Schema Mappings and Data Exchange for Graph Databases

Pablo Barceló
Universidad de Chile
pbarcelo@dcc.uchile.cl

Jorge Pérez
Universidad de Chile
jperez@dcc.uchile.cl

Juan Reutter
University of Edinburgh
juan.reutter@ed.ad.uk

ABSTRACT

Data exchange, data integration and, more generally, schema mapping management, have received little attention so far in the graph database context, and tools from relational or XML databases suffer from important drawbacks when applied on graph-structured data. In this paper we embark on the study of interoperability among graph-structured data sources. We study schema mappings, data exchange, query answering and the fundamental task of composing schema mappings, in the context of graph databases.

We propose a novel formalism for defining schema mappings over graph databases, called *regular graph exchange dependencies* (RGEDs). Our proposal is specifically tailored for graph structured data, since it specifies schema relationships by using navigational functionalities. We then show how to perform data exchange tasks when mappings are defined by RGEDs. As one of our main results, we prove that there is an important fragment of RGEDs that is very well behaved with respect to data exchange, and, in particular, allows for efficient mapping-based query answering. We also show that this class of mappings is closed under composition, and allows itself for mapping-based query rewriting.

1. INTRODUCTION

Graph-structured data has become pervasive in data-centric applications. Social networks, bioinformatics, astronomic databases, digital libraries, Semantic Web, and linked government data, are only a few examples of applications in which structuring data as graphs is, simply, essential. Moreover, with the advent of huge initiatives like the *Web of Open Linked Data* (small pieces of data publicly available on the Web and described by using semantic links to other pieces of data [11]), we can only expect having an increasing interest from practitioners, developers and theoreticians on using, managing, and studying graph databases.

The fundamental problem of querying graph databases has received considerable attention in the database community [34, 2, 23, 37, 9, 24]. Queries in this context are normally designed to recognize patterns of nodes that are connected via paths satisfying certain *navigational* constraints expressed as regular expressions. But despite the interest in query-answering techniques, more complex data management tasks, and, in particular, interoperability issues among graph-structured data sources such as schema mapping management and data exchange, remain almost unexplored from a foundational point of view (and specially when compared with its relational and XML counterparts [5]). Some exceptions can be found in the work by Calvanese et al. regarding rewriting of views for graph-databases [13, 14], and more recently the study of schema-mapping simplification in the context of graph-structured data [15].

The absence of foundational work on the subject is in sharp contrast with practice. Graph-structured data is being used in a variety of applications for which interoperability issues are essential. For instance, social networks permanently need to be restructured, while its data needs to be migrated in order to construct new applications [32]; and Semantic Web applications often need to collaborate exchanging RDF-graph data [4].

The backbone of data interoperability tasks in the relational context is the notion of *schema mappings*, which allow to specify high-level relationships between two independent schemas, usually known as the *source* and the *target* schemas [21, 5]. Regrettably, the possibility of using relational mappings tools for solving data-interoperability tasks is not suitable for graph databases. We delve into this issue below.

Consider the graph database G_1 shown in Figure 1. It shows a fragment of the *RDF Linked Data* representation of DBLP [19].[1] Assume that Σ_1 is the alphabet used in the labeling of such graph. Suppose now that one wants to extract information from G_1, and transform/exchange it into a graph database G_2 that is labeled by an alphabet $\Sigma_2 = \{\texttt{makes}, \texttt{inConf}\}$ according to the following intuitive rule: If nodes u_1, u_2 and u_3 in G_1 satisfy that u_1 is an author (creator) of paper u_2, and u_2 has been published in conference u_3. Then node ids u_1, u_2 and u_3 belong to the target database G_2, u_1 is connected in G_2 by an `makes`-labeled edge to node id u_2, and u_2 is connected to u_3 in G_2 by an `inConf`-labeled edge.

One can easily define this rule by means of a *source-to-target dependency* (std), which constitute the basic building block of the mappings used in relational data exchange [21, 5]. Indeed, the following std (with its intuitive meaning)

[1]For brevity we have not depicted all the RDF-URI prefixes in the figure. You can start browsing the graph data at, for example, `http://dblp.l3s.de/d2r/resource/publications/conf/pods/Vardi95`.

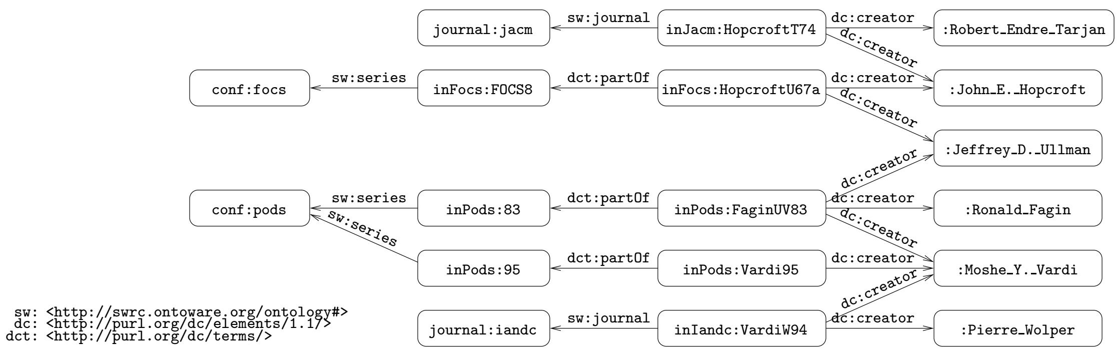

Figure 1: A fragment of the RDF Linked Data representation of DBLP available at `http://dblp.l3s.de/d2r/`.

specifies the rule used in our example (for simplicity, we omit prefixes `dc:`, `dct:`, and `sw:` in edge labels):

$$(y, \texttt{creator}, x) \wedge (y, \texttt{partOf}, z) \wedge (z, \texttt{series}, w) \longrightarrow \\ (x, \texttt{makes}, y) \wedge (y, \texttt{inConf}, w) \qquad (\dagger)$$

In general, stds are rules of the form $Q_{\mathbf{S}}(\bar{x}) \rightarrow Q_{\mathbf{T}}(\bar{x})$, where $Q_{\mathbf{S}}$ and $Q_{\mathbf{T}}$ are conjunctive queries over the source and the target schema, respectively. Such std specifies the following: Whenever the source database satisfies $Q_{\mathbf{S}}(\bar{a})$, for some tuple $\bar{a}$ of elements, then the target database must satisfy $Q_{\mathbf{T}}(\bar{a})$.

Unfortunately, specifying mappings with stds suffers from important drawbacks in the context of graph databases. First of all, additional features need to be included in these class of mappings in order to specify more complex navigational properties. For instance, if we want to migrate data based on a *coauthorship sequence* pattern as defined by expression $(\texttt{creator}^- \cdot \texttt{creator})^+$, then we would have to extend stds by some degree of recursion at least in the source side. However, this implies leaving relational data exchange tools behind, since relational engines do not cope well with recursive queries. Indeed, most of the standard tools developed for relational databases are not designed to compute reachability queries, and even though these navigational properties can sometimes be expressed in SQL or in other extensions of relational calculus, such as DATALOG, these queries can rarely be optimized. Furthermore, the dynamic essence of graph-databases reduces the likeliness of establishing indexes or other auxiliary data structures to help performing these tasks.

But, more importantly, it is well known [16] that the query evaluation problem is intractable in *combined* complexity (i.e. the complexity measured in terms of both the query and the data) for any mapping language based on conjunctive queries, such as the class of stds. Notice that evaluating a conjunctive query Q on a database D is of the order $|D|^{O(|Q|)}$ [1]. Although this may be reasonable for relational and XML databases that are often of moderate size, it is no longer reasonable for graph databases, which tend to be massive (think, for instance, of social networks or scientific databases that can store giga- or even terabytes of data). Indeed, having $O(|Q|)$ as the exponent may be prohibitively high for very large datasets. This rules out as impractical any mapping language for graph-structured data that is based on conjunctive queries.

