff \top_i \in V_{G'}$. Since β_3 is satisfied, at least one literal from each clause evaluates to true.

It follows that f is an assignment that evaluates ϕ to true.

□

Note that this proof only applied the Kleene star operator to edge labels, so it actually proved the hardness when considering the 'core' fragment of Reg-GXPath^{pos}.

This problem is NP-HARD when considering only node expressions from Reg-GXPath:

Theorem 12. *There exists a set R of Reg-GXPath node expressions such that the problem* ∃SUBSET-REPAIR *is* NP-COMPLETE.

Proof. We reduce 3-SAT to ∃SUBSET-REPAIR, considering a fixed set R of node expressions from Reg-GXPath. Given a 3CNF formula ϕ of n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$, we will construct a data-graph $G = (V, L, D)$ such that G has a non-trivial subset repair with respect to R if and only if ϕ is satisfiable. We give a description of R once we outline the structure of the data-graph G. For our reduction we require that $|\Sigma_e| \geq 4$ and $|\Sigma_n| \geq 4$, and in particular we will assume that we have edge labels $\{\text{ASSIGN}, \text{NEEDS\_TRUE}, \text{NEEDS\_FALSE}, \text{EXISTS}\} \subseteq \Sigma_e$ and data values $\{var, clause, \top, \bot\} \subseteq \Sigma_n$. As in Theorem 11, we will describe the edges of the data-graph by defining the set E of edges, instead of the function L.

The data-graph G is defined as

$$\begin{aligned} V_G =& \{\bot, \top\} \cup \{x_i : 1 \leq i \leq n\} \cup \{c_j : 1 \leq j \leq m\} \\ E_G =& E_{assign} \cup E_{needs} \cup E_{exists} \end{aligned}$$

$$D_G(v) = \begin{cases} var & v = x_i \text{ for some } 1 \leq i \leq n \\ clause & v = c_j \text{ for some } 1 \leq j \leq m \\ \star & v = \star \text{ for } \star \in \{\top, \bot\} \end{cases}$$

where:

$$\begin{aligned} E_{assign} =& \{(x_i, \text{ASSIGN}, \star) : \star \in \{\top, \bot\}\} \\ E_{needs} =& \{(c_j, \text{NEEDS\_TRUE}, x_i) : x_i \text{ is a literal of } c_j\} \\ & \cup \{(c_j, \text{NEEDS\_FALSE}, x_i) : \neg x_i \text{ is a literal of } c_j\} \\ E_{exists} =& \{(c_m, \text{EXISTS}, \bot), (\bot, \text{EXISTS}, \top), (\top, \text{EXISTS}, x_1)\} \\ & \cup \{(x_i, \text{EXISTS}, x_{i+1} : 1 \leq i \leq n-1)\} \\ & \cup \{(x_n, \text{EXISTS}, c_1)\} \\ & \cup \{(c_j, \text{EXISTS}, c_{j+1}) : 1 \leq j \leq m-1)\}. \end{aligned}$$

The set of node expressions $R = \{\psi_1, \psi_2, \psi_3\}$ is defined as:

$$\begin{aligned} \psi_1 =& \langle \downarrow_{\text{EXISTS}} \rangle \\ \psi_2 =& [var^{\neq} \vee \neg \langle \downarrow_{\text{ASSIGN}} \neq \downarrow_{\text{ASSIGN}} \rangle] \\ \psi_3 =& [clause^{\neq} \vee \langle \downarrow_{\text{NEEDS\_TRUE}} \downarrow_{\text{ASSIGN}} [\top] \rangle \vee \langle \downarrow_{\text{NEEDS\_FALSE}} \downarrow_{\text{ASSIGN}} [\bot] \rangle] \end{aligned}$$

The data-graph G is a representation of the input formula ϕ, where each variable is assigned to both boolean values $\bot$ and $\top$. A subset repair of G with respect to R will represent a proper assignment, where each variable node x_i has a unique outgoing edge of label ASSIGN. This condition is imposed by ψ_2, since it forbids the case in which a node with data value *var* has two outgoing edges to nodes having different data value.

Observe that ψ_3 implies that in a valid repair every clause will either contain a literal that is a negation of a variable that is assigned to $\bot$, or a non-negated variable assigned to $\top$. Finally, ψ_1 forces the repair to either preserve all nodes or none of them: if a proper subset of the nodes is deleted then at least one node will not have an outgoing edge of label EXISTS, and therefore R will not be satisfied.

Now, we prove that ϕ is satisfiable if and only if G has a non-trivial subset repair with respect to R.

$\Longrightarrow$) Let $f : \{x_i\}_{1 \leq i \leq n} \to \{\top, \bot\}$ be an assignment that satisfies ϕ. Hence, we can define a sub data-graph G' of G by removing the edges $(x_i, \text{ASSIGN}, \neg f(x_i))$. Since every node is present and every *var* node has an unique outgoing edge of label ASSIGN, ψ_1 and ψ_2 are satisfied in this data-graph. Moreover, ψ_3 is also satisfied: given a *clause* node c_j, there is a literal x_i or $\neg x_i$ in c_j that is satisfied through f, and the data-graph G' contains the edges of the form $(x_i, \text{ASSIGN}, f(x_i))$. Therefore, from every *clause* node there will be a path with the proper edge labels to either $\bot$ or $\top$. Since G' satisfies R and $G' \subset G$, there must be a non-trivial subset repair $G'' \supseteq G'$ of G with respect to R.

$\Longleftarrow$) Given a non-trivial subset repair G' of G with respect to R, we build a satisfying assignment of ϕ. Since ψ_1 is satisfied by G', every node from G is present in G'. We define the following assignment $f : \{x_i\}_{1 \leq i \leq n} \to \{\top, \bot\}$ such that $f(x_i) = \top \iff \textsc{assign} \in L_{G'}(x_i, \top)$. We claim that this assignment satisfies ϕ.

Given a clause c_j from ϕ, the node c_j in G' satisfies ψ_3. This means that, there is either a path $\downarrow_{\textsc{needs\_true}}\downarrow_{\textsc{assign}} [\top]$ or a path $\downarrow_{\textsc{needs\_false}}\downarrow_{\textsc{assign}} [\bot]$ starting at the node c_j. Assume without loss of generality that the path is of the form $\downarrow_{\textsc{needs\_false}}\downarrow_{\textsc{assign}} [\bot]$ (the other case follows analogously), and that the intermediate node is x_i. By construction, we know that $\neg x_i$ is a literal in c_j. Due to constraint ψ_2, the node x_i has at most one outgoing edge of label $\textsc{assign}$, and therefore $\textsc{assign} \in L(x_i, \bot)$ implies that $\textsc{assign} \notin L(x_i, \top)$, which by definition of f also implies that $f(x_i) = \bot$. We conclude that c_j is satisfied.

□

Computing a repair seems unfeasible when using path constraints from Reg-GXPath^{pos} or node constraints from Reg-GXPath, so we now study the ∃SUBSET-REPAIR problem when $\mathcal{L}$ contains only *node expressions* from Reg-GXPath^{pos}.

First, we prove that the positive fragment of Reg-GXPath^{pos} satisfies a certain property of monotony:

Lemma 13 (Monotony of Reg-GXPath^{pos}). *Let G be a data-graph, α be a Reg-GXPath^{pos} path expression, φ be a Reg-GXPath^{pos} node expression, and G' be a data-graph such that $G \subseteq G'$. Then:*

- $[\![\alpha]\!]_G \subseteq [\![\alpha]\!]_{G'}$
- $[\![\varphi]\!]_G \subseteq [\![\varphi]\!]_{G'}$

Proof. Intuitively, if a pair of nodes satisfies $(v, w) \in [\![\alpha]\!]_G$ where α is a positive expression, then there is a certain subgraph H of G that is a witness of that fact. For example, if $(v, w) \in [\![\downarrow_l [c^=] \downarrow_l]\!]_G$ then there is a node z such that $D(z) = c$ and a path vzw made by edges with label l. The subgraph composed by these three nodes and two edges is enough for the pair (v, w) to satisfy the positive path expression α, and therefore in every superset G' of G it will be the case that $(v, w) \in [\![\alpha]\!]_G$. See Appendix A for the proper proof.

□

This fact allows us to define an efficient procedure for finding a subset repair, based on the following observation: if a node v in G does not satisfy a positive node expression φ, then there is no subset repair G' such that v is a node from G'. We will say that a node *v violates a node expression φ* from R iff $v \notin [\![\varphi]\!]_G$. More generally, we have the following result.

Theorem 14. *Let G be a data-graph, R a set of Reg-GXPath^{pos} restrictions, and v a node in G that violates a node expression from R. Then,*

$$\subseteq\text{-}Rep(G, R) = \subseteq\text{-}Rep(G_{V_G \setminus \{v\}}, R)$$

Proof. For the $\subseteq$ direction: let $H \in \mathit{Rep}(G, R)$ be a subset repair of G with respect to R. Due to Lemma 13 and the fact that there is a node expression $\varphi \in R$ such that $v \notin [\![\varphi]\!]_G$ we conclude that v is not a node from H. This implies that $H \subseteq G_{V_G \setminus \{v\}}$, and since H is a maximal consistent subset of G with respect to R, it also is a maximal consistent subset from $G_{V_G \setminus \{v\}}$.

For the other direction: let $H \in \mathit{Rep}(G_{V_G \setminus \{v\}}, R)$. Since H is consistent with respect to R and $H \subseteq G$ we only need to prove that it is maximal with respect to G. Toward a contradiction, suppose there exists a data-graph H' such that $H \subset H' \subseteq G$ and H' satisfies R. Since v is not a node from H', it follows that $H' \subseteq G_{V_G \setminus \{v\}}$, which then implies that H is not a repair of $G_{V_G \setminus \{v\}}$. This results in a contradiction, and hence we proved that H is a subset repair of G with respect to R.

$\square$

Furthermore, given two data-graphs G_1 and G_2 satisfying a Reg-GXPath^{pos} node expression φ it can be shown that $G_1 \cup G_2$ satisfies φ as well (this follows from Lemma 13). Then, if R only contains Reg-GXPath^{pos} node expressions we can conclude that there is a *unique* subset repair of G with respect to R.

Given all these facts, we define an algorithm that computes the unique subset repair of a data-graph given a set of Reg-GXPath^{pos} node expressions:

Algorithm 1 *SubsetRepair*(G, R)

```
Require: $G$ is a data-graph and $R$ a set of Reg-GXPath$^{pos}$ node expressions.
1: while $(G, R)$ is inconsistent do
2:     $V_\perp \leftarrow \{ v \mid v \in V_G$ and $\exists\ \varphi \in R$ such that $v \notin [\![\varphi]\!]_G \}$
3:     $G \leftarrow G_{V_G \setminus V_\perp}$
4: end while
5: return $G$
```

This method is correct since Theorem 14 implies that $\mathit{Rep}(G, R) = \mathit{Rep}(G_{V_G \setminus V_\perp}, R)$. It also terminates, since $(\emptyset, R)$ is consistent. The set $V_\perp$ can be computed in polynomial time, and since there are at most $|V_G|$ iterations we conclude the following:

Theorem 15. *There is an algorithm such that, given a data-graph G and a set of Reg-GXPath^{pos} node expressions R, the algorithm computes the unique subset repair of G with respect to R in polynomial time.*

Note that Theorem 11 readily implies that the problem ∃SUBSET-REPAIR is NP-hard for $\mathcal{L} =$ Reg-GXPath^{pos} if we allow both node and path expressions, even when considering the data complexity of the problem.

We conclude this section by noticing that even though the problem of finding subset repairs turns out to be NP-complete for a fragment of quite simple path expressions (Theorem 11), the case for positive node expressions is more tractable (Theorem 15), even though they can be substantially expressive in some cases. More precisely, while some of our previous examples such as Examples 4 and 6, actually make use of node expression negation for building the 'implications', they can be rewritten to avoid the use of node expression negation under some assumptions over the database.

Indeed, the aforementioned examples use node expression negation to make 'typed restrictions' over the data-graph. These are a natural kind of restriction for databases in which each node has properties associated to a single value, in a manner codified via edges of the form $\downarrow_{\text{TYPE}}$ (either with a single generic label $\downarrow_{\text{TYPE}}$ or with many, such as having $\downarrow_{\text{NATIONALITY}}$ and $\downarrow_{\text{PROFESSION}}$). In these cases, we can use a node expression $\mathit{type}(\text{C}) := \langle \downarrow_{\text{TYPE}} [\text{C}^{=}] \rangle$ to indicate that a node is of type C with respect to the property TYPE, and we can ask that all such nodes satisfy a particular restriction via the node expression $\mathit{type}(\text{C}) \Rightarrow \mathit{restriction}$. In the cases where Σ_n is finite and where each node has exactly 1 outgoing edge $\downarrow_{\text{TYPE}}$, then instead of writing $\mathit{type}(\text{C}) \implies \mathit{restriction}$ we can write the Reg-GXPath^{pos} node expression $(\bigvee_{\text{D}\neq\text{C}} \mathit{type}(\text{D})) \vee \mathit{restriction}$, which will retain the original semantics of the implication. Also, since the set of node labels Σ_n is fixed, this translation into the positive fragment will at most increment the expression length by a constant factor (given by the quotient of the lengths of $(\bigvee_{\text{D}\neq\text{C}} \mathit{type}(\text{D})) \vee \mathit{restriction}$ and $\mathit{type}(\text{C}) \Rightarrow \mathit{restriction}$, which is independent of the particular C or TYPE).

Finally, we remark that the algorithm for Reg-GXPath^{pos} node expressions will work as long as the expressions satisfy monotony. This means that we could add more monotone tools to the language and the procedure would still be correct and run in polynomial type, assuming of course that the expressions from the new language can be evaluated in polynomial time given the length of the expressions and the data-graph.