The latter discussion raises the need for the design of schema mappings specially tailored for graph databases. We already mentioned that schema mapping languages for graph databases must allow highly efficient procedures when used for exchanging data. On the other hand, as previous experiences on this subject have taught us, it would also be desirable that such language is expressive enough to specify interesting exchange properties, as well as to lend itself to feasible *mapping management*, in particular, allowing high-level schema-mapping operations such as composition [8].

In order to find a good balance between expressive power and efficiency, our mapping design will be based on navigational languages that were originally proposed to meet the specific needs of graph-database applications. In order to do so, we need to deviate from the *source-to-target* paradigm adopted in relational and XML schema mappings, and focus instead on an approach that allows to exchange data *while* navigating the source graph database.

We explain the basic underlying idea behind our mappings by continuing with the previous example. We notice that by posing the expression

$$r \ := \ \texttt{creator}^- \cdot \texttt{partOf} \cdot \texttt{series},$$

we specify in the graph database G_1 each path that starts in an author u_1, then passes through a conference paper u_2 which he/she has (co-)authored, and finishes in the conference u_3 in which such paper was published. Our mappings should be able to specify in this case, for each path ρ defined by r in G_1, which pieces of the data (the nodes) in ρ are to be transferred into the target schema and how those pieces of data are to be structured with respect to such schema. In our case, the std in (†) tells us that each path ρ specified by r in G_1 of the form

$$u_1 \cdot \texttt{creator}^- \cdot u_2 \cdot \texttt{partOf} \cdot u_4 \cdot \texttt{series} \cdot u_3$$

should be transformed into a path

$$u_1 \cdot \texttt{makes} \cdot u_2 \cdot \texttt{inConf} \cdot u_3$$

in the target graph database. That is, only nodes u_1, u_2 and u_3, but not u_4, have to be transferred, with u_1 and u_2 being connected by an edge labeled `makes`, and u_2 and u_3 being connected by an edge labeled `inConf`.

The previous idea can be captured by the following simple rule from Σ_1 to Σ_2:

$$\mathcal{T}_1 \ = \ (\texttt{creator}^-/\texttt{makes})(\texttt{partOf} \cdot \texttt{series}/\texttt{inConf})$$

The "/" symbol makes the distinction between source and target schema, the source side selects a path in the source graph database satisfying some navigational properties (in

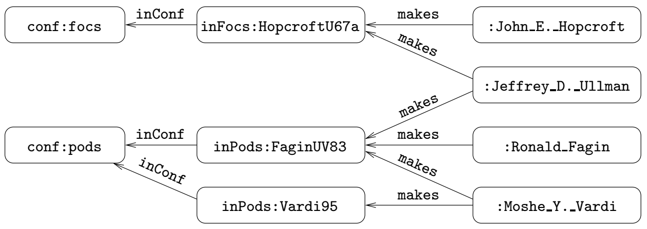

Figure 2: Result of exchanging the graph database in Figure 1 with rule $(\texttt{creator}^-/\texttt{makes})(\texttt{partOf}\cdot\texttt{series}/\texttt{inConf})$.

this case, according to the regular expression with inverse $\texttt{creator}^-\cdot\texttt{partOf}\cdot\texttt{series}$), while the target side reflects how such nodes are to be connected when constructing the exchanged graph database. Notice that this rule is capable of transforming and exchanging data from source to target *while* navigating the source graph database. Figure 2 shows a possible graph database that results from exchanging the information in G_1 according to the rule $\mathcal{T}_1$. Notice that this graph database also results from exchanging the data in G_1 according to the std in (†).

Our first and main task in the paper is to define a meaningful language for specifying schema mappings over graph-structured data. The main building block of our mappings is a class of dependencies, called *regular graph-exchange dependencies* or RGEDs. These rules, which form a suitable generalization of the rule $\mathcal{T}_1$ presented above, will be constructed along the guidelines sketched in the previous paragraphs. In particular, RGEDs will allow us to transfer data into the target schema while navigating the source graph database.

Having settled on a language, we turn to study basic properties of mappings specified by RGEDs (or RGED-mappings). We focus on data exchange, which is the problem of, given a mapping $\mathcal{M}$ from a source to a target schema and a source database G, materializing a target database G' that satisfies the mapping with respect to G. Such G' is usually called a *solution* for G under $\mathcal{M}$. In data exchange the space of solutions for a source database is often infinite, and hence one concentrates in materializing a solution that is a *representative* of the whole space of them [5].

The usual approach for solving this problem under relational or XML mappings, is to *extract* at once all the information that is to be transferred on the source side, and *load* it into the target database. In our case, RGEDs essentially specify how paths from the source side are to be transformed or exchanged into paths on the target side. However, since graph databases normally contain cycles, the number of different paths satisfying a specification could be infinite. Thus, we cannot directly use relational or XML techniques to solve the data exchange problem over graph-structured data, since this time the extraction process may yield an infinite set. This raises the need for techniques that are fundamentally different from the ones previously studied in relational or XML data exchange. To develop them we will need to draw concepts from automata theory, and apply them into the context of graph structured data.

We start by identifying a relevant class of RGED-mappings, called *full* RGED-mappings, with particularly good properties for interoperability tasks such as data exchange. We notice that full REGD-mappings can express interesting specifications, such as the one defined by rule $\mathcal{T}_1$ presented above. We show that under full RGED-mappings, for each source graph database G we can construct in polynomial time (in combined complexity) a *minimal* representative G', in the sense that G' is contained in every other solution for G. With the help of this result, we show that the task of answering expressive navigational languages in the context of graph data exchange under full RGED-mappings, can also be solved in polynomial time in combined complexity. Further, by using *query rewriting* techniques, we develop an algorithm for answering queries in linear time in *data* complexity (i.e. the complexity measured only in terms of the size of the data). This rewriting algorithm promises to be useful as well in the context of data integration, since it allows to answer queries without requiring the materialization of a target graph database. Finally, we show that full RGED-mappings also present themselves as a good language with respect to metadata management, since they are *closed under composition*, in the sense that the composition of two full RGED-mappings can always be specified with a full RGED-mapping.

We continue then with the study of data exchange problems for arbitrary RGED-mappings. Although source graph databases do not admit minimal representatives in general, we show that a *universal* representative (i.e. one that represents the whole space of solutions) can be constructed in polynomial time, if we allow graph databases to be extended with *incomplete* information [10]. This is a standard process also applied in relational and XML data exchange [5]. We also study query answering over data exchange scenarios given by arbitrary RGED-mappings. Although we show that the query answering problem in this case becomes intractable, we identify an intermediate class of RGED-mappings, called *semifull* RGED-mappings, that provide a good combination of efficiency and expressiveness: Despite being more expressive than full RGED-mappings, they still allow for tractable query answering.

The rest of the paper is organized as follows. In Section 2 we formally define graph databases, and the language of nested regular expressions. Section 3 introduces RGEDs and RGED-mappings, and study their basic properties. Section 4 shows how to perform the main data exchange tasks for full RGED-mappings, and Section 5 extends the study to arbitrary RGED-mappings. Due to space limitations, most of the proofs and some algorithms are included in an extended version, that can be downloaded from `www.homepages.inf.ed.ac.uk/s0932806/vldb12-ext.pdf`.

2. GRAPH DATABASES AND NESTED REGULAR EXPRESSIONS

Graph databases As customary in the literature, we use *edge-labeled* directed graphs to represent graph databases. Formally, let $\mathbf{V}$ be a countably infinite set of *node ids*, and Σ a finite alphabet. A *graph database* G over Σ is a pair (V, E), where V is a finite set of node ids (that is V is a finite subset of $\mathbf{V}$) and $E \subseteq V \times \Sigma \times V$. The fact that (u, a, v) belongs to E represents that there is an edge from node u into node v labeled a. For a graph database $G = (V, E)$, we write $(u, a, v) \in G$ whenever $(u, a, v) \in E$. For instance, Figure 1 depicts a graph database over alphabet $\Sigma_1 = \{\texttt{creator}, \texttt{partOf}, \texttt{series}\}$.

Notice that each graph database $G = (V, E)$ can be naturally seen as a nondeterministic finite automata (NFA), where the nodes of G correspond to the states of the NFA and there is a transition labeled a from state v into state v' if and only if $(v, a, v') \in E$. This last property of graph databases will be heavily exploited throughout the paper.

Nested regular expressions Mappings specify how to extract data from a source schema and exchange it into a target schema. This implies that at the core of mappings lies a language that allows to specify the extraction and exchange processes. Hence, our study on mappings over graph databases cannot start without a discussion on the underlying languages that will be used by those mappings.

Query languages over graph databases are typically navigational, in the sense that they allow to traverse the edges of the graph while checking for the existence of paths satisfying certain conditions. There exist many query languages for graph databases with a navigational flavour. However, since current applications of graph databases tend to store massive volumes of data, if we want our study to be practically meaningful we should concentrate on languages that satisfy an extra constraint in terms of query evaluation. In fact, we should require the *combined* complexity of the query evaluation problem associated with the language (i.e. the complexity measured in terms of both the query and the data) to be polynomial in the size of the database. Indeed, any language that does not satisfy this can be immediately rendered infeasible for current graph database applications.

Several navigational languages for graph databases satisfy this constraint. In this work we have chosen to work with *nested regular expressions* (NREs), which were proposed in [36] for querying Semantic Web data. The reason for our choice is that, as we will see later, NREs are expressive enough for subsuming several laguages of interest, while at the same time its combined complexity is not only polynomial but linear in the size of the data (see Proposition 2.2). Next we formalize the language of nested regular expressions in the context of graph databases.

Let Σ be a finite alphabet. The NREs over Σ extend classical regular expressions with an *existential nesting test* operator $[\,\cdot\,]$ (or just nesting operator, for short), and an *inverse* operator a^-, over each $a \in \Sigma$. The syntax of NREs is given by the following grammar:

$$\begin{aligned} \mathit{nexp} \; := \; & \varepsilon \mid a \; (a \in \Sigma) \mid a^- \; (a \in \Sigma) \mid \mathit{nexp} \cdot \mathit{nexp} \\ & \mathit{nexp}^* \mid \mathit{nexp} + \mathit{nexp} \mid [\mathit{nexp}] \end{aligned}$$

As it is customary, we use n^+ as shortcut for $n \cdot n^*$.

Intuitively, NREs specify pairs of node ids in a graph database, subject to the existence of a path satisfying a certain regular condition among them. That is, each NRE *nexp* defines a binary relation $[\![\mathit{nexp}]\!]_G$ when evaluated over a graph database G. This binary relation is defined inductively as follows, where we assume that a is a symbol in Σ, and n, n_1 and n_2 are arbitrary NREs:

$$\begin{aligned} [\![\varepsilon]\!]_G &= \{(u, u) \mid u \text{ is a node id in } G\} \\ [\![a]\!]_G &= \{(u, v) \mid (u, a, v) \in G\} \\ [\![a^-]\!]_G &= \{(u, v) \mid (v, a, u) \in G\} \\ [\![n_1 \cdot n_2]\!]_G &= [\![n_1]\!]_G \circ [\![n_2]\!]_G \\ [\![n_1 + n_2]\!]_G &= [\![n_1]\!]_G \cup [\![n_2]\!]_G \\ [\![n^*]\!]_G &= [\![\varepsilon]\!]_G \cup [\![n]\!]_G \cup [\![n \cdot n]\!]_G \cup [\![n \cdot n \cdot n]\!]_G \cup \cdots \\ [\![\,[n]\,]\!]_G &= \{(u, u) \mid \text{there exists } v \text{ s.t. } (u, v) \in [\![n]\!]_G\}. \end{aligned}$$

Here, the symbol $\circ$ denotes the usual composition of binary relations, that is, $[\![n_1]\!]_G \circ [\![n_2]\!]_G = \{(u, v) \mid$ there exists w s.t. $(u, w) \in [\![n_1]\!]_G$ and $(w, v) \in [\![n_2]\!]_G\}$.

Example 2.1. Let G_1 be the graph database in Figure 1. The following is a simple NRE that matches all pairs (x, y) such that x is an author that published a paper in conference y (for simplicity, we omit prefixes `dc:`, `dct:`, and `sw:` in edge labels):

$$n_1 \; = \; \texttt{creator}^- \cdot \texttt{partOf} \cdot \texttt{series}.$$

For example the pairs (`:Jeffrey_D._Ullman`, `conf:focs`) and (`:Ronald_Fagin`, `conf:pods`) are in $[\![n_1]\!]_G$. Consider now the following expression that matches pairs (x, y) such that x and y are connected by a *coautorship sequence*:

$$n_2 \; = \; (\texttt{creator}^- \cdot \texttt{creator})^+.$$

For example the pair (`:John_E._Hopcroft`, `:Pierre_Wolper`), is in $[\![n_2]\!]_G$. Finally the following expression matches all pairs (x, y) such that x and y are connected by a coautorship sequence that only considers conference papers:

$$n_3 \; = \; (\texttt{creator}^- \cdot [\,\texttt{partOf} \cdot \texttt{series}] \cdot \texttt{creator})^+.$$

Let us give the intuition of the evaluation of this expression. Assume that we start at node u. The (inverse) edge $\texttt{creator}^-$ makes us to navigate from u to a paper v created by u. Then the existential test $[\,\texttt{partOf} \cdot \texttt{series}]$ is used to check that from v we can navigate to a conference (and thus, v is a conference paper). Finally, we follow edge `creator` from v to an author w of v. The $(\cdot)^+$ over the expression allows us to repeat this sequence several times. For instance, (`:John_E._Hopkroft`, `:Moshe_Y._Vardi`) is in $[\![n_3]\!]_G$, but (`:John_E._Hopkroft`, `:Pierre_Wolper`) is not in $[\![n_3]\!]_G$. ☐

Complexity and expressiveness of NREs The following result, proved in [36], shows a remarkable property of NREs. It states that the query evaluation problem for NREs is not only polynomial in *combined* complexity (i.e. when both the database and the query are given as input), but also that it can be solved linearly in both the size of the database and the expression. Given a graph database G and an NRE *nexp*, we use $|G|$ to denote the size of G (in terms of the number of egdes $(u, a, v) \in G$), and $|\mathit{nexp}|$ to denote the size of *nexp*.

Proposition 2.2 (from [36]). *Checking, given a graph database G, a pair of nodes (u, v), and an NRE nexp, whether $(u, v) \in [\![\mathit{nexp}]\!]_G$, can be done in time $O(|G| \cdot |\mathit{nexp}|)$.*

On the expressiveness side, NREs subsume several important query languages for graph databases. For instance, by disallowing the inverse operator a^- and the nesting operator $[\,\cdot\,]$ we obtain the class of *regular path queries* (RPQs) [18, 34], while by only disallowing the nesting operator $[\,\cdot\,]$ we obtain the class of RPQs with *inverse* or 2RPQs [14]. (In particular, both expressions n_1 and n_2 in Example 2.1 are 2RPQs). In turn, NREs allow for an important increase in expressive power over those languages. For example, it can be shown that NRE expression n_3 in Example 2.1 cannot be expressed without the nesting operator $[\,\cdot\,]$, and hence it is not expressible in the language of 2RPQs (c.f. [36]).