3.2 Superset Repairs

We start this section with the following remark, contrasting with the subset repair problem:

Remark 16. There exists a data-graph G and a set R of Reg-GXPath^{pos} expressions such that there is no superset repair of G with respect to R.

For example, consider $G = (\{v\}, L, D)$ where $D(v) = c$, $L(v, v) = \emptyset$ and the set R with only one node expression $\phi = [c^{\neq}]$. Every superset G' of G contains v with data value c, which implies that $v \notin [\![\phi]\!]_{G'}$, and thus G' is not consistent with respect to R.

In order to study the complexity of finding superset repairs, we define the following decision problem:

Problem: ∃SUPERSET-REPAIR
Input: A data-graph G and a set R of $\mathcal{L}$ expressions.
Output: Decide if G has a superset repair G' with respect to R.

When fixing R the problem remains intractable in general.

Theorem 17. *There exists a set R of path expressions from Reg-GXPath and a set of edge labels Σ_e such that the problem* ∃SUPERSET-REPAIR *is undecidable.*

Proof. To prove undecidability, we reduce the superset-CQA problem (Barceló & Fontaine, 2017, Theorem 4) to our problem. Since the problem is undecidable, so will be ours.

In (Barceló & Fontaine, 2017, Theorem 4), the following CQA problem is proven to be undecidable for a particular choice of Σ, q, Γ: given a finite alphabet Σ, a non-recursive RPQ query q, a set of word constraints Γ, a graph G, and a tuple (x, y) of nodes of G, decide whether $(x, y) \in [\![q]\!]_{G'}$ for every $G' \in \supseteq\text{-}Rep(G, \Gamma)$ (i.e., there is a q-labeled path from x to y in all supersets of G satisfying the constraints Γ).

For ease of reference, we now provide the required definitions. Given an alphabet Σ, regular path queries over Σ, noted RPQ, are defined by the following grammar:

$$\eta = \epsilon \mid \textsc{a} \mid \eta.\eta \mid \eta \cup \eta \mid \eta^* \tag{1}$$

We say that an RPQ is *non-recursive* if it does not mention the Kleene-star.

The semantics for RPQ formulas over a graph G with edges labeled in Σ is as follows:

$$\begin{aligned}
&[\![\epsilon]\!]_G = \{(v, v) \mid v \in V\} \\
&[\![\textsc{a}]\!]_G = \{(v, w) \mid v, w \in V, \textsc{a} \in L(v, w)\} \\
&[\![\eta.\eta']\!]_G = \{(v, x) \mid \exists w \in G \text{ s.t. } (v, w) \in [\![\eta]\!]_G \text{ and } (w, x) \in [\![\eta']\!]_G\} \\
&[\![\eta \cup \eta']\!]_G = [\![\eta]\!]_G \cup [\![\eta']\!]_G \\
&[\![\eta^*]\!]_G = \{(v, w) \mid (v, w) \text{ belongs to the reflexive-transitive closure of } [\![\eta]\!]_G\}
\end{aligned}$$

A word constraint is defined as a formula $\alpha_1 \subseteq \alpha_2$ where α_i is a word formula for $i \in \{1, 2\}$; that is, a finite conjunction of labels. A graph G satisfies a word constraint $\alpha_1 \subseteq \alpha_2$ if $[\![\alpha_1]\!]_G \subseteq [\![\alpha_2]\!]_G$.

We note that a word constraint $\alpha = \textsc{a}_1 \dots \textsc{a}_m \subseteq \textsc{a}'_1 \dots \textsc{a}'_n$ is equivalent to a Reg-GXPath formula $\alpha^{\mathcal{T}} = \downarrow_{\textsc{a}_1} \dots \downarrow_{\textsc{a}_m} \Rrightarrow \downarrow_{\textsc{a}'_1} \dots \downarrow_{\textsc{a}'_n}$ (that is, $\downarrow_{\textsc{a}'_1} \dots \downarrow_{\textsc{a}'_n} \cup \overline{\downarrow_{\textsc{a}_1} \dots \downarrow_{\textsc{a}_m}}$). For a non-recursive RPQ q there is also a straightforward translation to an equivalent Reg-GXPath formula $q^{\mathcal{T}}$ that preserves its semantics, since a non-recursive RPQ is a finite union of word formulas.

We define $R' = \Gamma^{\mathcal{T}} \cup \{\downarrow_{\textsc{x}} \Rrightarrow \overline{q^{\mathcal{T}}}\}$, where $\Gamma^{\mathcal{T}} = \{\alpha^{\mathcal{T}} : \alpha \in \Gamma\}$ and x is a fresh edge label. We consider the set of edge labels $\Sigma_e = \Sigma \cup \{\textsc{x}\}$. Given $(G, (x, y))$, let $\hat{G}$ be the graph G augmented with an edge of label x such that x is connected with y via $\downarrow_{\textsc{x}}$. Thus, it follows that:

$$\begin{aligned}
&(G, (x, y)) \notin \supseteq\text{-}CQA(q, \Gamma) \\
\iff &\exists G' \in \supseteq\text{-}Rep(\hat{G}, \Gamma^{\mathcal{T}}) : (x, y) \notin [\![q^{\mathcal{T}}]\!]_{G'} \\
\iff &\exists G' \in \supseteq\text{-}Rep(\hat{G}, R') \\
\iff &\exists \text{SUPERSET-REPAIR}(\hat{G},\ R') = \textsc{true}
\end{aligned}$$

□

Notice that the argument of the previous proof only requires a limited fragment of Reg-GXPath, and therefore the following observation holds:

Observation 18. *The problem is undecidable even when only considering restrictions from the fragment of Reg-GXPath that has no node expressions and whose path expressions are of the form:*

$$\alpha, \beta = \epsilon \mid \textsc{a} \mid \alpha.\beta \mid \alpha \cup \beta \mid \overline{\alpha}$$

As for the particular case of ∃SUPERSET-REPAIR that considers restriction sets only consisting of node-expressions, we have the following result:

Theorem 19. *There is a set of labels Σ_e such that* ∃SUPERSET-REPAIR *is undecidable, even when R is a set of node expressions (but it is not fixed).*

Proof. We reduce the problem Finite 2RPQ Entailment from $\mathcal{ALCOIF}$ KBs (Rudolph, 2016) to our problem. First, we give some background and the required definitions.

Let N_C, N_R and N_I be countably infinite disjoint sets representing *concept names* (unary relations), *role names* (binary relations) and *individual names* (constants), respectively. An *assertion* is an expression of the form $C(a)$ or $r(a,b)$, where a,b are individual names, C is a concept name, and r is a role name. A *concept* is any of the following expressions:

$$A, B = \top \mid C \mid \neg A \mid A \sqcap B \mid A \sqcup B \mid \forall r.A \mid \exists r.A \mid \{a\} \tag{2}$$

where $C \in N_C$, $r \in N_R$ and $a \in N_I$. A concept of the form $\{a\}$ is called a *nominal.* The set of all the nominals given by Σ_I is denoted by *nom*, which we assume disjoint from N_C, N_R.

An $\mathcal{ALCOIF}$ *axiom* is a concept inclusion $A \sqsubseteq B$ or a functionality restriction $Fun(r)$ where r is a role name (see semantics below). A *knowledge base* (KB from now on) is a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ where $\mathcal{T}$ (the *TBox*) is a finite set of $\mathcal{ALCOIF}$ axioms and $\mathcal{A}$ (the *ABox*) is a finite set of assertions. We denote $CN(\mathcal{K})$ to the set of concept names in $\mathcal{K}$, $ind(\mathcal{K})$ to the set of individuals in $\mathcal{K}$ and $nom(\mathcal{K})$ to the set of nominals in $\mathcal{K}$.

An *interpretation* is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty set called the *domain* of $\mathcal{I}$, and $\cdot^{\mathcal{I}}$ is a mapping called the *interpretation function*, that assigns as follows:

- $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ for every concept name C
- $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for every role name r
- $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for every individual name a

We say that $\mathcal{I}$ is a *finite interpretation* if $\Delta^{\mathcal{I}}$ is finite. For convenience, we assume $\Delta^{\mathcal{I}} \cap N_I = \emptyset$ for every $\mathcal{I}$. The semantics for the remaining concepts can be extended as follows:

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
(\neg A)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}} \\
(A \sqcap B)^{\mathcal{I}} &= A^{\mathcal{I}} \cap B^{\mathcal{I}} \\
(A \sqcup B)\mathcal{I} &= A^{\mathcal{I}} \cup B^{\mathcal{I}} \\
(\forall r.A)^{\mathcal{I}} &= \{u \in \Delta^{\mathcal{I}} : \forall v.(u,v) \in r^{\mathcal{I}} \to v \in A^{\mathcal{I}}\} \\
(\exists r.A)^{\mathcal{I}} &= \{u \in \Delta^{\mathcal{I}} : \exists v.(u,v) \in r^{\mathcal{I}} \wedge v \in A^{\mathcal{I}}\} \\
\{a\}^{\mathcal{I}} &= \{a^{\mathcal{I}}\}
\end{aligned}$$

An interpretation $\mathcal{I}$ is a *model* of $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ if all of the following assertions hold:

- $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ for every concept inclusion $A \sqsubseteq B$ in $\mathcal{T}$

- for every $Fun(r)$ in $\mathcal{T}$ is true that $(u, v), (u, w) \in r^{\mathcal{I}}$ implies $v = w$, for all $u, v, w \in \Delta^{\mathcal{I}}$
- $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for every assertion $C(a)$ in $\mathcal{A}$
- $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ for every assertion $r(a, b)$ in $\mathcal{A}$

The signature of any KB structure can be extended with new unary symbols to obtain a normal form for the concept inclusions (Gogacz et al., 2018), hence we assume without loss of generality that the TBox only contains concept inclusions of the form

$$\bigsqcap_i A_i \sqsubseteq \bigsqcup_j B_j, \quad A \sqsubseteq \forall r.B, \quad A \sqsubseteq \exists r.B, \quad A \equiv \{a\},$$

where A_i, B_j, A, B are concept names and $\{a\}$ is a nominal.

A 2RPQ is defined analogously as in (1), with the difference that we also allow to traverse edges backwards, i.e., we add r^- to the syntax.

The problem Finite 2RPQ Entailment from $\mathcal{ALCOIF}$ KBs asks, given a KB $\mathcal{K}$ and a 2RPQ q, whether $\mathcal{I} \models \exists x.q(x, x)$ is true for every finite model $\mathcal{I}$ of $\mathcal{K}$, or not. Notice that this problem is quite similar to superset-CQA, with the exception that the underlying structure given by the ABox is not just a graph database, but is a graph that allows multiple labels on edges and nodes, and the set of restrictions is now given by the TBox. In other words, we ask if the query is valid on any model that "repairs" the KB, seen as a partial representation of a graph-like model. Finite 2RPQ Entailment from $\mathcal{ALCOIF}$ KBs is undecidable (Rudolph, 2016), even for a finite set Σ of role names, which is the particular case of the problem we consider from now on.

For every KB $\mathcal{K}$ and 2RPQ q we will construct a data-graph $G(\mathcal{K})$ and a set of node constraints $R(\mathcal{K}, q)$ such that $G(\mathcal{K})$ has a superset-repair with respect to $R(\mathcal{K}, q)$ if and only if there is a finite *counter-model* for $\mathcal{K}$ and $\exists x.q(x, x)$, that is, a finite model of $\mathcal{K}$ that does not satisfy the query.

We fix the set of edge labels to be $\Sigma_e = \{\downarrow_r : r \in \Sigma\} \cup \{\downarrow_{\text{TOTAL}}\}$, and the set of data values to be the disjoint union $\Sigma_n = \{P_I : I \subset N_C \cup \textit{nom} \text{ is finite}\} \sqcup \{T_I : I \subset N_C \cup \textit{nom} \text{ is finite}\}$.[2] Intuitively, we will use the data values P_I as a preliminary description of the concepts used in the ABox of $\mathcal{K}$, and the data values T_I as a total description of the concepts used in a model of $\mathcal{K}$. We call *partial node* a node having data value P_I, and *total node* a node having data value T_I. For a particular $\mathcal{K}$, the pair $(G(\mathcal{K}), R(\mathcal{K}, q))$ will be defined over the set of data values $\{P_I : I \subseteq CN(\mathcal{K}) \cup \textit{nom}(\mathcal{K})\} \sqcup \{T_I : I \subseteq CN(\mathcal{K}) \cup \textit{nom}(\mathcal{K})\}$. In general, we assume that the indices I, J vary in the set of subsets of $CN(\mathcal{K}) \cup \textit{nom}(\mathcal{K})$. We say that a node *contains* a concept name A if A lies in the set I that corresponds to the index of its data value. For $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ let us consider $G(\mathcal{K}) = (V, L, D)$ where:

- $V = \textit{ind}(\mathcal{K})$
- $L(a, b) = \{\downarrow_r : r(a, b) \in \mathcal{A}\}$
- $D(a) = P_I$, where $I = \{C \in CN(\mathcal{K}) : C(a) \in \mathcal{A}\} \cup \{\{a\}\}$, if $\{a\} \in \textit{nom}(\mathcal{K})$, or simply $I = \{C \in CN(\mathcal{K}) : C(a) \in \mathcal{A}\}$ otherwise

2. Notice that Σ_n is countable since it is the union of two countable sets, each of them indexed over the set of finite subsets of $N_C \cup \textit{nom}$, and the set of all finite subsets of any countable set is also countable.