On the other hand, the class of NREs fails capturing more expressive languages for graph-structured data that combine navigational properties with quantification over node ids. Some of the most paradigmatic examples of such languages are the classes of *conjunctive* RPQs and 2RPQs, that close RPQs and 2RPQs, respectively, under conjunctions and existential quantification. Both classes of queries have been studied in depth, as they allow identifying complex patterns over graph-structured data [17, 25, 12].

It is not hard to prove that C2RPQs subsume, but are strictly more expressive than NREs. This naturally raises the question of why not to choose C2RPQs as our core query language for graph databases. The reason is simple: C2RPQs contain the class of conjunctive queries. This implies, as we mentioned in the introduction, that its evaluation is impractical over current graph database applications that are massive in size.

In summary, NREs present a good balance between expressiveness and complexity of query evaluation when compared to 2RPQs and C2RPQs. In fact, NREs are strictly more expressive than 2RPQs while retaining its good properties for query evaluation. On the other hand, they are also a suitable restriction of the class of C2RPQs with tractable (and even linear in the size of the data and the expression) query evaluation complexity.

3. SCHEMA MAPPINGS FOR GRAPH DATA

3.1 Graph-exchange dependencies: Syntax and semantics

In this section we define a class of rules, called regular graph-exchange dependencies or RGEDs, that will serve as the basic building block of our mapping language for graph databases. In doing so, we will have to follow the guidelines sketched along the introduction of the paper, in particular, in terms of designing rules that allow to transfer data while navigating the source graph database. More specifically, our RGEDs will generalize rules such as

$$\mathcal{T}_1 \;=\; (\texttt{creator}^-/\texttt{makes})(\texttt{partOf}\cdot\texttt{series}/\texttt{inConf})$$

presented earlier in Section 1.

We now formally define the language of RGEDs. As we mentioned in the previous motivation, we want this formalism to be able to transfer data while navigating the source graph database. In the case of RGEDs, such navigation will be enabled by the use of regular expressions. We could certainly extend the language of RGEDs by allowing more expressive yet efficient navigational languages as a building block, e.g. 2RPQs or NREs. Nevertheless, at this stage this would only complicate the definition of the semantics of RGEDs while not providing an important increase in expressive power.

On the other hand, RGEDs are more than simple regular expresssions. In fact, RGEDs are regular expressions over the (infinite) alphabet that consists of all pairs of the form (n_1, n_2), where n_1 and n_2 are NREs over the source and target schema, respectively. This allows to define how source data is to be transfered into the target schema. The reason why we have chosen NREs as the atomic symbols used in our RGEDs is that, as we mentioned in Section 2, they possess a good balance between expressive power and complexity of query evaluation. Hence they allow us to express interesting patterns for extracting and exchanging data, but at a reasonable cost for practical applications.

Given a finite alphabet Σ, we denote by $\mathrm{NREG}(\Sigma)$ the set of nested regular expressions over Σ. We then define the class of RGEDs as follows:

Definition 3.1 (RGEDs). *Let Σ and Σ' be finite alphabets and let $\mathcal{A}(\Sigma, \Sigma')$ be the set*

$$\{(nexp/nexp') \mid nexp \in \mathrm{NREG}(\Sigma),\ nexp' \in \mathrm{NREG}(\Sigma')\}.$$

A regular graph-exchange dependency*, or just* RGED*, from Σ to Σ', is a regular language over the (infinite) alphabet $\mathcal{A}(\Sigma, \Sigma')$.*

Since there exists a simple polynomial-time translation from regular expressions into nondeterminstic finite automata (NFAs), we often take advantage of the automata representation of RGEDs when proving algorithmic properties of them. In such case, we assume that an RGED is given as an NFA over the alphabet $\mathcal{A}(\Sigma, \Sigma')$.

Every RGED that is written as a regular expression over the alphabet $\{(nexp/a) \mid nexp \in \mathrm{NREG}(\Sigma),\ a \in \Sigma')\}$ is called *full*. The terminology comes from the fact that they resemble *full* stds, as usually found in relational data exchange [21]. We will see in Section 4 that mappings specified by full RGEDs have particularly good properties for graph data exchange.

Example 3.2. The rule

$$\mathcal{T}_1 \;=\; (\texttt{creator}^-/\texttt{makes})(\texttt{partOf}\cdot\texttt{series}/\texttt{inConf})$$

is a full RGED from Σ_1 to Σ_2. Consider now the alphabet $\Sigma_3 = \{\texttt{confConnected}\}$. Then the expression

$$\mathcal{T}_2 \;=\; \big((\texttt{makes}\cdot\texttt{makes}^-)^+/\texttt{coauthor}\big)$$

is also a full RGED, this time from Σ_2 to Σ_3.

The full RGED $\mathcal{T}_2$ can be modified into the (non-full) RGED

$$\mathcal{T}_3 \;=\; \big((\texttt{makes}\cdot\texttt{makes}^-)^+/\texttt{coauthor}^+\big).$$

Intuitively, $\mathcal{T}_3$ can be considered as a relaxation of $\mathcal{T}_2$; it only states that for each two nodes (u, v) of the source graph connected by a path of the form $(\texttt{makes}\cdot\texttt{makes}^-)^+$, the target graph must contain a path from u to v with all intermediate edges labeled by $\texttt{coauthor}$.

The presence of NREs in RGEDs allows us to express a rule from Σ_1 to Σ_2 of the form:

$$\big((\texttt{creator}^-\cdot[\texttt{partOf}\cdot\texttt{series}]\cdot\texttt{creator})/\texttt{confCoauthor}\big)$$

Intuitively, this rule states that whenever a pair of node ids in the source graph database belongs to the evaluation of $(\texttt{creator}^-\cdot[\texttt{partOf}\cdot\texttt{series}]\cdot\texttt{creator})$, then the same pair of node ids must be connected by an edge labeled $\texttt{confCoauthor}$ in the target database (i.e. they are coauthors in a conference paper). $\square$

We now define the semantics of RGEDs along the lines developed in Section 1. The semantics is based on the notions of source and target *matchings* that we define below. Intuitively, a pair (G, G') of graph databases over source and target schema, respectively, belongs to the semantics of an RGED $\mathcal{T}$, if every source matching of $\mathcal{T}$ over G is also a target matching of $\mathcal{T}$ over G'.

Let Σ and Σ' be finite alphabets, G a graph database over Σ and $\mathcal{T}$ an RGED from Σ to Σ'. A *source matching* of $\mathcal{T}$ over G is a sequence

$$v_0(e_1/f_1)v_1(e_2/f_2)v_2 \cdots v_{n-1}(e_n/f_n)v_n,$$

with $n \geq 1$, satisfying the following:

- $(e_1/f_1)(e_2/f_2)\cdots(e_n/f_n)$ is a string over $\mathcal{A}(\Sigma,\Sigma')$ that belongs to the regular language defined by $\mathcal{T}$, and
- for each $1 \leq i \leq n$ it holds that $(v_{i-1}, v_i) \in \llbracket e_i \rrbracket_G$.

For instance, ρ defined as

$$\texttt{Ronald\_Fagin} \cdot (\texttt{creator}^-/\texttt{makes}) \cdot \texttt{inPods:FaginUV83} \cdot (\texttt{partOf} \cdot \texttt{series}/\texttt{inConf}) \cdot \texttt{conf:PODS}$$

is a source matching of $\mathcal{T}_1$ over graph database G_1 shown in Figure 1.