We now present the formulas in $R(\mathcal{K}, q)$, denoted by ψ_i, along with a short description of the property we want to model in each case:

(i) Formula ψ_1 as the conjunction of the following formulas:

$$\bigwedge_I (P_I^= \Rightarrow \langle \downarrow_{\text{TOTAL}} [\bigvee_{I \subseteq J} T_J^=] \rangle),$$

meaning that every partial node must be connected to a total node through the relation $\downarrow_{\text{TOTAL}}$, while preserving the original data values;

$$\neg \langle \downarrow_{\text{TOTAL}}^- \downarrow_{\text{TOTAL}} \cap \bar{\epsilon} \rangle,$$

implying that every partial node is connected to precisely one total node through $\downarrow_{\text{TOTAL}}$; and

$$\langle \downarrow_{\text{TOTAL}} \rangle \Rightarrow \bigvee_J P_J^=,$$

meaning that every total edge in the data-graph must be an outgoing edge from a partial node. In toto, ψ_1 means $\downarrow_{\text{TOTAL}}$ can be seen as an assignation of the individuals of $\mathcal{K}$. A total node connected to a partial node through the relation $\downarrow_{\text{TOTAL}}$ is called *the total counterpart* of that partial node. For convenience, we say that an outgoing edge $\downarrow_r$ of a partial node is also an outgoing edge of its total counterpart.

(ii) For every $\bigsqcap_i A_i \sqsubseteq \bigsqcup_j B_j$ in $\mathcal{T}$ consider:

$$\psi_2 = \neg \bigvee_{\substack{\forall i.\, A_i \in I \\ \forall j.\, B_j \notin I}} T_I^=,$$

which states that every node containing all the concepts A_i (that is, every node with data value T_I such that $A_i \in I$ for all i), must contain some B_j as well.

(iii) For every $A \sqsubseteq \forall r.B$ in $\mathcal{T}$ consider:

$$\psi_3 = \neg \langle [\bigvee_{A \in I} T_I^=] (\epsilon \cup \downarrow_{\text{TOTAL}}^-) \downarrow_r (\epsilon \cup \downarrow_{\text{TOTAL}}) [\bigvee_{B \notin J} T_J^=] \rangle,$$

which states that a total node containing the concept name A cannot reach a total node that does not contain the concept B through a path in $(\epsilon \cup \downarrow_{\text{TOTAL}}^-) \downarrow_r (\epsilon \cup \downarrow_{\text{TOTAL}})$. This path overlaps all the possible cases for which $\downarrow_r$ is an outgoing edge of the aforementioned total node, as established in item (i).

(iv) For every $A \sqsubseteq \exists r.B$ in $\mathcal{T}$ consider:

$$\psi_4 = \bigvee_{A \in I} T_I^= \Rightarrow \langle (\epsilon \cup \downarrow_{\text{TOTAL}}^-) \downarrow_r (\epsilon \cup \downarrow_{\text{TOTAL}}) [\bigvee_{B \in J} T_J^=] \rangle,$$

which states that every total node containing the concept name A must have an outgoing edge $\downarrow_r$ joining it with another total node containing concept B.

(v) For every $A \equiv \{a\}$ in $\mathcal{T}$, we consider the conjunction of the following formulas:

$$\psi_5 = \bigvee_{\{a\} \in I} P_I^{=} \Rightarrow \langle \downarrow_{\text{TOTAL}} [\bigvee_{A \in J} T_J^{=}] \rangle$$

$$\psi_6 = \bigvee_{A \in J} T_J^{=} \Rightarrow \langle \downarrow_{\text{TOTAL}}^{-} [\bigvee_{\{a\} \in I} P_I^{=}] \rangle$$

which states that a total node contains concept A if and only if such node is the total counterpart of a partial node containing the nominal $\{a\}$.

(vi) For every $Fun(r)$ in $\mathcal{T}$, we consider the conjunction of the following formulas:

$$\psi_7 = \neg \langle [\bigvee_I T_I^{=}] \downarrow_r^{-} \downarrow_r [\bigvee_I T_I^{=}] \cap \overline{\epsilon} \rangle$$

$$\psi_8 = \neg \langle [\bigvee_I T_I^{=}] \downarrow_r^{-} \downarrow_{\text{TOTAL}}^{-} \downarrow_r (\epsilon \cup \downarrow_{\text{TOTAL}}) [\bigvee_I T_I^{=}] \cap \overline{\epsilon} \rangle$$

$$\psi_9 = \neg \langle [\bigvee_I T_I^{=}] \downarrow_{\text{TOTAL}}^{-} \downarrow_r^{-} \downarrow_r (\epsilon \cup \downarrow_{\text{TOTAL}}) [\bigvee_I T_I^{=}] \cap \overline{\epsilon} \rangle$$

Notice that these three formulas could be unified into a single one. However, for simplicity we write them separately to help us clarify which patterns (and how) we wish to avoid in a valid repair, as depicted in Figure 3.

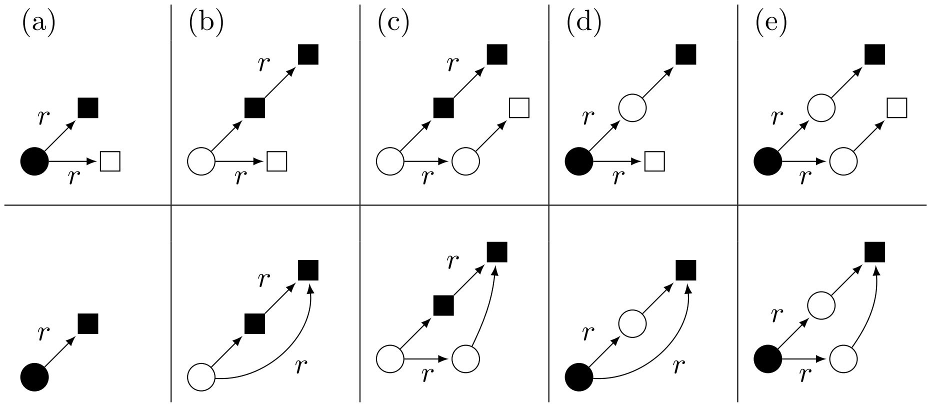

Figure 3: In the upper boxes, the patterns we do not wish to appear in a repair as specified by the constraints. In the lower boxes, how to fix each problem by merging the problematic □ node with a ■ node. The nodes are represented as follows: □ undesired total nodes, ■ total nodes, ○ partial nodes, ● any kind of nodes. The unlabeled edges represent total edges, thus the □ or ■ node is the total counterpart of the incident ○ node. The case (a) is handled by ψ_7, cases (b) and (c) by ψ_8, and cases (d) and (e) by ψ_9.

(vii) Finally, we should add a restriction that ensures the non-satisfiability of the query. But first, we show how to translate a 2RPQ α to obtain the path expression $\alpha^{\mathcal{S}}$ that we need. The translation is as follows:

$\epsilon \mapsto \epsilon \cup \downarrow_{\text{TOTAL}} \cup \downarrow_{\text{TOTAL}}^{-}$, denoted by $\epsilon^{\mathcal{S}}$

$$\begin{aligned}
&r \mapsto \epsilon^{\mathcal{S}} \downarrow_r \epsilon^{\mathcal{S}}\\
&r^- \mapsto \epsilon^{\mathcal{S}} \downarrow_r^- \epsilon^{\mathcal{S}}\\
&\alpha^* \mapsto (\alpha^{\mathcal{S}})^*\\
&\alpha \cdot \beta \mapsto \alpha^{\mathcal{S}} \cdot \beta^{\mathcal{S}}\\
&\alpha \cup \beta \mapsto \alpha^{\mathcal{S}} \cup \beta^{\mathcal{S}}
\end{aligned}$$

We add the formula $\psi_{10} = \langle \epsilon \cap \overline{q^{\mathcal{S}}} \rangle$ to $R(\mathcal{K}, q)$. Notice that if we delete every appearance of the symbol $\downarrow_{\text{TOTAL}}$ in a word ω from $\alpha^{\mathcal{S}}$ we obtain a word from $\alpha^{\mathcal{T}}$, where $\alpha^{\mathcal{T}}$ is the translation of α as defined in the proof of Theorem 17.

We now proceed to prove that there is a finite counter-model for $\mathcal{K}$ and $\exists x.q(x,x)$ if and only if there is a superset-repair for $G(\mathcal{K})$ and $R(\mathcal{K}, q)$.

$\implies$) Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a counter-model for $\mathcal{K}$ and $\exists x.q(x,x)$. We define the data-graph $G' = (V', L', D')$, where $V' = ind(\mathcal{K}) \cup \Delta^{\mathcal{I}} = V \cup \Delta^{\mathcal{I}}$, $D'|_V = D$, $L'|_{V \times V} = L$, for $u \in \Delta^{\mathcal{I}}$, $D'(u) = T_I$ with $I = \{A \in CN(\mathcal{K}) : u \in A^{\mathcal{I}}\} \cup \{\{a\} \in nom(\mathcal{K}) : a^{\mathcal{I}} = u\}$, and

$L'(a, a^{\mathcal{I}}) = \{\downarrow_{\text{TOTAL}}\}$, for every $a \in V$

$L'(u, v) = \{\downarrow_r : (u, v) \in r^{\mathcal{I}}\}$, for every $u, v \in \Delta^{\mathcal{I}}$

and $L'(u, v)$ is empty for any other remaining case

To differenciate an element u of $\mathcal{I}$ from a node of G' we underline it: $\underline{u}$ expresses "$u \in \Delta^{\mathcal{I}}$ as a node of G'".

Notice that if $r(a, b)$ is an assertion, then $\downarrow_r \in L'(a, b) \cap L'(\underline{a^{\mathcal{I}}}, \underline{b^{\mathcal{I}}})$, and if $\{a\}$ is a nominal in $\mathcal{K}$, then $[\![\bigvee_{\{a\} \in I} P_I^{=}]\!]_{G'} = \{a\}$. This is relevant to simplify the proof below.

We show that the graph G' satisfies each of the formulas in $R(\mathcal{K}, q)$. It follows by definition that formula ψ_1 in item (i) is satisfied.

For item (ii), suppose that $\bigsqcap_i A_i \sqsubseteq \bigsqcup_j B_j$ is in $\mathcal{T}$ and let $x \in V'$. If x is a partial node, then $x \in [\![\psi_2]\!]_{G'}$. If $x = \underline{u}$ and does not contain some concept A_i, then $x \in [\![\psi_2]\!]_{G'}$. If otherwise, $x = \underline{u}$ contains all the concepts A_i, to prove that $x \in [\![\psi_2]\!]_{G'}$ we only need to ensure that x also contains some concept B_j. Let T_I be the data value of x, that is, $D'(x) = T_I$ with $A_i \in I$ for all i. By definition of G', $u \in \Delta^{\mathcal{I}}$ and $u \in A_i^{\mathcal{I}}$ for every i. Since $\mathcal{I} \models \mathcal{T}$, we have that $u \in B_k^{\mathcal{I}}$ for some k, hence $B_k \in I$.

For item (iii), suppose that $A \sqsubseteq \forall r.B$ is in $\mathcal{T}$ and let $x \in V'$. If x is a partial node, or a total node not containing the concept A, then $x \in [\![\psi_3]\!]_{G'}$. If $x = \underline{u}$ contains the concept A but $\downarrow_r$ is not an outgoing edge of x (see item (i) for the meaning of outgoing edge), then $x \in [\![\psi_3]\!]_{G'}$. The only remaining case is when $x = \underline{u}$ contains the concept A and $\downarrow_r$ is an outgoing edge of x. Then, x reaches a node $y = \underline{v}$ through the path $(\epsilon \cup \downarrow_{\text{TOTAL}}^-) \downarrow_r (\epsilon \cup \downarrow_{\text{TOTAL}})$. To prove that $x \in [\![\psi_3]\!]_{G'}$ we only need to ensure that y contains the concept B. By definition of G', there are only two cases to consider: x reaches y through the path $\downarrow_r$; or through the path $\downarrow_{\text{TOTAL}}^- \downarrow_r \downarrow_{\text{TOTAL}}$ when both x and y are total counterparts of partial nodes. The first case occurs only if $(u, v) \in r^{\mathcal{I}}$ and since $u \in A^{\mathcal{I}}$ and $\mathcal{I} \models \mathcal{T}$, then $v \in B^{\mathcal{I}}$, as we wanted. For the second case, notice that it occurs only if $u = a^{\mathcal{I}}$, $v = b^{\mathcal{I}}$ and $r(a, b)$ is an assertion, so $\downarrow_r \in L'(x, y)$, and we are in the first case.

For item (iv), suppose that $A \sqsubseteq \exists r.B$ is in $\mathcal{T}$ and let $x \in V'$. If x is a partial node, or a total node not containing the concept A, then $x \in [\![\psi_4]\!]_{G'}$. If $x = \underline{u}$ contains the concept A, to prove that $x \in [\![\psi_4]\!]_{G'}$ we only need to ensure that x reaches through the path $(\epsilon \cup \downarrow^{-}_{\text{TOTAL}}) \downarrow_r (\epsilon \cup \downarrow_{\text{TOTAL}})$ a node y that contains the concept B. Indeed, by definition of G', $u \in A^{\mathcal{I}}$ and since $\mathcal{I} \models \mathcal{T}$, then there is $v \in \Delta^{\mathcal{I}}$ such that $(u, v) \in r^{\mathcal{I}}$ and $v \in B^{\mathcal{I}}$. Taking $y = \underline{v}$ we obtain the result.

For item (v), suppose $A \equiv \{a\}$ is in $\mathcal{T}$ and let $x \in V'$. Since $[\![\bigvee_{\{a\} \in I} P_I^{=}]\!]_{G'} = \{a\}$, if $x \neq a$ then $x \in [\![\psi_5]\!]_{G'}$. If $x = a$, we just need to prove that $\underline{a}^{\mathcal{I}}$ (the total counterpart of a) contains the concept A. But this is clear from the fact that $\mathcal{I} \models \mathcal{T}$ and so $a^{\mathcal{I}} \in A^{\mathcal{I}}$. Now, by a similar argument, if $x \in V'$ is a partial node or a total node not containing the concept A, then $x \in [\![\psi_6]\!]_{G'}$. If x contains the concept A, we just need to prove that x is the total counterpart of a. By definition of G', $x = \underline{u}$ for some $u \in \Delta^{\mathcal{I}}$, and since $u \in A^{\mathcal{I}}$, then $u = a^{\mathcal{I}}$, as we wanted.