Similarly as we define a source matching, we can define a target matching of $\mathcal{T}$. Given a graph database G' over Σ', a *target matching* of $\mathcal{T}$ over G' is a sequence

$$u_0(e_1/f_1)u_1(e_2/f_2)u_2 \cdots u_{n-1}(e_n/f_n)u_n,$$

with $n \geq 1$, such that:

- $(e_1/f_1)(e_2/f_2)\cdots(e_n/f_n)$ is a string over $\mathcal{A}(\Sigma,\Sigma')$ that belongs to the regular language defined by $\mathcal{T}$, and
- for each $1 \leq i \leq n$ it holds that $(u_{i-1}, u_i) \in \llbracket f_i \rrbracket_{G'}$.

For instance, sequence ρ defined above is also a target matching of $\mathcal{T}_1$ over graph database G_2 depicted in Figure 2.

Intuitively, the source (target) matchings of $\mathcal{T}$ are the paths defined by the regular expression $\mathcal{T}$ while navigating the source (target) data. The source matchings define which data from the source will be extracted in order to perform the exchange, while the target matchings specify how the extracted source data is exchanged and restructured according to the target schema.

Since RGEDs force data to be exchanged while navigating the source database, its semantics is based on a tight coupling of source and target matchings. With this in mind we define when a pair (G, G') satisfies an RGED.

Definition 3.3 (Semantics of RGEDs). *Let G and G' be graph databases over Σ and Σ', respectively, and let $\mathcal{T}$ be an RGED from Σ to Σ'. Then (G, G') satisfies $\mathcal{T}$, denoted by $(G, G') \models \mathcal{T}$, if every source matching of $\mathcal{T}$ over G is also a target matching of $\mathcal{T}$ over G'.*

Example 3.4. We can now formally prove that $(G_1, G_2) \models \mathcal{T}_1$, where G_1 and G_2 are the graph databases in Figures 1 and 2, respectively, and $\mathcal{T}_1$ is the RGED $(\texttt{creator}^-/\texttt{makes})$ $(\texttt{partOf} \cdot \texttt{series}/\texttt{inConf})$. The proof easily follows from the remarks in Section 1.

It is also not hard to see that the semantics of $\mathcal{T}_1$ coincides with the semantics of the std (†) shown in Section 1 over G_1, and, indeed, over any graph database defined with respect to Σ_1. $\square$

3.2 RGED-based schema mappings

Let Σ and Σ' be two finite alphabets. A *schema mapping* (or just mapping for short) from Σ to Σ', is a set of pairs of the form (G, G'), where G is a graph database over Σ and G' is a graph database over Σ'. Following the usual data exchange terminology, for a mapping $\mathcal{M}$ and pair of graphs G and G' such that $(G, G') \in \mathcal{M}$, we say that G' is a *solution* for G under $\mathcal{M}$. The set of solutions for G under $\mathcal{M}$ is denoted by $\text{Sol}_{\mathcal{M}}(G)$.

As we mentioned, RGEDs are our essential building block for constructing mappings for graph-structured data. This is done as follows. Let $\mathfrak{R}$ be a finite set of RGEDs from Σ to Σ'. We say that a mapping $\mathcal{M}$ from Σ to Σ' is *specified* by $\mathfrak{R}$, denoted by $\mathcal{M} = (\Sigma, \Sigma', \mathfrak{R})$, if

$$(G, G') \in \mathcal{M} \;\text{ iff }\; (G, G') \models \mathcal{T} \text{ for every } \mathcal{T} \in \mathfrak{R}.$$

Whenever $\mathcal{M}$ is a mapping specified by a set of RGEDs, we say that $\mathcal{M}$ is an *RGED-based mapping*, or just an *RGED-mapping*. If each RGED in the set is full, we call $\mathcal{M}$ a *full* RGED-mapping.

The following result implies that we can concentrate in mappings specified by a single RGED. This follows easily from the semantics of RGEDs.

Proposition 3.5. *Let Σ and Σ' be finite alphabets and $\mathfrak{R} = \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ a finite set of RGEDs from Σ to Σ'. Consider the RGED $\mathcal{T} = \mathcal{T}_1 + \cdots + \mathcal{T}_n$, and let $\mathcal{M}$ be the mapping specified by $\mathfrak{R}$, and $\mathcal{M}'$ the mapping specified by $\{\mathcal{T}\}$. Then $\mathcal{M} = \mathcal{M}'$.*

Thus, from now on we assume that every RGED-mapping is specified by a single RGED, and we write $\mathcal{M} = (\Sigma, \Sigma', \mathcal{T})$ for a mapping specified by RGED $\mathcal{T}$. Since full RGEDs are closed under union, we can also assume that full RGED-mappings are specified by a single full RGED.

3.3 Relative expressiveness of RGED-mappings

Two other formalisms for specifying mappings over graph-structured have received attention in the literature. These formalisms are called 2RPQ- and C2RPQ-based mappings, respectively [13, 14, 15].

Let Σ and Σ' be finite alphabets. A *2RPQ-based* mapping $\mathcal{M}$ from Σ into Σ' is a set of rules of the form $r_1 \to r_2$, where r_1 and r_2 are 2RPQs over Σ and Σ', respectively. Given graph databases G and G' over Σ and Σ', respectively, it is the case that $(G, G') \in \mathcal{M}$ if and only if $\llbracket r_1 \rrbracket_G \subseteq \llbracket r_2 \rrbracket_{G'}$. Not surprisingly, it is possible to prove that our RGED-mappings can express 2RPQ-based mappings.

Proposition 3.6. *For every 2RPQ-based mapping $\mathcal{M}$ there exists an RGED-based mapping $\mathcal{M}'$ such that $\mathcal{M} = \mathcal{M}'$.*

The definition of C2RPQ-based mappings is obtained from the definition of 2RPQ-based mappings by replacing the role of 2RPQs with C2RPQs. Although these mappings are not subsumed by our RGED-mappings, they suffer from the drawback of being based on conjunctive queries, and hence, as we have mentioned, being impractical for current applications of graph databases which are often massive.

4. GRAPH EXCHANGE WITH FULL RGEDS

Data exchange is one of the main applications of schema mappings [21]. We start by studying graph data exchange under *full* RGED-mappings, that is, we use full RGED-mappings for specifying how to translate graph-structured data from a source into a target schema. Recall that a full RGED-mapping is one of the form $(\Sigma, \Sigma', \mathcal{T})$, where $\mathcal{T}$ is a full RGED, i.e. a regular expression over the alphabet

$\{(nexp/a) \mid nexp \in \text{NREG}(\Sigma),\ a \in \Sigma'\}$. For instance, rule $\mathcal{T}_1 = (\texttt{creator}^-/\texttt{makes})(\texttt{partOf} \cdot \texttt{series}/\texttt{inConf})$ introduced in Section 1 specifies a full RGED mapping from Σ into Σ'.

The reason why we chose to start our study with full RGED-mappings is that, as we will see in this section, they form a well-behaved class for graph data exchange inside the class of RGED-mappings. At the same time, they can express interesting exchange properties, as shown by rule $\mathcal{T}_1$. In very rough terms, full RGED-mappings allow us to specify the interpretation of symbols in the target graph database based on NREs views over the source data (analogously to what full stds do over relational databases).

Assume we have an RGED-mapping $\mathcal{M}$ from a source alphabet $\Sigma_{\mathbf{S}}$ into a target alphabet $\Sigma_{\mathbf{T}}$, and a source graph database $G_{\mathbf{S}}$. The *graph data exchange problem* consists in computing a target graph database $G_{\mathbf{T}}$ that is a solution for $G_{\mathbf{S}}$ under $\mathcal{M}$ (i.e. $G_{\mathbf{T}} \in \text{Sol}_{\mathcal{M}}(G_{\mathbf{S}})$). As in the relational and XML scenarios, solutions for a source database are not unique (and, indeed, there is an infinite number of them [5]). Thus, in data exchange one usually wants to compute a "universal representative" [21, 5, 7], which is essentially a finite representation of the set of all solutions.

In this section we show that, in the case of full RGED-mappings, one can construct for each source graph database $G_{\mathbf{S}}$ a target graph database $G_{\mathbf{T}}$ that is a universal representative for $G_{\mathbf{S}}$. Not only that, actually in this case we can construct a target graph database $G_{\mathbf{T}}$ that works as a "minimal representative" for $G_{\mathbf{S}}$, in the sense that $G_{\mathbf{T}}$ is contained in any other solution for $G_{\mathbf{S}}$. More importantly, we provide an algorithm that computes $G_{\mathbf{T}}$ in polynomial time in *combined* complexity, i.e. assuming both the source data and the mapping as input to the problem. This result is particularly useful, as it will become an important tool when solving some of the main problems associated with graph data exchange, such as query answering.

4.1 Computing minimal representatives

Intuitively, universal representatives can be seen as *most general* solutions, since they can be embedded into any other solution. In the present context we are interested in even more restrictive representatives, that are *contained* in every other solution.

Definition 4.1 (Minimal representative). *Let $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$ be a full RGED-mapping and $G_{\mathbf{S}}$ a graph database over $\Sigma_{\mathbf{S}}$. A graph database $G_{\mathbf{T}}$ is a* minimal representative *of $G_{\mathbf{S}}$ under $\mathcal{M}$ iff $G_{\mathbf{T}} \subseteq G$, for each $G \in \text{Sol}_{\mathcal{M}}(G_{\mathbf{S}})$.*

The procedure Minimal Representative($G_{\mathbf{S}}, \mathcal{M}$), shown below, specifies how to construct a minimal representative for a source graph database $G_{\mathbf{S}}$ under a full RGED-mapping $\mathcal{M}$.

Algorithm: Minimal Representative($G_{\mathbf{S}}, \mathcal{M}$)
Input: A graph database $G_{\mathbf{S}} = (V, E)$ over $\Sigma_{\mathbf{S}}$ and a full RGED-mapping $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$.
Output: A graph database $G_{\mathbf{T}}$ that is a minimal representative of $G_{\mathbf{S}}$ under $\mathcal{M}$.

1. Construct an NFA $\mathcal{N}_{\mathcal{T}}$ over alphabet $\mathcal{A}(\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}})$ that is equivalent to $\mathcal{T}$.
2. Construct the *product automaton (graph database)* $G_{\mathbf{S}} \times \mathcal{N}_{\mathcal{T}}$ over Σ_2 as follows:
   2.1. The nodes of $G_{\mathbf{S}} \times \mathcal{N}_{\mathcal{T}}$ are $V \times Q$, assuming that Q are the states of $\mathcal{N}_{\mathcal{T}}$.
   2.2. Include a transition from node (n_1, q_1) to (n_2, q_2) with label $a \in \Sigma_2$ iff $\mathcal{N}_{\mathcal{T}}$ contains a transition of the form $(q_1, (nexp/a), q_2)$ and $(n_1, n_2) \in [\![nexp]\!]_{G_{\mathbf{S}}}$.
3. Compute a target graph database G' by removing from $G_{\mathbf{S}} \times \mathcal{N}_{\mathcal{T}}$ all nodes (n, q) such that:
   - (n, q) is not reachable from a node of the form (n', q_0), with $n' \in V$, or
   - from (n, q) one cannot reach a node of the form (n'', q_f), with $n'' \in V$ and $q_f \in F$.
4. Finally, compute $G_{\mathbf{T}}$ by taking the *projection* of the nodes of G' over V. That is, replace each node id of the form (n, q) in G' with n. $\square$

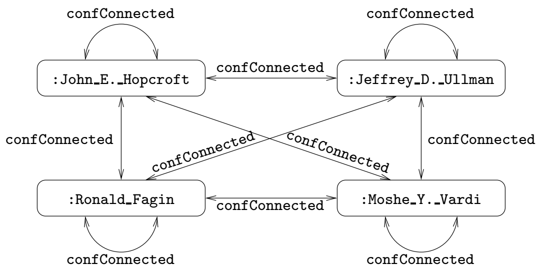

Figure 3: Result of exchanging the graph database in Figure 2 with $(\texttt{makes} \cdot \texttt{makes}^-)^+/\texttt{confConnected}$.

The following theorem shows the correctness of the algorithm and provides its running time.

Theorem 4.2. *Let $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$ be a full RGED-mapping and $G_{\mathbf{S}}$ a graph database over $\Sigma_{\mathbf{S}}$. Then* Minimal Representative*($G_{\mathbf{S}}, \mathcal{M}$) computes a graph database $G_{\mathbf{T}}$ that is a minimal representative for $G_{\mathbf{S}}$ under $\mathcal{M}$. Moreover, the algorithm runs in time $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{T}|)$.*

Example 4.3. Consider again the full RGED

$$\mathcal{T}_1 \;=\; (\texttt{creator}^-/\texttt{makes})(\texttt{partOf} \cdot \texttt{series}/\texttt{inConf})$$

presented in Section 1, and let $\mathcal{M}_1 = (\Sigma_1, \Sigma_2, \mathcal{T}_1)$. When exchanging data with $\mathcal{M}_1$ from the graph database G_1 in Figure 1, Minimal Representative($G_1, \mathcal{M}_1$) returns the graph database G_2 in Figure 2. Moreover, consider Σ_3 and

$$\mathcal{T}_2 \;=\; \big((\texttt{makes} \cdot \texttt{makes}^-)^+/\texttt{coauthor}\big)$$

as in Example 3.2, and let $\mathcal{M}_2 = (\Sigma_2, \Sigma_3, \mathcal{T}_2)$. Then when using $\mathcal{M}_2$ for exchanging data from G_2, one obtains the graph database G_3 over Σ_3 shown in Figure 3. $\square$

Most of the polynomial-time results for constructing universal representatives in relational and XML data exchange assume the mapping specification to be fixed [21, 6]. Moreover, it is known that the complexity of the problem when both the source database and the specification are given as input, that is, the combined complexity, is intractable [28]. Theorem 4.2 proves that the combined complexity of the problem in graph data exchange with full RGEDs is polynomial, which is particularly relevant given the size of current graph database applications. The next result shows that the bound in Theorem 4.2 is actually tight.

PROPOSITION 4.4. *There exist families of full RGED-mappings $\{\mathcal{M}_n = (\Sigma^n_{\mathbf{S}}, \Sigma^n_{\mathbf{T}}, \mathcal{T}_n)\}_{n\geq 1}$ and source graph databases $\{G^m_{\mathbf{S}}\}_{m\geq 1}$ such that $|\mathcal{T}_n|$ is $O(n)$, $|G^m_{\mathbf{S}}|$ is $O(m)$, and every minimal representative for $G^m_{\mathbf{S}}$ under $\mathcal{M}_n$ is of size $\Omega(m^2 \cdot n)$.*

4.2 Query Answering

One of the main tasks in data exchange is answering queries posed over the target schema [5]. Recall that in data exchange one is interested in computing the *certain* answers of queries [21, 6, 5]. Intuitively, these are the answers that hold regardless of the solution one chooses to materialize. In formal terms, given an RGED-mapping $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$, a graph database $G_{\mathbf{S}}$ over $\Sigma_{\mathbf{S}}$, and a query Q over $\Sigma_{\mathbf{T}}$, we define the certain answers of Q w.r.t. $G_{\mathbf{S}}$ under $\mathcal{M}$, denoted by $\textsc{certain}_{\mathcal{M}}(Q, G_{\mathbf{S}})$, as the set

$$\bigcap\{[\![Q]\!]_{G_{\mathbf{T}}} \mid G_{\mathbf{T}} \in \mathrm{Sol}_{\mathcal{M}}(G_{\mathbf{S}})\},$$

where $[\![Q]\!]_{G_{\mathbf{T}}}$ denotes the evaluation of Q over $G_{\mathbf{T}}$. We are interested in the complexity of the following problem:

Problem:	CERTAINANSWERS
Input:	Mapping $\mathcal{M}$ from $\Sigma_{\mathbf{S}}$ to $\Sigma_{\mathbf{T}}$, graph database $G_{\mathbf{S}}$ over $\Sigma_{\mathbf{S}}$, n-ary query Q over $\Sigma_{\mathbf{T}}$, and n-ary tuple $\bar{t} \in \mathbf{V}^n$.
Question:	Is $\bar{t} \in \textsc{certain}_{\mathcal{M}}(Q, G_{\mathbf{S}})$?