For item (vi), suppose $Fun(r)$ is in $\mathcal{T}$. We will only show that $[\![\psi_9]\!]_{G'} = V'$, since a similar argument holds for the remaining formulas of the conjunction. If $x \in ind(\mathcal{K})$ or $u \neq \underline{b}^{\mathcal{I}}$ for every $b \in ind(\mathcal{K})$, then it is straightforward that $x \in [\![\psi_9]\!]_{G'}$. If instead $u = \underline{b}^{\mathcal{I}}$ for $b \in ind(\mathcal{K})$, and $r(a, b), r(a, c)$ are assertions in $\mathcal{K}$, then we assert that $b^{\mathcal{I}} = c^{\mathcal{I}}$ since $\mathcal{I} \models \mathcal{K}$. This implies $x \in [\![\psi_9]\!]_{G'}$.

For item (vii) we will prove that, if $[\![\psi_{10}]\!]_{G'} \neq V'$, then $\mathcal{I}$ satisfies the query, i.e. there is a cycle in $\mathcal{I}$ reading a word from the language q. First, notice that for every 2RPQ α, $a \in V$ and $u \in V'$, the pair $(a, u) \in [\![\alpha^{\mathcal{S}}]\!]$ if and only if $(a^{\mathcal{I}}, u) \in [\![\alpha^{\mathcal{S}}]\!]$, and the pair $(u, a) \in [\![\alpha^{\mathcal{S}}]\!]$ if and only if $(u, a^{\mathcal{I}}) \in [\![\alpha^{\mathcal{S}}]\!]$. Since $\downarrow_r \in L'(a, b) \cap L'(a^{\mathcal{I}}, b^{\mathcal{I}})$ for every assertion $r(a, b)$ in $\mathcal{K}$, we conclude that, if $u, v \in \Delta^{\mathcal{I}}$ and $(u, v) \in [\![\alpha^{\mathcal{S}}]\!]$, then $(u, v) \in [\![\alpha^{\mathcal{T}}]\!]$. In other words, if u and v are connected by a path with label in $\alpha^{\mathcal{S}}$, then u and v are also connected by a path that has no edge $\downarrow_{\text{TOTAL}}$. This statement can be easily proved by induction on the structure of α. Suppose now that $[\![\psi_{10}]\!]_{G'} \neq V'$, and let $u \in V'$ be a node for which the constraint ψ_{10} is not valid. By definition, this occurs when $(u, u) \in [\![q^{\mathcal{S}}]\!]_{G'}$. We may assume that $u \in \Delta^{\mathcal{I}}$, thus $(u, u) \in [\![q^{\mathcal{T}}]\!]_{G'}$, which implies that $\mathcal{I}$ has a cycle reading a word from q that contains the node u.

Therefore, G' is a finite data-graph that contains $G(\mathcal{K})$ and satisfies $R(\mathcal{K}, q)$, hence it contains a superset-repair of $G(\mathcal{K})$.

$\Longleftarrow$) Let $G' = (V', L', D')$ be a superset-repair of $G(\mathcal{K})$ with respect to $R(\mathcal{K}, q)$. We will obtain a counter-model for $\mathcal{K}$ and $\exists x.q(x, x)$ by applying an equivalence relation on V'. Let $\sim$ be the relation on V' defined as: $u \sim v$ if and only if $(u, v) \in [\![(\downarrow_{\text{TOTAL}} \cup \downarrow^{-}_{\text{TOTAL}})^*]\!]_{G'}$. We use the Kleene star to guarantee that the relation is transitive and reflexive, and the symmetry is a direct consequence of the definition. In essence, we are collapsing every node having data value P_I with its total counterpart, which we know exists since (i) is satisfied. For every $u \in V'$, we denote by $[u]$ its equivalence class. Let us consider $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\mathcal{I} = V'/\sim$ and $\cdot^{\mathcal{I}}$ is defined over the signature of $\mathcal{K}$ as follows:

- $C^{\mathcal{I}} = \{[u] : \exists I, \exists u' \sim u.\, D'(u') = T_I \text{ and } C \in I\}$ for every concept name C,
- $r^{\mathcal{I}} = \{([u], [v]) : \exists u' \sim u, v' \sim v.\ \downarrow_r \in L'(u', v')\}$ for every role name r,
- $a^{\mathcal{I}} = [a]$ for every individual name a.

Notice that every class $[u]$ contains exactly one total node, and it may contain several (or no) elements from $ind(\mathcal{K})$ and other partial nodes. We denote by $[u]_T$ the total node of the class $[u]$.

We will now show that $\mathcal{I}$ is a counter-model for $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and $\exists x.q(x,x)$. It follows from the definition that $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for every assertion $C(a)$ in $\mathcal{A}$, and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ for every assertion $r(a,b)$ in $\mathcal{A}$.

Suppose that $\bigsqcap_i A_i \sqsubseteq \bigsqcup_j B_j$ is in $\mathcal{T}$, and let $[u] \in \Delta^{\mathcal{I}}$ such that $[u] \in A_i^{\mathcal{I}}$ for every i. For the element $u' = [u]_T$, we know that $D'(u') = T_I$ with $A_i \in I$ for every i. Since G' satisfies the constraint ψ_2 from item (ii), it follows that $B_k \in I$ for some k, which implies that $[u] \in B_k^{\mathcal{I}}$.

Suppose that $A \sqsubseteq \forall r.B$ is in $\mathcal{T}$, and let $[u], [v] \in \Delta^{\mathcal{I}}$ such that $[u] \in A^{\mathcal{I}}$ and $([u],[v]) \in r^{\mathcal{I}}$. Thus, $D'([u]_T) = T_I$ where $A \in I$, and there are nodes u' and v', such that $u' \sim u, v' \sim v$, and $\downarrow_r \in L'(u',v')$. Since G' satisfies ψ_3 from item (iii), if $D'(v') = T_J$ for some J then B must be in J. If instead $D'(v') = P_J$, then $([u]_T, [v]_T) \in [\![(\epsilon \cup \downarrow^{-}_{\text{TOTAL}}) \downarrow_r \downarrow_{\text{TOTAL}}]\!]_{G'}$ and thus $D'([v]_T) = T_{J'}$ for $B \in J'$. Whichever the case may be, we obtain $[v] \in B^{\mathcal{I}}$.

Suppose that $A \sqsubseteq \exists r.B$ is in $\mathcal{T}$, and let $[u] \in \Delta^{\mathcal{I}}$ such that $[u] \in A^{\mathcal{I}}$. Thus $D'([u]_T) = T_I$ where $A \in I$, and since G' satisfies ψ_4 from item (iv), there is a node v such that $D'(v) = T_J$ with $B \in J$ and $([u]_T, v) \in [\![(\epsilon \cup \downarrow^{-}_{\text{TOTAL}}) \downarrow_r (\epsilon \cup \downarrow_{\text{TOTAL}})]\!]_{G'}$. This implies that $([u],[v]) \in r^{\mathcal{I}}$ and $[v] \in B^{\mathcal{I}}$.

Suppose that $A \equiv \{a\}$ is in $\mathcal{T}$, and let $[u] \in \Delta^{\mathcal{I}}$ such that $[u] \in A^{\mathcal{I}}$. Hence, $D'([u]_T) = T_I$ where $A \in I$, and since G' satisfies ψ_5 and ψ_6 from item (v), it follows that $[u] = a^{\mathcal{I}}$.

Suppose that $Fun(r)$ is in $\mathcal{T}$ and $([u],[v]),([u],[w]) \in r^{\mathcal{I}}$. We need to prove that $[v] = [w]$ or, equivalently, that $[v]_T = [w]_T$. By hypothesis, there exist $u' \sim u, u'' \sim u, v' \sim [v]_T$ and $w' \sim [w]_T$ such that $\downarrow_r \in L'(u',v')$ and $\downarrow_r \in L'(u'',w')$. There are several cases to analyse. If $u' = u''$, $v' = [v]_T$ and $w' = [w]_T$, it follows that $v' = w'$ from the satisfaction of formula ψ_7 from item (vi). For the remaining cases, consider that every node that is different from its total counterpart is connected to it by a total edge. As a consequence of this fact and the satisfaction of formulas ψ_8 and ψ_9, we obtain that the statement holds for the remaining cases.

Finally, we have to check that $\mathcal{I}$ does not satisfy the query. Toward a contradiction, let $[u] \in \Delta^{\mathcal{I}}$ such that, for a cycle in $\mathcal{I}$ containing $[u]$, the label of the path starting and finishing in $[u]$ reads a word on the language q. We will prove that there is a cycle in G' reading a word from the language $q^{\mathcal{S}}$. Moreover, this cycle contains $[u]_T$ and the word read starts and finishes in this node. Suppose the cycle in $\mathcal{I}$ is

$$([u_0], r_1, [u_1]), ([u_1], r_2, [u_2]), \ldots, ([u_{n-1}], r_n, [u_n]),$$

where $[u_0] = [u_n] = [u]$ and $\omega = r_1 r_2 \cdots r_n \in q$. The triplet (v, α, w) means $(v,w) \in [\![\alpha]\!]$. Then,

$$([u_0]_T, r_1^{\mathcal{S}}, [u_1]_T), ([u_1]_T, r_2^{\mathcal{S}}, [u_2]_T), \ldots, ([u_{n-1}]_T, r_n^{\mathcal{S}}, [u_n]_T)$$

is a cycle in G'. Since any word in $r_1^{\mathcal{S}} r_2^{\mathcal{S}} \cdots r_n^{\mathcal{S}}$ is also a word in $q^{\mathcal{S}}$ we obtain that G' cannot satisfy ψ_{10} since $([u]_T, [u]_T) \in [\![q^{\mathcal{S}}]\!]_{G'}$.

□

As pointed out in Remark 16, simple data tests may prevent the existence of superset repairs. Observe that in the absence of them, any data-graph G has a superset repair if we consider only the positive fragment of Reg-GXPath: add every possible edge label $l \in \Sigma_e$ to every pair of nodes $v, v' \in V_G$ and the resulting graph will satisfy any expression. The proof of this fact follows quite straightforward by induction in the expression's structure.

Before proceeding, we define some concepts related to data values:

Definition 20. Let η be a Reg-GXPath expression. We define the set of data values present in η as the set of all those $c \in \Sigma_n$ such that the subexpression $[c^=]$ or $[c^{\neq}]$ is used in η. We denote it as $\boldsymbol{\Sigma_n^\eta}$.

Analogously, we define the set of data values used by a set of Reg-GXPath expressions R as $\mathbf{\Sigma_n^R} = \bigcup_{\eta \in R} \Sigma_n^\eta$.

We also denote the set of data values used in a graph G as $\boldsymbol{\Sigma_n^G} = \{D_G(v) : v \in V_G\}$.

Even though Σ_n may be infinite, when considering only Reg-GXPath^{pos} expressions we obtain the following lemma:

Lemma 21. *Let G be a data-graph and R a set of Reg-GXPath^{pos} expressions. If there is a superset repair G' of G with respect to the constraints R, then there is another superset repair H that only uses data values from $\Sigma_n^R \cup \Sigma_n^G$ plus at most two extra data values not mentioned in R.*

Proof. Intuitively, values that are not mentioned in R are not necessary to satisfy R, and therefore, if there exists a repair, there must be one with only data values from R and the original data values from G. Nonetheless, it could be the case that every data value mentioned in R is in an expression of the form $[c^{\neq}]$, and in that case we might need an extra data value (for example, consider the expression $\varphi = \langle\gamma\rangle$ where $\gamma = \downarrow_x [\bigvee_{c \in \mathcal{D}} c^{\neq}]$ for $\mathcal{D}$ a set of data values). Actually, we might need two fresh data values to satisfy some expression of the form $\langle \alpha \neq \beta \rangle$ (replace $\alpha = \beta = \gamma$). See Appendix A for the detailed proof.

□

The following observation is a consequence of the previous lemma:

Observation 22. *If there is a superset repair G with respect to R, then there exists a superset repair with a number of different data values that linearly depends on $|G| + |R|$.*

Notice that this observation does not imply that there must be a superset repair with linear size on $|G| + |R|$: it could be the case that there is an exponential number of nodes having the same data value. However, we could somehow 'merge' all those nodes with the same data value while preserving some edges, such that the resulting graph will still satisfy all those Reg-GXPath^{pos} expressions that were satisfied in the original graph:

Lemma 23. *Let G be a data-graph that satisfies a Reg-GXPath^{pos} restriction η. Let V_d be a set of nodes from G having the same data value d. If we define a new data-graph H where*

$$\begin{aligned}
V_H &= (V_G \setminus V_d) \cup \{v_d\} \\
L_H(v,w) &= L_G(v,w) \forall v,w \in V_G \setminus V_d \\
L_H(v,v_d) &= \{e \in \Sigma_n \mid \exists w \in V_d \textit{ such that } e \in L_G(v,w)\}, \forall v \in V_G \setminus V_d \\
L_H(v_d,v) &= \{e \in \Sigma_n \mid \exists w \in V_d \textit{ such that } e \in L_G(w,v)\}, \forall v \in V_G \setminus V_d \\
L_H(v_d,v_d) &= \{e \in \Sigma_n \mid \exists w_1,w_2 \in V_d \textit{ such that } e \in L(w_1,w_2)\} \\
D_H(v) &= D_G(v), \forall v \in V_G \setminus V_d \\
D_H(v_d) &= d
\end{aligned}$$

Then, H satisfies η.

Proof. Expressions from Reg-GXPath^{pos} can only interact with data values and edge labels, ignoring the actual identity of the nodes[3]. Then, observe that in the data-graph just defined the neighbourhoods of all nodes are the same as in data-graph G, except that we might have collapsed some set of nodes. But for those nodes collapsed, we created a new one with the same data value and the same (or even bigger) neighbourhood. Finally, the positive expression of Reg-GXPath^{pos} cannot really distinguish this new node from the previous ones. See Appendix A for the actual proof.