Notice that the source graph, the mapping and the query are part of the input, and thus, we are considering the combined complexity of the problem. The next result shows that we can decide the certain-answers problem for NREs in polynomial time in combined complexity.

PROPOSITION 4.5. *The* CERTAINANSWERS *problem for the class of full RGED-mappings and NREs can be solved in time $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{T}| \cdot |nexp|)$.*

To prove this result we use the fact that a minimal representative can be constructed in time $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{T}|)$ for full RGED-mappings, and then show that the certain answers of NREs can be obtained by simply evaluating the query over the minimal representative. More precisely, given an NRE *nexp*, a full RGED mapping $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$, and a graph database $G_{\mathbf{S}}$ over $\mathbf{S}$, we use the following procedure to compute $\textsc{certain}_{\mathcal{M}}(nexp, G_{\mathbf{S}})$:

- Compute $G_{\mathbf{T}}$:=MINIMAL REPRESENTATIVE$(G_{\mathbf{S}}, \mathcal{M})$.
- Output $\textsc{certain}_{\mathcal{M}}(nexp, G_{\mathbf{S}})$ as $[\![nexp]\!]_{G_{\mathbf{T}}}$.

From Proposition 2.2 and Theorem 4.2, it is immediate that this approach can be performed in time $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{T}| \cdot |nexp|)$. The proof that this procedure effectively computes the certain answers for *nexp* over $G_{\mathbf{S}}$ under $\mathcal{M}$ follows easily from the fact that NREs are *monotone*, i.e. they are closed under adding tuples to the graph database. The kind of procedure presented above is standard in data exchange scenarios [5].

Notably, we can even compute certain answers for NREs in linear time in the size of the data, if we allow for a slight increase in terms of the size of the mapping.

THEOREM 4.6. *The* CERTAINANSWERS *problem for the class of full RGED-mappings and NREs can be solved in time $O(|G_{\mathbf{S}}| \cdot |\mathcal{T}|^2 \cdot |nexp|)$.*

It follows from the previous theorem that the *data* complexity of the problem, i.e. the complexity measured only in terms of the source database, is linear. Notice that by the lower bound proved in Proposition 4.4, one cannot use the materialization of a minimal representative to prove Theorem 4.6. Instead the proof of such theorem is based on *query rewriting* techniques, which we explain in the next section.

4.3 Query rewriting for full RGED-mappings

In the usual data exchange scenario, we are given a graph $G_{\mathbf{S}}$ over $\Sigma_{\mathbf{S}}$, and one wishes to obtain the certain answers of an NRE *nexp* over the target schema $\Sigma_{\mathbf{T}}$ with respect to $G_{\mathbf{S}}$ under a mapping $\mathcal{M}$. The literature has identified two main strategies for obtaining the certain answers. We have already seen the first technique: One first materializes a minimal representative for the set of solutions of $G_{\mathbf{S}}$ under $\mathcal{M}$, and then uses this representative for obtaining the certain answers.

The second technique is based on the notion of *rewriting*, which first arose in the contexts of query answering using views [30] and data integration [29]. The idea is to construct, from $\mathcal{M}$ and *nexp*, a rewriting of *nexp* over the source schema. This rewriting is another NRE *nexp*$'$, now over $\Sigma_{\mathbf{S}}$, such that $[\![nexp']\!]_{G_{\mathbf{S}}}$ coincides exactly with $\textsc{certain}_{\mathcal{M}}(nexp, G_{\mathbf{S}})$; that is, for each pair (u, v) of node ids, $(u, v) \in [\![nexp']\!]_{G_{\mathbf{S}}}$ iff $(u, v) \in \textsc{certain}_{\mathcal{M}}(nexp, G_{\mathbf{S}})$. Naturally, the advantage of this approach is that one can then compute the certain answers of *nexp* without materializing a representative. Moreover, as we have hinted in the previous section, by doing so one can reduce the data complexity of query answering in graph data exchange.

It is important to mention that rewritings have been heavily used in the context of graph databases in order to accomplish the task of answering queries using views [13, 14]. In such context, views and queries are usually defined by 2RPQs or 2CRPQs. Here we adapt rewriting techniques to deal with full RGED-mappings and queries given as NREs.

Before studying rewritings in our context, we need to introduce some new terminology and procedures that are key to our algorithms. Given an alphabet Σ, we denote by REG(Σ) the set of all usual regular expressions that can be built using symbols from Σ.

Inverse of a nested expression Let *nexp* be a NRE over Σ. An *inverse* $nexp^{-1}$ of *nexp* is an NRE over Σ that satisfies $(u, v) \in [\![nexp]\!]_G$ iff $(v, u) \in [\![nexp^{-1}]\!]_G$, for each graph database G over Σ and pair (u, v) of node ids in G. For example, consider again the NRE $\texttt{creator}^-[\texttt{partOf} \cdot \texttt{series}]$ over alphabet Σ_1 presented in the previous sections. Intuitively, it retrieves all pairs (u, v) of node ids such that there is an edge labeled `creator` from v to u, and a path labeled $\texttt{partOf} \cdot \texttt{series}$ starting from v. It is not hard to see that an inverse of this NRE can be represented by means of the expression $nexp^{-1} = [\texttt{partOf} \cdot \texttt{series}] \cdot \texttt{creator}$.

There is a simple recursive procedure INV(*nexp*) that, given an NRE *nexp*, computes an inverse $nexp^{-1}$ for *nexp* in linear time. We skip the presentation of such procedure for the sake of space. Details can be found in the full version of the paper.

Remnant of a regular expression We also need to introduce the notion of *remnant* of a regular expression. Let Σ be a finite alphabet. Given a letter $a \in \Sigma$, and a regular expression *exp* over Σ, a remnant of the expression *exp* with respect to letter a, denoted as $rem_a(exp)$, is a set $\{(e_1, s_1), \ldots, (e_n, s_n)\}$ of pairs of regular expressions over Σ such that: (1) For each pair of words $u, v \in \Sigma^*$, if $u \cdot a \cdot v \in L(exp)$, then $u \in L(e_i)$ and $v \in L(s_i)$, for some $1 \leq i \leq n$, and (2) the language $\bigcup_{1 \leq i \leq n} L(e_i \cdot a \cdot s_i)$ is contained in $L(exp)$.

Intuitively, the remnant of *exp* with respect to a contains the information of all the different ways in which one can create a word that mentions the symbol a in the language of *exp*. For instance, consider the expression ab^*cb. Then the remnant of *exp* with respect to c is $rem_c(exp) = \{(ab^*, b)\}$. On the other hand, $rem_b(exp) = \{(ab^*, b^*cb), (ab^*c, \varepsilon)\}$.

It is easy to prove that remnants need not be unique. However, as the following lemma states, remnants of *small* size can always be constructed efficiently. The proof of the lemma can be found in the full version of this paper.