□

As it was mentioned previously, when we define the graph H the set of nodes V_d is collapsed into a unique node v_d while preserving all the edges that were incident to the set V_d. This operation is usually called vertex contraction. Using Lemmas 21 and 23, we prove the following very useful fact:

Theorem 24. *Let G be a data-graph and R a set of Reg-GXPath^{pos} expressions. If there is a superset repair of G with respect to R, then there is a superset repair of G with respect to R of polynomial size depending on $|G| + |R|$.*

Proof. We define a data-graph H such that $G \subseteq H$, $H \models R$, and the size of H polynomially depends on $|G| + |R|$. This suffices to prove the theorem.

Let G' be a superset repair of G with respect to R that uses at most $|G| + |R| + 2$ data values. Such a superset repair exists as a consequence of Lemma 21 and the hypothesis. Thus, for every data value $c \in \Sigma_n^{G'} \setminus \Sigma_n^{G}$, we contract all the nodes having data value c to a unique node. Let H' be the graph obtained once those nodes were contracted. Notice that H' satisfies R due to Lemma 23.

We contract every other node $v \in V_{G'} \setminus V_G$ in H' that has a data value from Σ_n^G with a node from G having the same data value. The resulting graph is indeed the H we are looking for, since once again due to Lemma 23, H satisfies R and $|V_H| = |V_G| + |\Sigma_n^{G'} \setminus \Sigma_n^G| \leq |V_G| + |R| + 2$. Therefore, $|H| \leq (|V_G| + |R| + 2)^2$, which is polynomial on G and R.

□

3. This is not entirely true, since the path expression ϵ can be used to relate a node with only itself.

This last theorem shows that the problem $\exists$SUPERSET-REPAIR lies in NP for Reg-GXPath^{pos} expressions, since we can use the repair itself as a witness for a positive instance. Moreover, given a data-graph G, we can use the same proof to define a polynomial-time algorithm w.r.t. the size of G that computes a superset repair, if there exists one. More precisely, if there exists a superset repair of G with respect to R, then there is a 'small' graph H such that $H \models R$ and H has a very precise structure: the only nodes that are added to G in order to obtain H have a one-to-one correspondence with those new data values that were not present in G. From now on we will refer to this structure as the *standard form of a superset repair.* Hence, we may find a minimal –with respect to node inclusion– data-graph H' that satisfies $H' \models R$ and $G \subseteq H'$ as follows: Iterate over every possible subset S of $(\Sigma_n^R \cup \{c, d\}) \setminus \Sigma_n^G$, where c and d are the 'fresh' data values mentioned in Lemma 21. Then, add one node to G for each data value in S and every possible edge between any pair of nodes. Finally, check if the resulting data-graph satisfies R.

If none of these data-graphs satisfies R, then it follows from Theorem 24 that there is no superset repair. Otherwise, after computing the minimal –with respect to node inclusion– data-graph H', we may find a repair by deleting edges from H': if once we delete an edge e from H' we notice that R is not satisfied anymore, then it follows from Lemma 13 that there is no subset H'' of H' that does not contain the edge e such that H'' satisfies R and such that H'' is a data-graph that still contains G. This allows us to assert that e will belong to the final repair.

Considering all the previous discussions, we design Algorithm 2. It relies on an auxiliary procedure *buildGraph(G, S)* that given a data-graph $G = (V, L, D)$ and a set of data values S not in G builds the "candidate" repair for that set of data values, defined as $\mathit{buildGraph}(G, S) = (V', L', D')$ where $V' = V \cup \{v_s : s \in S\}$, $L'(v, w) = \Sigma_e$ for $v, w \in V'$ and $D'(v) = D(v)$ if $v \in V$ whereas $D(v_s) = s$. Clearly, $\mathit{buildGraph}(G, S)$ can be computed in $O((|V| + |S|)^2)$

The first **for** will be executed at most $2^{\Sigma_n^R + 2}$ times. If Σ_n is finite, then this number of executions is constant. Otherwise, it can depend exponentially on $|R|$. Everything inside the loop can be computed in polynomial time thanks to Theorem 10. The second loop will run at most $(|V_G| + \Sigma_n^R)^2$ times, which is quadratic in respect to the input size. Since the procedure is correct, we obtain the final result:

Corollary 25. *There is an algorithm such that, given a data-graph G and a set R of Reg-GXPath^{pos} expressions, if Σ_n is finite, then the algorithm computes a superset repair of G with respect to R in polynomial time, if such a repair exists.*

If Σ_n is infinite, then we can still compute a superset repair in polynomial time if R is fixed.

Note that if R only contains positive node expressions then we can use a similar procedure as in Algorithm 1: we start by building the data-graph $H = \mathit{buildGraph}(G, S)$ where $S = \Sigma_n^R \cup \{c, d\} \setminus \Sigma_n^G$ and then we iteratively remove vertices that do not satisfy some node expression $\phi \in R$. By Theorem 24 we know that if there exists a superset repair, then there must exist some superset repair G' such that $G \subseteq G' \subseteq H$. Also, if $v \notin [\![\phi]\!]_H$ for some v then $v \notin G'$. Therefore, in at most $|V_H \setminus V_G| \leq |R| + 2$ iterations we will find a data-graph that contains the same set of nodes as some superset repair of G with respect to R (if such

Algorithm 2 *SupersetRepair*(G, R)

```
Require: $G$ is a data-graph and $R$ a set of Reg-GXPath$^{pos}$ expressions.
1: for $S \in \mathcal{P}(\Sigma_n^R \cup \{c, d\} \setminus \Sigma_n^G)$ do
2:     $H \leftarrow buildGraph(G, S)$
3:     if $H \models R$ and $H \subseteq H'$ then
4:         $H' \leftarrow H$
5:     end if
6: end for
7: if $H'$ is not initialized then
8:     return 'There is no superset repair'
9: end if
10: for $e \in E_{H'} \setminus E_G$ do
11:     if $(V_{H'}, E_{H'} \setminus \{e\}, D_{H'}) \models R$ then
12:         $H' \leftarrow (V_{H'}, E_{H'} \setminus \{e\}, D_{H'})$
13:     end if
14: end for
15: return $H'$
```

repair exists). Then, we only need to remove the edges to reach the minimality condition. This yields the following result:

Corollary 26. *The* ∃SUPERSET-REPAIR *problem can be solved in polynomial-time in combined complexity if R only contains node expressions from Reg-GXPath^{pos}.*

If we do not impose any of these restrictions the problem is once again intractable in general:

Theorem 27. *If the set of Reg-GXPath^{pos} expressions R is not fixed and Σ_n is infinite, then the problem* ∃SUPERSET-REPAIR *is* NP-complete.

Proof. We reduce 3-SAT to an instance of ∃SUPERSET-REPAIR, where $|R|$ depends on the input formula of 3-SAT and we rely on Σ_n being infinite. We take $\Sigma_e \supseteq \{\textsc{down}\}$ and $\Sigma_n \supseteq \{x_i \mid i \in \mathbb{N}\} \cup \{\neg x_i \mid i \in \mathbb{N}\} \cup \{null\}$.

Given the 3-CNF formula ϕ with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$ we define the data-graph $G = (V, L, D)$ where:

$$\begin{aligned} V_G &= \{v\} \\ L(v, v) &= \emptyset \\ D(v) &= null \end{aligned}$$

We would like that every superset repair of G represents a valuation (or at least a partial valuation) on the variables x_i. Intuitively, given a superset repair G', a variable x_i will evaluate to $\top$ if there is a node in G' with data value x_i, and will evaluate to $\bot$ if there is a node in G' with data value $\neg x_i$.

In order to obtain valid valuations, we need to avoid the scenario where both data values x_i and $\neg x_i$ are present in a repair. To do this, we define the following constraints:

$$\alpha_i = ([x_i^=] \downarrow_{\textsc{down}}^* [\neg x_i^{\neq}]) \cup ([x_i^{\neq}] \downarrow_{\textsc{down}}^* [\neg x_i^=]) \cup ([x_i^{\neq}] \downarrow_{\textsc{down}}^* [\neg x_i^{\neq}])$$

where i iterates over $1 \leq i \leq n$. α_i cannot be satisfied by any graph that has nodes with data values x_i and $\neg x_i$. This follows from the fact that, for any of the three subexpressions, the path cannot begin and end in nodes having data values x_i and $\neg x_i$ respectively.

Then, we add the following constraint to ensure that, for every repair, the defined valuation evaluates the clauses to true:

$$\beta_j = (\downarrow^*_{\text{DOWN}} [(c^j_1)^=] \downarrow^*_{\text{DOWN}}) \cup (\downarrow^*_{\text{DOWN}} [(c^j_2)^=] \downarrow^*_{\text{DOWN}}) \cup (\downarrow^*_{\text{DOWN}} [(c^j_3)^=] \downarrow^*_{\text{DOWN}})$$

where j iterates over $1 \leq j \leq m$ and c^j_k denotes the kth literal appearing in c_j, either x_i or $\neg x_i$, for some i. Any repair of G will satisfy all β_j, which implies that in any repair there must be, for every clause, one node whose data value represents a variable assignation that satisfies the clause.

In summary, we defined $R = \{\alpha_i \mid 1 \leq i \leq n\} \cup \{\beta_j \mid 1 \leq j \leq m\}$. Now we show that ϕ is satisfiable if and only if G has a superset repair with respect to R.

$\Longrightarrow$) Let f be a valuation of the variables of ϕ that evaluates ϕ to true. We define the data-graph $H = (V_H, L_H, D_H)$ as:

$$\begin{aligned}
V_H &= \{v\} \cup \{x \mid 1 \leq i \leq n \text{ and } f(x_i) = \top\} \cup \{\neg x_i \mid 1 \leq i \leq n \text{ and } f(x_i) = \bot\} \\
L(a, b) &= \Sigma_e, \text{ for every pair } a, b \in V_H \\
D(v) &= \mathit{null} \\
D(x_i) &= x_i, \text{ for every node } x_i \in V_H \\
D(\neg x_i) &= \neg x_i, \text{ for every node } \neg x_i \in V_H
\end{aligned}$$

Every expression α_i is satisfied, since there are no pairs of nodes with 'opposite' data values and the graph is fully connected. The expressions β_j are also satisfied given that for every clause c_j, at least one of the variables that evaluates the clause to true is present in H. Since $G \subseteq H$ and $H \models R$ there exists a superset repair G' of G with respect to R.

$\Longleftarrow$) Let G' be a superset repair of G with respect to R. We define a valuation on the variables of ϕ by considering the data values present in G'. If the data value x_i is present then $f(x_i) = \top$, and we define $f(x_i) = \bot$ otherwise. Observe that since α_i is satisfied for every i this is a valid valuation. Moreover, notice that since β_j is satisfied for every j, this valuation makes every clause evaluate to true. Let us consider an arbitrary clause c_j. Since β_j is satisfied in G', one of the literals x^j_k for $k \in \{1, 2, 3\}$ has to appear as data value in G'. If one of those literals that appear is positive (i.e. without negation) then it evaluates to true by f, and thus c_j is satisfied. Otherwise, there is a literal with negation from c_j as data value in G', and it evaluates to true through f, and thus c_j evaluates to true. This shows that ϕ is satisfiable.

□

4. Related Work

Models for graph databases and knowledge graphs have been developed intensively since the 1990s, along with query languages and integrity constraints for them (Angles & Gutierrez, 2008; Barceló, 2013). During the last years they have earned significant attention from industry and academy due to their efficiency when modeling diverse, dynamic and large-scale collections of data (Hogan et al., 2021). There are open source knowledge graphs such as YAGO (Rebele et al., 2016) or DBPEDIA (Lehmann et al., 2015) that implement many features studied by the research community.

Reg-GXPath is a query language developed for graph databases with data values in the nodes (Libkin et al., 2016) largely inspired by *XPath* (Benedikt & Koch, 2009), a language for traversing data trees (i.e. XML documents). The main results concerning the expressiveness of this language and its relation to other query languages can be found at (Libkin et al., 2016), as well as a detailed analysis on several subfragments of it. As we already mentioned, our notion of consistency and the problems we defined can be understood under any other navigational language, such as those studied at (Angles et al., 2018; Barceló, 2013; Francis et al., 2018; van Rest et al., 2016). For an analysis of the different features relevant for a modern graph query language you may see (Angles et al., 2017).

Different types of integrity and path constraints have been defined and studied in this context (Abiteboul & Vianu, 1999; Barceló & Fontaine, 2017; Buneman et al., 2000). There is still no standard definition of *consistency* for a graph database over some constraints, and therefore we developed our own based on a set of typical examples found in the literature. Other type of graph database constraints can be expressed through graph patterns and graph dependencies (Fan, 2019).

Our definition of consistency allows to globally restrict the structure of the data-graph, since we require the Reg-GXPath expressions to be satisfied in every node and pair of nodes. Therefore, we can express stronger requirements than those allowed for constraints based on an origin (which was the original proposal of Abiteboul and Vianu, see (Calvanese et al., 2016) or (Barceló & Fontaine, 2017) for recent work around these semantics). Also, since negation is included into the grammar of Reg-GXPath it is possible to build constraints similar to "path implications" such as those considered in (Abiteboul & Vianu, 1999) or (Barceló & Fontaine, 2017) without having to define the notion of implication from outside the language.

The notion of *database repair* and the *repair computing* problem, as well as the *Consistent Query Answering* (CQA) problem, were first introduced in the relational context (Arenas et al., 1999). Since then, CQA has received much attention from the research community in data management, developing techniques and efficient algorithms under different notions of repairs and considering all kinds of combinations of classes of integrity constraints and queries (Bertossi, 2011, 2019). These concepts were successfully extended to diverse data models such as *XML* or *description logics* (Arenas & Libkin, 2008; Gheerbrant et al., 2012; Lembo et al., 2015; Lukasiewicz et al., 2015). In the case of graph databases there has been some work concerning CQA over graphs without data values (Barceló & Fontaine, 2017), however, to the best of our knowledge, no previous work studies the problem of computing repairs neither the case when the graphs have data values in their nodes.

When studying CQA and repair problems in contexts such as description logics, it seems more natural to focus on subset repairs, since DL semantics is *open world* by nature. This means that the (explicit or implicit) non-presence of a fact in the database is not enough to derive the negation of the fact. In the case of graph databases, some applications can respond either to *closed* or *open world* semantics, and therefore both types of set-based repairs could turn out to be meaningful. In this work, we considered both types of set-based repairs, however we note that we left for future work the study of *symmetric difference*-based repairs –which were actually studied in (Barceló & Fontaine, 2017). Usually, reasoning on symmetric difference repairs is much harder, and, in the case of data-graphs, the definition of *symmetric difference* is not as straightforward to formalize as it is in the models defined in (Barceló & Fontaine, 2017).

5. Conclusions

In this work, we presented a graph database model along with a notion of consistency based upon path constraints defined by the Reg-GXPath language. We proved many results concerning the complexity of computing subset and superset repairs of data-graphs given a set of restrictions (which can be summarized in Table 1). We proved that, depending on the semantics of the repair and the syntactic conditions imposed on the set of constraints, the problem ranges from polynomial-time solvable to undecidable.

When restricting the language to the positive fragment of Reg-GXPath, which we denote Reg-GXPath^{pos}, we obtained polynomial-time algorithms (on data complexity) for superset-repair computing, and when considering only node expressions from Reg-GXPath^{pos} we found a polynomial-time algorithm for computing the unique subset repair. We note that the algorithm for subset repairs runs in polynomial time on both the size of the graph G and the size of the restriction set R (in other words, we consider *combined complexity* of the problem). Furthermore, for every intractable case studied, we also obtained tight bounds on the combined and data complexity of the problem.

Some questions regarding repair computing in this context remain open. For example, if the ∃SUPERSET-REPAIR problem remains undecidable when considering only a fixed set of node expressions from Reg-GXPath. It does not seem direct to extend the same proofs and techniques used for the case of path expressions or node expressions (under unfixed constraints).

Another direction of future research we are interested in is the development of preference criteria over repairs. In the problems we studied, we were concerned in finding *any* repair of a data-graph given a set of constraints. This assumes no prior information about the domain semantics. However, in many contexts we might have a preference criterion that may yield an ordering over all possible repairs.

The complexity of CQA under this context has not been studied yet. Taking into account the results from (Ten Cate et al., 2015) and (Barceló & Fontaine, 2017) it is plausible that reasoning over superset repairs will turn out to be harder than in subset repairs, but meanwhile the complexity of the problem remains unknown. Nevertheless, we note that the CQA problem turns out to be trivial in some cases based on the facts developed through this work concerning Reg-GXPath and Reg-GXPath^{pos}; for example, when considering subset repairs and Reg-GXPath^{pos} there is a unique repair which is computable in polynomial time, and therefore the CQA problem is easy to solve.

	Subset	Superset
Reg-GXPath	NP-complete, even for a fixed set of node constraints (Th. 12)	Undecidable for fixed set of path expressions (Th. 17) and for an unfixed set of node expressions (Th. 19)
Reg-GXPath^{pos}	PTime for node constraints. (Th. 15) NP-complete in data complexity for path constraints (Th. 11)	PTime for any fixed constraints (Th. 25) or for an unfixed set of node expressions (Th. 26). NP-complete otherwise (Th. 27)

Table 1: Summary of the results obtained for the subset and superset repair problems.

Acknowledgments

This work was funded in part by Secretaría de Investigación Científica y Tecnológica FCEN–UBA (RESCS-2020-345-E-UBA-REC), CONICET under the PIP (grant 11220200101408CO), Agencia Nacional de Promoción Científica y Tecnológica, Argentina under grants PICT-2018-0475 (PRH-2014-0007) and PICT-2020-SERIEA-01481. We thank Dr. Ricardo O. Rodriguez (Department of Computer Science, Universidad de Buenos Aires) for his helpful comments and suggestions during the development of this work.

Appendix A.

Proof of Lemma 13. We will prove it by induction over the structure of the expressions. Let us note $G = (V_G, L_e, D)$ and $G' = (V_{G'}, L_{e'}, D')$.

In the base case of a path expression α, it must be of the form ϵ, $\_$, A or A^-. In all of these cases the property holds: since $G \subseteq G'$, we have that $\{(x,x) \mid x \in V_G\} \subseteq \{(x,x) \mid x \in V_{G'}\}$, that $\{(x,y) \mid x,y \in V_G \wedge L_e(x,y) \neq \emptyset\} \subseteq \{(x,y) \mid x,y \in V_{G'} \wedge L_{e'}(x,y) \neq \emptyset\}$, etc.

For the base case of a node expression φ, it can only be of the form $\text{c}^=$ or $\text{c}^{\neq}$, and it is easy to see that the property also holds.

For the inductive step, we consider all possible cases:

- If $\alpha = [\psi]$ then $[\![\psi]\!]_G \subseteq [\![\psi]\!]_{G'}$ (which holds by inductive hypothesis) implies $[\![[\psi]]\!]_G \subseteq [\![[\psi]]\!]_{G'}$.
- If $\alpha = \beta_1.\beta_2$ then $(v,w) \in [\![\alpha]\!]_G \iff \exists z$ such that $(v,z) \in [\![\beta_1]\!]_G$ and $(z,w) \in [\![\beta_2]\!]_G$. Since $[\![\beta_i]\!]_G \subseteq [\![\beta_i]\!]_{G'}$ for $i \in \{1,2\}$, then $(v,z) \in [\![\beta_1]\!]_{G'}$ and $(z,w) \in [\![\beta_2]\!]_{G'}$, which implies $(v,w) \in [\![\alpha]\!]_{G'}$.
- If $\alpha = \beta_1 \cup \beta_2$ then $[\![\beta_i]\!]_G \subseteq [\![\beta_i]\!]_{G'}$ for $i \in \{1,2\}$ readily implies $[\![\alpha]\!]_G \subseteq [\![\alpha]\!]_{G'}$.
- If $\alpha = \beta^*$ then $(v,w) \in [\![\beta^*]\!]_G \iff \exists z_1, z_2, ..., z_m$ such that $z_1 = v$, $z_m = w$ and $(z_i, z_{i+1}) \in [\![\beta]\!]_G$ for $1 \leq i < m$. Since $[\![\beta]\!]_G \subseteq [\![\beta]\!]_{G'}$ then $(z_i, z_{i+1}) \in [\![\beta]\!]_{G'}$ for $1 \leq i < m$, which implies $(v,w) \in [\![\beta^*]\!]_{G'}$.

- If $\alpha = \beta^{n,m}$ then $(v,w) \in [\![\beta^{n,m}]\!]_G \iff \exists z_1, ..., z_k, n+1 \leq k \leq m+1$ such that $z_1 = v$, $z_k = w$ and $(z_i, z_{i+1}) \in [\![\beta]\!]_G$ for $1 \leq i \leq k-1$. By the inductive hypothesis then $(z_i, z_{i+1}) \in [\![\beta]\!]_{G'}$, which implies $(z,w) \in [\![\alpha^{n,m}]\!]_{G'}$.
- If $\alpha = \beta_1 \cap \beta_2$ then $[\![\beta_i]\!]_G \subseteq [\![\beta_i]\!]_{G'}$ for $i \in \{1,2\}$ implies $[\![\beta_1 \cap \beta_2]\!]_G = [\![\beta_1]\!]_G \cap [\![\beta_2]\!]_G \subseteq [\![\beta_1]\!]_{G'} \cap [\![\beta_2]\!]_{G'} = [\![\beta_1 \cap \beta_2]\!]_{G'}$.
- If $\varphi = \psi_1 \wedge \psi_2$ then $[\![\psi_i]\!]_G \subseteq [\![\psi_i]\!]_{G'}$ for $i \in \{1,2\}$ readily implies $[\![\psi_1 \wedge \psi_2]\!]_G = [\![\psi_1]\!] \cap [\![\psi_2]\!] \subseteq [\![\psi_1]\!]_{G'} \cap [\![\psi_2]\!]_{G'} = [\![\psi_1 \wedge \psi_2]\!]_{G'}$
- The case $\varphi = \psi_1 \vee \psi_2$ can be treated in the same way.
- If $\varphi = \langle\beta\rangle$ then $[\![\beta]\!]_G \subseteq [\![\beta]\!]_{G'}$ implies $[\![\langle\beta\rangle]\!]_G \subseteq [\![\langle\beta\rangle]\!]_{G'}$
- If $\varphi = \langle\beta_1 \star \beta_2\rangle$, with $\star \in \{=, \neq\}$, then $v \in [\![\varphi]\!]_G \iff \exists z_1, z_2 \in V_G$ such that $(v, z_i) \in [\![\beta_i]\!]_G$ for $i \in \{1,2\}$ and $D(z_1) \star D(z_2)$. Since $[\![\beta_i]\!]_G \subseteq [\![\beta_i]\!]_{G'}$, then $(v, z_i) \in [\![\beta_i]\!]_{G'}$. And since we also have that $D(z_i) = D'(z_i)$ (since the z_i are in V_G), we obtain that $v \in [\![\langle\beta_1 \star \beta_2\rangle]\!]_{G'}$, as we wanted.

□

Proof of Lemma 21. In order to prove this Lemma we need to first prove another property of Reg-GXPath^{pos} expressions:

Lemma 28. *Let α be a Reg-GXPath^{pos} path expression, φ a Reg-GXPath^{pos} node expression, $E \subseteq \Sigma_e$ a set of edge labels and K a data-graph where for every pair of nodes $v, w \in V_K$ it is the case that $L(v,w) = E$. Then it holds that:*

1. *If $(v,w) \in [\![\alpha]\!]_K$, $v \neq w$, and $D_K(w) \notin \Sigma_n^\alpha$, then we have that $(v,z) \in [\![\alpha]\!]_K$ for every node $z \in V_K$ such that $D_K(z) \notin \Sigma_n^\alpha$.*
2. *If $(w,v) \in [\![\alpha]\!]_K$, $v \neq w$, and $D_K(w) \notin \Sigma_n^\alpha$, then we have that $(z,v) \in [\![\alpha]\!]_K$ for every node $z \in V_K$ such that $D_K(z) \notin \Sigma_n^\alpha$.*
3. *If $(v,v) \in [\![\alpha]\!]_K$ and $D_K(v) \notin \Sigma_n^\alpha$ then $(z,z) \in [\![\alpha]\!]_K$ for every node $z \in V_K$ such that $D_K(z) \notin \Sigma_n^\alpha$.*
4. *If $v \in [\![\varphi]\!]_K$ for some $v \in V_K$ such that $D_K(v) \notin \Sigma_n^\varphi$ then $z \in [\![\varphi]\!]_K$ for every node $z \in V_K$ that satisfies $D_K(z) \notin \Sigma_n^\varphi$.*

Proof of Lemma 28. We present a proof of the four properties simultaneously by structural induction. For the base cases:

- $\alpha = \_$: if $E \neq \emptyset$ then every pair $v, z \in V_K$ is in $[\![\_]\!]_K$, and this case follows trivially. Otherwise there are no edges in the graph, and there isn't any hypothesis that can be satisfied.
- $\alpha = \textsc{a}$ and $\alpha = \textsc{a}^-$: if there is a pair of nodes $v, w \in V_K$ such that $(v,w) \in [\![\textsc{a}]\!]_K$ then $\textsc{a} \in E$, and every pair of nodes $x, y \in V_K$ satisfies $(x,y) \in [\![\textsc{a}]\!]_K$. The same reasoning holds for $\textsc{a}^-$.

- $\alpha = \epsilon$: this case follows trivially, since the hypothesis of the statements 1 and 2 cannot be satisfied, and the consequent of statement 3 is always satisfied.

- $\varphi = \text{E}^{=}$: the hypothesis of statement 4 cannot be satisfied in this case, and then this follows trivially.

- $\varphi = \text{E}^{\neq}$: every node $v \in V_K$ such that $D_K(v) \notin \Sigma_n^{\text{E}^{\neq}} = \{\text{E}\}$ satisfies $v \in [\![\text{E}^{\neq}]\!]_K$.

We now proceed with the inductive cases:

- $\alpha = [\psi]$: the hypothesis of statements 1 and 2 cannot be satisfied in this case, and therefore we only consider statement 3. Suppose there is a node $v \in V_K$ such that $D_K(v) \notin \Sigma_n^{[\psi]}$ and $(v, v) \in [\![[\psi]]\!]_K$. By hypothesis, every other node $z \in V_K$ such that $D_K(z) \notin \Sigma_n^{\psi}$ satisfies $z \in [\![\psi]\!]_K$, which then implies that $(z, z) \in [\![[\psi]]\!]_K$.

- $\alpha = \beta_1.\beta_2$: for the statement 1, let $v, w \in V_K$ satisfy its hypothesis ($(v, w) \in [\![\alpha]\!]_K$, $v \neq w$, $D(w) \notin \Sigma_n^{\alpha}$), and let x be a node such that $(v, x) \in [\![\beta_1]\!]_K$ and $(x, w) \in [\![\beta_2]\!]_K$. Then if $x \neq w$ we know by the inductive hypothesis that $(x, z) \in [\![\beta_2]\!]_K$ for every $z \in V_K$ with $D_K(z) \notin \Sigma_n^{\beta_2}$ (which is a superset of those nodes y such that $D_K(y) \notin \Sigma_n^{\beta_1.\beta_2}$), and then we can conclude that $(v, z) \in [\![\beta_1.\beta_2]\!]_K$. If $w = x$ the result follows by considering that $(v, z) \in [\![\beta_1]\!]_K$ and then $(z, z) \in [\![\beta_2]\!]_K$ by the statement 3. The statement 2 can be proved the same way.

  Now for statement 3 let v be a node from K that satisfies its hypothesis ($(v, v) \in [\![\alpha]\!]_K$ and $D_K(z) \notin \Sigma_n^{\alpha}$) and let x be the 'intermediate' node. Let z be a node such that $D_K(z) \notin \Sigma_n^{\beta_1.\beta_2}$. If $x = v$ it follows by using statement 3 as inductive hypothesis that $(z, z) \in [\![\beta_1.\beta_2]\!]_K$. Otherwise (i.e. $x \neq v$) by using statement 1 and 2 as inductive hypothesis we can obtain the same result.

- $\alpha = \beta_1 \cup \beta_2$: this follows by inductive hypothesis considering, without loss of generality, that $(v, w) \in [\![\beta_1]\!]_K$ (and respectively (w, v) and (v, v) for statements 2 and 3).

- The case $\alpha = \beta_1 \cap \beta_2$ can be treated in a similar way.

- $\alpha = \beta^*$: if $(v, w) \in [\![\beta^*]\!]_K$ then either $(v, w) \in [\![\epsilon]\!]_K$ or $(v, w) \in [\![\beta^n]\!]_K$ for some $n > 0$. The first scenario has already been proved, while the second one follows from what we have seen in the concatenation case.

- If $\alpha = \beta^{n,m}$: the result follows with the same argument used in the β^* case since $(v, w) \in [\![\beta^{n,m}]\!] \iff (v, w) \in [\![\beta^k]\!]$ for some $n \leq k \leq m$.

- $\varphi = \psi_1 \wedge \psi_2$: let $v \in V_K$ be a node such that $v \in [\![\varphi]\!]_K$ and $D_K(v) \notin \Sigma_n^{\varphi}$. By inductive hypothesis, every other $z \in V_K$ with $D_K(z) \notin \varphi$ satisfies $z \in [\![\psi_i]\!]_K$ for $i \in \{1, 2\}$, which implies that $z \in [\![\varphi]\!]_K$.

- $\varphi = \psi_1 \vee \psi_2$: let $v \in V_K$ satisfy the hypothesis of statement 4, and without loss of generality assume $v \in [\![\psi_1]\!]_K$. By inductive hypothesis every other node $z \in V_K$ such that $D_K(z) \notin \Sigma_n^{\varphi}$ satisfies $z \in [\![\psi_1]\!]_K$, which implies that $z \in [\![\psi_1 \vee \psi_2]\!]_K$.

- $\varphi = \langle \beta \rangle$: let $v \in V_K$ satisfy the hypothesis of statement 4. Then $(v, x) \in [\![\beta]\!]_K$ for some x. Thanks to statement 2 we can deduce that if $x \neq v$ then $(z, x) \in [\![\beta]\!]_K$ for every z such that $D_K(z) \notin \Sigma_n^{\langle \beta \rangle}$. If $x = v$ we use statement 3.
- $\varphi = \langle \beta_1 = \beta_2 \rangle$: let $v \in V_K$ satisfy the hypothesis of statement 4, x_1 and x_2 be nodes such that $(v, x_i) \in [\![\beta_i]\!]_K$ for $i \in \{1, 2\}$ and $D_K(x_1) = D_K(x_2)$, and $z \in V_K$ other node such that $D_K(z) \notin \Sigma_n^{\varphi}$. If both x_i are different from v then $(z, x_i) \in [\![\beta_i]\!]_K$ thanks to statement 2. Otherwise, suppose $x_1 = v$ without loss of generality, which then implies that $(z, z) \in [\![\beta_1]\!]_K$ due to statement 3. Since $D_K(x_2) = D_K(x_1) = D_K(v) \notin \Sigma_n^{\varphi}$ we can guarantee that $(z, z) \in [\![\beta_2]\!]_K$: if $x_2 = v$ then this follows due to statement 3, and if $x_2 \neq v$ then due to statement 2 we know that $(x_2, x_2) \in [\![\beta_2]\!]_K$ and then that $(z, z) \in [\![\beta_2]\!]_K$ through statement 3.
- $\varphi = \langle \beta_1 \neq \beta_2 \rangle$: let $v \in V_K$ satisfy the hypothesis of statement 4, x_1 and x_2 be nodes such that $(v, x_i) \in [\![\beta_i]\!]_K$ for $i \in \{1, 2\}$ and $D_K(x_1) \neq D_K(x_2)$, and $z \in V_K$ other node such that $D_K(z) \notin \Sigma_n^{\varphi}$. If $v \neq x_i$ for $i \in \{1, 2\}$ then the result follows by statement 2. Otherwise, consider that $x_1 = v$, which then implies that $(z, z) \in [\![\beta_1]\!]_K$ by statement 3. If $x_2 \neq v$ then by statement 2 we can conclude that $(z, x_2) \in [\![\beta_2]\!]_K$. If $D_K(x_2) \neq D_K(z)$ we have finished, and if not, knowing that $D_K(x_2) \notin \Sigma_n^{\varphi}$ (this follows from $D_K(x_2) = D_K(z)$), we can consider that $(z, v) \in [\![\beta_2]\!]_K$ by statement 1 and that $D_K(x_2) \neq D_K(x_1) = D_K(v)$. The case $x_2 = v$ cannot happen, since we are assuming that $x_1 = v$ and $D_K(x_1) \neq D_K(x_2)$.

□

Now we continue with the Lemma 21.

Let G' be a superset repair of G with respect to R. If $\Sigma_n^{G'} \subseteq \Sigma_n^G \cup \Sigma_n^R \cup \{c, d\}$ for some $c, d \in \Sigma_n \setminus \Sigma_n^R$, $c \neq d$, then G' is already the witness we want for this lemma. Otherwise, take $c, d \in \Sigma_n^{G'} \setminus (\Sigma_n^G \cup \Sigma_n^R)$, $c \neq d$. We define the graph $H = (V_H, L_H, D_H)$ where:

$$V_H = V_{G'}$$

$$L_H(v, w) = \Sigma_e^{G'} \quad \forall v, w \in V_H$$

$$D_H(v) = \begin{cases} D_{G'}(v) & \text{if } D_{G'}(v) \in \Sigma_n^G \cup \Sigma_n^R \cup \{c, d\} \\ c & \text{otherwise} \end{cases}$$

where $\Sigma_e^{G'}$ is the set of all edge labels mentioned in G' (i.e. $\Sigma_e^{G'} = \{l \in \Sigma_e \mid \exists v, w \in V_{G'}$ such that $l \in L_{G'}(v, w)\}$).

Intuitively, we have changed all data values that were not in $\Sigma_n^G \cup \Sigma_n^R \cup \{c, d\}$ into c, and we also added every possible edge considering those labels already present in G'. Note that $\Sigma_n^H \subseteq \Sigma_n^G \cup \Sigma_n^R \cup \{c, d\}$, so showing that $H \models R$ is enough to prove the lemma. We will show by induction in the expressions' structure that if a pair $v, w \in V_{G'}$ satisfies $(v, w) \in [\![\alpha]\!]_{G'}$

then $(v, w) \in [\![\alpha]\!]_H$, and that if $v \in [\![\varphi]\!]_{G'}$ then $v \in [\![\varphi]\!]_H$ (where the expressions considered only use data values from Σ_n^R).

For the base cases:

- $\alpha = \_$: if some pair (v, w) is in $[\![\_]\!]_{G'}$ then $\Sigma_e^{G'} \neq \emptyset$, and $(v, w) \in [\![\_]\!]_{G'}$.
- $\alpha = \text{A}$ and $\alpha = \text{A}^-$: if $(v, w) \in [\![\text{A}]\!]_{G'}$ then clearly $\text{A} \in \Sigma_e^{G'}$ and $\text{A} \in L_H(v, w)$.
- $\alpha = \epsilon$: this case follows trivially.
- $\varphi = \text{E}^=$: if $v \in [\![\text{E}^=]\!]_{G'}$ then $D_{G'}(v) = \text{E}$. Since $\text{E} \in \Sigma_n^R$ the data value was preserved in H ($D_H(v) = \text{E}$) and thus $v \in [\![\text{E}^=]\!]_H$.
- $\varphi = \text{E}^{\neq}$: let $v \in [\![\text{E}^{\neq}]\!]_{G'}$. If $D_{G'}(v) \in \Sigma_n^G \cup \Sigma_n^R \cup \{c, d\}$ then the data value of v was preserved, and $v \in [\![\text{E}^{\neq}]\!]_H$. Otherwise $D_H(v) = \text{C}$, and since $\text{C} \notin \Sigma_n^R$ we know $v \in [\![\text{E}^{\neq}]\!]_H$.

Now, for the inductive cases:

- $\alpha = [\psi]$: let $(v, v) \in [\![[\psi]]\!]_{G'}$, which implies that $v \in [\![\psi]\!]_{G'}$. By inductive hypothesis we have that $v \in [\![\psi]\!]_H$, and then $(v, v) \in [\![[\psi]]\!]_H$.
- $\alpha = \beta_1.\beta_2$: if $(v, w) \in [\![\beta_1.\beta_2]\!]_{G'}$ then there exists $z \in V_{G'}$ such that $(v, z) \in [\![\beta_1]\!]_{G'}$ and $(z, w) \in [\![\beta_2]\!]_{G'}$. By inductive hypothesis $(v, z) \in [\![\beta_1]\!]_H$ and $(z, w) \in [\![\beta_2]\!]_H$, which implies that $(v, w) \in [\![\beta_1.\beta_2]\!]_H$.
- $\alpha = \beta_1 \cup \beta_2$: let $(v, w) \in [\![\beta_1 \cup \beta_2]\!]_{G'}$. Without loss of generality, we can assume that $(v, w) \in [\![\beta_1]\!]_{G'}$. By inductive hypothesis $(v, w) \in [\![\beta_1]\!]_H$, and then $(v, w) \in [\![\beta_1 \cup \beta_2]\!]_H$.
- $\alpha = \beta_1 \cap \beta_2$: since $(v, w) \in [\![\beta_1 \cap \beta_2]\!]_G \iff (v, w) \in [\![\beta_1]\!]_G \cap [\![\beta_2]\!]_G$ then the inductive hypothesis implies $(v, w) \in [\![\beta_1]\!]_H \cap [\![\beta_2]\!]_H = [\![\beta_1 \cap \beta_2]\!]_H$.
- $\alpha = \beta^*$: $(v, w) \in [\![\beta^*]\!]_{G'}$ if there exists $z_1, ... z_m \in V_{G'}$ such that $z_1 = v$, $z_m = w$ and $(z_i, z_{i+1}) \in [\![\beta]\!]_{G'}$ for $1 \leq i \leq m - 1$. By inductive hypothesis $(z_i, z_{i+1}) \in [\![\beta]\!]_H$ for $1 \leq i \leq m - 1$, and then $(v, w) \in [\![\beta^*]\!]_H$.
- The case $\alpha = \beta^{n,m}$ follows with the same argument used in the $.^*$ case.
- $\varphi = \psi_1 \wedge \psi_2$: let $v \in [\![\psi_1 \wedge \psi_2]\!]_{G'}$. Since $v \in [\![\psi_i]\!]_{G'}$ for $i \in \{1, 2\}$ we know by hypothesis that $v \in [\![\psi_i]\!]_H$ for $i \in \{1, 2\}$, which implies that $v \in [\![\psi_1 \wedge \psi_2]\!]_H$.
- $\varphi = \psi_1 \vee \psi_2$: we can assume without loss of generality that if $v \in [\![\psi_1 \vee \psi_2]\!]_G$ then $v \in [\![\psi_1]\!]_G$, and we can readily conclude by inductive hypothesis that $v \in [\![\psi_1]\!]_H \subseteq [\![\psi_1 \vee \psi_2]\!]_H$.
- $\varphi = \langle\beta\rangle$: if $v \in [\![\langle\beta\rangle]\!]_{G'}$ then there exists $z \in V_{G'}$ such that $(v, z) \in [\![\beta]\!]_{G'}$. Then by hypothesis $(v, z) \in [\![\beta]\!]_H$ and $v \in [\![\langle\beta\rangle]\!]_H$.

- $\varphi = \langle \beta_1 = \beta_2 \rangle$: if $v \in [\![\langle \beta_1 = \beta_2 \rangle]\!]_{G'}$ then there exists $z_1, z_2 \in V_{G'}$ such that $(v, z_1) \in [\![\beta_1]\!]_{G'}$, $(v, z_2) \in [\![\beta_2]\!]_{G'}$ and $D_{G'}(z_1) = D_{G'}(z_2)$. By construction $D_H(z_1) = D_H(z_2)$, and by hypothesis $(v, z_1) \in [\![\beta_1]\!]_H$ and $(v, z_2) \in [\![\beta_2]\!]_H$, which implies that $v \in [\![\langle \beta_1 = \beta_2 \rangle]\!]_H$.
- $\varphi = \langle \beta_1 \neq \beta_2 \rangle$: this is the most interesting case. Note that the paths used in G to satisfy the expression could now lead to nodes with the same data value. If $v \in [\![\langle \beta_1 \neq \beta_2 \rangle]\!]_{G'}$ then there exist nodes $z_1, z_2 \in V_{G'}$ such that $(v, z_1) \in [\![\beta_1]\!]_{G'}$, $(v, z_2) \in [\![\beta_2]\!]_{G'}$ and $D_{G'}(z_1) \neq D_{G'}(z_2)$. If $D_H(z_1) \neq D_H(z_2)$ then we can conclude using the inductive hypothesis that $v \in [\![\langle \beta_1 \neq \beta_2 \rangle]\!]_H$. If this is not the case, then both nodes had in G' as data value either c or one of those we changed (i.e. one from $\Sigma_n^{G'} \setminus (\Sigma_n^G \cup \Sigma_n^R \cup \{c, d\})$. Note that since $D_{G'}(z_1) \neq D_{G'}(z_2)$, it must hold that either $z_1 \neq v$ or $z_2 \neq v$; without loss of generality we can assume that $z_1 \neq v$. Let v_d be a node from H that has data value d (by construction there must be at least one such node). Since $(v, z_1) \in [\![\beta_1]\!]_H$, $v \neq z_1$, $D_H(z_1) \notin \Sigma_n^R$ and H is a 'complete graph' under edge labels form $\Sigma_e^{G'}$ we can deduce using statement 1 of Lemma 28 that $(v, v_d) \in [\![\beta_1]\!]_H$. Then we can conclude that $v \in [\![\langle \beta_1 \neq \beta_2 \rangle]\!]_H$ taking into account that $(v, z_2) \in [\![\beta_2]\!]_H$ due to the inductive hypothesis.

We can then conclude, since $G' \models R$, that $H \models R$ holds.

□

Proof of Lemma 23. Let $f : V_G \to V_H$ be a mapping between the nodes of G and H such that $f(v) = v$ if $v \in V_G \setminus V_d$, and $f(v) = v_d$ otherwise. Now we show, by induction over the formula's structure, that if a pair of nodes $v, w \in V_G$ satisfies $(v, w) \in [\![\alpha]\!]_G$, then $(f(v), f(w)) \in [\![\alpha]\!]_H$, and that if $v \in [\![\varphi]\!]_G$ then $f(v) \in [\![\varphi]\!]_H$. This is enough to prove the lemma.

For the base cases:

- $\alpha = \_$: if both v and w were not in V_d then their edges were preserved. If $w \in V_d$ and $v \notin V_d$ then by construction $L_G(v, w) \subseteq L_H(v, v_d)$. The case when $v \in V_d$ is analogous. If both v and w are in V_d we use the fact that $L_G(v, w) \subseteq L_H(v_d, v_d)$.
- $\alpha = \text{A}$ or $\alpha = \text{A}^-$: idem the previous case.
- $\alpha = \epsilon$: this one follows trivially.
- $\varphi = \text{C}^=$: v and $f(v)$ have the same data value, which means that this case holds.
- $\varphi = \text{C}^{\neq}$: idem the previous case.

Now, the inductive ones:

- $\alpha = [\psi]$: If $(v, v) \in [\![[\psi]]\!]_G$ then $v \in [\![\psi]\!]_G$, which implies by inductive hypothesis that $f(v) \in [\![\psi]\!]_H$ and then $(f(v), f(v)) \in [\![[\psi]]\!]_H$.
- $\alpha = \beta_1.\beta_2$: if $(v, w) \in [\![\beta_1.\beta_2]\!]_G$ then there exists $z \in V_G$ such that $(v, z) \in [\![\beta_1]\!]_G$ and $(z, w) \in [\![\beta_2]\!]_G$. Then by inductive hypothesis $(f(v), f(z)) \in [\![\beta_1]\!]_H$ and $(f(z), f(w)) \in [\![\beta_2]\!]_H$, which implies that $(f(v), f(w)) \in [\![\beta_1.\beta_2]\!]_H$.

- $\alpha = \beta_1 \cup \beta_2$: suppose without loss of generality that $(v, w) \in [\![\beta_1]\!]_G$. Then by induction $(f(v), f(w)) \in [\![\beta_1 \cup \beta_2]\!]_H$.
- $\alpha = \beta_1 \cap \beta_2$: if $(v, w) \in [\![\beta_1 \cap \beta_2]\!]_G$ then $(v, w) \in [\![\beta_1]\!]_G \cap [\![\beta_2]\!]_G$, and by the inductive hypothesis we can deduce that $(f(v), f(w)) \in [\![\beta_1]\!]_H \cap [\![\beta_2]\!]_H = [\![\beta_1 \cap \beta_2]\!]_H$.
- $\alpha = \beta^*$: $(v, w) \in [\![\beta^*]\!]_G$ if there exists $z_1, ..., z_m \in V_G$ such that $z_1 = v$, $z_m = w$ and $(z_i, z_{i+1}) \in [\![\beta]\!]_G$ for $1 \leq i \leq m - 1$. By hypothesis it follows that $(f(z_i), f(z_{i+1})) \in [\![\beta]\!]_H$, which implies that $(f(v), f(w)) \in [\![\beta^*]\!]_H$.
- $\alpha = \beta^{n,m}$: this case follows using the same argument used in the $.^*$ case.
- $\varphi = \psi_1 \wedge \psi_2$: $v \in [\![\psi_1 \wedge \psi_2]\!]_G$ if $v \in [\![\psi_1]\!]_G$ and $v \in [\![\psi_2]\!]_G$. By using the hypothesis we conclude that $f(v) \in [\![\psi_1]\!]_H$ and $f(v) \in [\![\psi_2]\!]_H$, which implies that $f(v) \in [\![\psi_1 \wedge \psi_2]\!]_H$.
- $\varphi = \psi_1 \vee \psi_2$: if $v \in [\![\psi_1 \vee \psi_2]\!]_G$ then without loss of generality we can assume $v \in [\![\psi_1]\!]_G$. Then by inductive hypothesis $f(v) \in [\![\psi_1]\!]_H \subseteq [\![\psi_1 \vee \psi_2]\!]_H$.
- $\varphi = \langle\beta\rangle$: $v \in [\![\langle\beta\rangle]\!]_G$ if $(v, z) \in [\![\beta]\!]_G$ for some z. By hypothesis $(f(v), f(z)) \in [\![\beta]\!]_H$, and then $f(v) \in [\![\langle\beta\rangle]\!]_H$.
- $\varphi = \langle\beta_1 \star \beta_2\rangle$, with $\star \in \{=, \neq\}$: $v \in [\![\langle\beta_1 \star \beta_2\rangle]\!]$ if there exists $z_1, z_2 \in V_G$ such that $(v, z_1) \in [\![\beta_1]\!]_G$, $(v, z_2) \in [\![\beta_2]\!]_G$ and $D_G(z_1) \star D_G(z_2)$. In H both z_1 and z_2 preserve their data values (i.e. $D_G(z_i) = D_H(f(z_i))$ for $i \in \{1, 2\}$), and by inductive hypothesis $(f(v), f(z_i)) \in [\![\beta_i]\!]_H$ for $i \in \{1, 2\}$. Then $f(v) \in [\![\langle\beta_1 \star \beta_2\rangle]\!]_H$.

□

References

Abiteboul, S., & Vianu, V. (1999). Regular path queries with constraints. *Journal of Computer and System Sciences*, *58*(3), 428–452.

Anand, M. K., Bowers, S., & Ludäscher, B. (2010). Techniques for efficiently querying scientific workflow provenance graphs. *In Proc. of International Conference on Extending Database Technology (EDBT 2010)*, 287–298.

Angles, R., Arenas, M., Barceló, P., Boncz, P., Fletcher, G., Gutierrez, C., Lindaaker, T., Paradies, M., Plantikow, S., Sequeda, J., et al. (2018). G-core: A core for future graph query languages. *Proceedings of the 2018 International Conference on Management of Data*, 1421–1432.

Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J., & Vrgoč, D. (2017). Foundations of modern query languages for graph databases. *ACM Computing Surveys (CSUR)*, *50*(5), 1–40.

Angles, R., & Gutierrez, C. (2008). Survey of graph database models. *ACM Computing Surveys (CSUR)*, *40*(1), 1–39.

Arenas, M., Bertossi, L., & Chomicki, J. (1999). Consistent query answers in inconsistent databases. *Proc. of ACM Symposium on Principles of Database Systems (PODS 1999)*, *99*, 68–79.

Arenas, M., & Libkin, L. (2008). XML data exchange: Consistency and query answering. *Journal of the ACM (JACM)*, *55*(2), 1–72.

Arenas, M., & Pérez, J. (2011). Querying semantic web data with SPARQL. *In Proc. of ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 305–316.

Barceló, P. (2013). Querying graph databases. *Proc. of ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, 175–188.

Barceló, P., & Fontaine, G. (2017). On the data complexity of consistent query answering over graph databases. *Journal of Computer and System Sciences*, *88*, 164–194.

Barceló, P., Pérez, J., & Reutter, J. L. (2012). Relative expressiveness of nested regular expressions. *Alberto Mendelzon International Workshop on Foundations of Data Management*, *12*, 180–195.

Benedikt, M., & Koch, C. (2009). XPath leashed. *ACM Computing Surveys (CSUR)*, *41*(1), 1–54.

Bertossi, L. (2011). Database repairing and consistent query answering. *Synthesis Lectures on Data Management*, *3*(5), 1–121.

Bertossi, L. (2019). Database repairs and consistent query answering: Origins and further developments. *Proc. of ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 48–58.

Bienvenu, M., Bourgaux, C., & Goasdoué, F. (2019). Computing and explaining query answers over inconsistent DL-Lite knowledge bases. *Journal of Artificial Intelligent Research*, *64*(1), 563–644.

Buneman, P., Fan, W., & Weinstein, S. (2000). Path constraints in semistructured databases. *Journal of Computer and System Sciences*, *61*(2), 146–193.

Calvanese, D., Ortiz, M., & Šimkus, M. (2016). Verification of evolving graph-structured data under expressive path constraints. *19th International Conference on Database Theory (ICDT 2016)*.

Fan, W. (2012). Graph pattern matching revised for social network analysis. *Proc. of International Conference on Database Theory (ICDT 2012)*, 8–21.

Fan, W. (2019). Dependencies for graphs: Challenges and opportunities. *Journal of Data and Information Quality (JDIQ)*, *11*(2), 1–12.

Flesca, S., Furfaro, F., & Parisi, F. (2007). Preferred database repairs under aggregate constraints. *Proc. of International Conference on Scalable Uncertainty Management (SUM 2007)*, 215–229.

Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P., & Taylor, A. (2018). Cypher: An evolving query language for property graphs. *Proceedings of the 2018 International Conference on Management of Data*, 1433–1445.

Gheerbrant, A., Libkin, L., & Tan, T. (2012). On the complexity of query answering over incomplete XML documents. *Proc. of International Conference on Database Theory (ICDT 2012)*, 169–181.

Gogacz, T., Ibáñez-Garcıéa, Y. A., & Murlak, F. (2018). Finite query answering in expressive description logics with transitive roles. *CoRR*, *abs/1808.03130*. http://arxiv.org/abs/1808.03130

Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., Melo, G. d., Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., et al. (2021). Knowledge graphs. *Synthesis Lectures on Data, Semantics, and Knowledge*, *12*(2), 1–257.

Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., Van Kleef, P., Auer, S., et al. (2015). Dbpedia–a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic web*, *6*(2), 167–195.

Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., & Savo, D. F. (2010). Inconsistency-tolerant semantics for description logics. *Proc. of International Conference on Web Reasoning and Rule Systems (RR 2010)*, 103–117.

Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., & Savo, D. F. (2015). Inconsistency-tolerant query answering in ontology-based data access. *Journal of Web Semantics*, *33*, 3–29.

Libkin, L., Martens, W., & Vrgoč, D. (2016). Querying graphs with data. *Journal of the ACM (JACM)*, *63*(2), 1–53.

Libkin, L., & Vrgoč, D. (2012). Regular path queries on graphs with data. *Proc. of International Conference on Database Theory (ICDT 2012)*, 74–85.

Lopatenko, A., & Bertossi, L. (2007). Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. *Proc. of International Conference of Database Theory (ICDT 2007)*, 179–193.

Lukasiewicz, T., Malizia, E., Martinez, M. V., Molinaro, C., Pieris, A., & Simari, G. I. (2022). Inconsistency-tolerant query answering for existential rules. *Artificial Intelligence*, *307*, 103685.

Lukasiewicz, T., Martinez, M. V., Pieris, A., & Simari, G. I. (2015). From classical to consistent query answering under existential rules. *Proc. of AAAI Conference on Artificial Intelligence (AAAI 2015)*.

Lukasiewicz, T., Martinez, M. V., & Simari, G. I. (2013). Complexity of inconsistency-tolerant query answering in Datalog+/−. *Proc. of OTM Confederated International Conferences On the Move to Meaningful Internet Systems*, 488–500.

Pérez, J., Arenas, M., & Gutierrez, C. (2010). nSPARQL: A navigational language for RDF. *Journal of Web Semantics*, *8*(4), 255–270.

Rebele, T., Suchanek, F., Hoffart, J., Biega, J., Kuzey, E., & Weikum, G. (2016). YAGO: A multilingual knowledge base from Wikipedia, WordNet, and GeoNames. *Proc. of International semantic web conference (ISWC 2016)*, 177–185.

Rudolph, S. (2016). Undecidability results for database-inspired reasoning problems in very expressive description logics. *Proc. of International Conference on the Principles of Knowledge Representation and Reasoning (KR 2016)*.

Staworko, S., Chomicki, J., & Marcinkowski, J. (2012). Prioritized repairing and consistent query answering in relational databases. *Annals of Mathematics and Artificial Intelligence*, *64*(2), 209–246.

Ten Cate, B., Fontaine, G., & Kolaitis, P. G. (2012). On the data complexity of consistent query answering. *Proc. of International Conference on Database Theory (ICDT 2012)*, 22–33.

Ten Cate, B., Fontaine, G., & Kolaitis, P. G. (2015). On the data complexity of consistent query answering. *Theory of Computing Systems*, *57*(4), 843–891.

van Rest, O., Hong, S., Kim, J., Meng, X., & Chafi, H. (2016). Pgql: A property graph query language. *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, 1–6.

Vardi, M. Y. (1982). The complexity of relational query languages. *Proc. of ACM symposium on Theory of computing*, 137–146.

Wijsen, J. (2002). Condensed representation of database repairs for consistent query answering. *Proceedings of the 9th International Conference on Database Theory*, 378–393.