Lemma 4.7. *Let exp be a regular expression over Σ. For each letter $a \in \Sigma$, there is a remnant of exp with respect to a that contains at most m pairs, where m is the number of occurrences of a in exp. Moreover, such remnant can be constructed in time $O(m \cdot |exp|)$.*

Due to Lemma 4.7, we can assume as a black box the existence of a procedure Remnant(*exp*,a). It receives as input a regular expression *exp* and a letter a, and outputs in time $O(m \cdot |exp|)$ a remnant of *exp* with respect to a of at most m tuples, where m is the number of occurrences of a in *exp*.

The Rewriting Algorithm Before showing our rewriting algorithm, we illustrate the construction by means of an example. Consider again the RGED

$$\mathcal{T}_1 = (\mathtt{creator}^-/\mathtt{makes})(\mathtt{partOf} \cdot \mathtt{series}/\mathtt{inConf})$$

presented in Section 1. Assume now that one wants to obtain a rewriting of the NRE

$$nexp := (\mathtt{makes} \cdot \mathtt{makes}^-)^+$$

over mapping $\mathcal{M} = (\Sigma_1, \Sigma_2, \mathcal{T}_1)$. Intuitively, a pair of authors (u, v) belongs to the certain answers of *nexp* over G_1 under $\mathcal{M}$ if and only if u and v are two different authors connected via a path of conference paper coauthors of arbitrary length. Recall that G_1 is the graph database depicted in Figure 1.

We use an inductive approach for rewriting queries. For the case of *nexp*, our first task is rewriting the subexpression `makes`. In other words, we seek to obtain an NRE $nexp'$ such that, for each graph database G over Σ_1, a pair of node ids (u, v) belongs to $[\![nexp']\!]_G$ if and only if u is connected to v via an `makes`-labeled edge in every G' that belongs to $\mathrm{Sol}_{\mathcal{M}}(G)$.

The first idea that comes into mind is rewriting the expression `makes` simply as $\mathtt{creator}^-$. However, this rewriting is not correct. Indeed, $\mathtt{creator}^-$ does not help selecting authors of conference papers (which is precisely what we want), but of all authors in the source database (even if they have no conference papers).

On the other hand, the NRE $\mathtt{creator}^-[\mathtt{partOf} \cdot \mathtt{series}]$ retrieves precisely what we want over each graph database G labeled in Σ_1: The set of all pairs (u, v) such that (u, v) is connected by an edge labeled `makes` in any target graph database that belongs to $\mathrm{Sol}_{\mathcal{M}}(G)$. The rewriting of the subexpression $\mathtt{makes}^-$ is the inverse of the above query, which according to the procedure Inv corresponds to the query $[\mathtt{partOf} \cdot \mathtt{series}] \cdot \mathtt{creator}$. From this, it is easy to derive that the NRE

$$\big(\mathtt{creator}^-[\mathtt{partOf} \cdot \mathtt{series}] \cdot [\mathtt{partOf} \cdot \mathtt{series}] \cdot \mathtt{creator}\big)^+$$

corresponds to the rewriting of *nexp* over mapping $\mathcal{M}$.

It is worth noticing that the rewriting of *nexp* makes use of the nesting operator $[\cdot]$, although neither *nexp* nor $\mathcal{T}_1$ uses such operator. In a sense, this suggests (as it is indeed the case) that the presence of the nesting operator is crucial for the rewriting algorithm, even for queries and mappings that do not make use of it.

We are now ready to provide the details of our rewriting algorithm. For simplicity of exposition, we assume that the input expression *nexp* is not equivalent to the language ε that only consists of the empty word, and does not use the symbol ε. For the same reason we only consider the $+$ operator, instead of the Kleene-star $*$. The general algorithm that considers the remaining cases can be found in the full version of the paper. Given an RGED $\mathcal{T}$, the *source projection* of $\mathcal{T}$, denoted by $\mathtt{sp}(\mathcal{T})$, is the NRE obtained from $\mathcal{T}$ by replacing every symbol of the form (e/f) in $\mathcal{T}$ by e.

```
Algorithm: QRew($nexp$,$\mathcal{M}$)
Input: Mapping $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$ and NRE $nexp$ over $\Sigma_{\mathbf{T}}$.
Output: A NRE $nexp'$ that is an rewriting of $nexp$ over $\mathcal{M}$.

1. If $nexp = a$, for some $a \in \Sigma_{\mathbf{T}}$. Let $\{n_1, \ldots, n_p\}$ be the set
   of all NREs over $\Sigma_{\mathbf{S}}$ such that symbol $(n_i/a)$ $(1 \leq i \leq p)$
   is mentioned in $\mathcal{T}$. For each $i = 1, \ldots, p$:
     1.1 Let $\{(E_1, S_1), \ldots, (E_m, S_m)\}$ be the output of the
         procedure Remnant($\mathcal{T}$,$(n_i/a)$). (Notice that this
         output corresponds to a set of pairs of RGEDs from
         $\Sigma_{\mathbf{S}}$ into $\Sigma_{\mathbf{T}}$).
     1.2 Construct the NRE
         $nexp'_i = \Sigma_{1 \leq j \leq m}[(\mathtt{sp}(E_j))^{-1}] \cdot n_i \cdot [\mathtt{sp}(S_j)]$.
   Then return $nexp' := \Sigma_{1 \leq i \leq p}\, nexp'_i$.
2. If $nexp = a^-$, for $a \in \Sigma_{\mathbf{T}}$, then $nexp'$ :=Inv(QRew($a$,$\mathcal{M}$)).
3. If $nexp = nexp_1 \cdot nexp_2$ then
       $nexp'$ :=QRew($nexp_1$,$\mathcal{M}$)$\cdot$QRew($nexp_2$,$\mathcal{M}$).
4. If $nexp = nexp_1 + nexp_2$ then
       $nexp'$ :=QRew($nexp_1$,$\mathcal{M}$)+QRew($nexp_2$,$\mathcal{M}$).
5. If $nexp = (nexp_1)^+$ then $nexp'$ := (QRew($nexp$,$\mathcal{M}$))$^+$.
6. If $nexp = [nexp_1]$ then $nexp'$ := [QRew($nexp_1$,$\mathcal{M}$)].   □
```

The following theorem states the correctness of the rewriting algorithm. It also provides an upper bound for the size of the rewriting of an NRE, which follows from an easy analysis of the procedures QRew, Remnant and Inv. The proof can be found in the full version of the paper.

Theorem 4.8. *Let $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$ be a full RGED-mapping and nexp an NRE over $\Sigma_{\mathbf{T}}$. Assume that $nexp'$ is the result of* QRew*(nexp,$\mathcal{M}$). Then:*

1. *$|nexp'|$ is $O(|\mathcal{T}|^2 \cdot |nexp|)$, and*

2. *$(u, v) \in \textsc{certain}_{\mathcal{M}}(G, nexp)$ iff $(u, v) \in [\![nexp']\!]_G$, for each graph database G over $\Sigma_{\mathbf{S}}$ and pair (u, v) of node ids in G.*

With the help of procedure QRew($nexp$,$\mathcal{M}$), we can now prove Theorem 4.6.

Proof of Theorem 4.6. Given a full RGED-mapping $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$, a graph database $G_{\mathbf{S}}$ over $\Sigma_{\mathbf{S}}$, a pair of nodes (u, v) from $G_{\mathbf{S}}$ and an NRE $nexp$ over $\Sigma_{\mathbf{T}}$, we proceed as follows to check whether $(u, v) \in \text{CERTAIN}_{\mathcal{M}}(nexp, G_{\mathbf{S}})$:

1. First obtain $nexp'$ =QRew($nexp$,$\mathcal{M}$).
2. Check wether $(u, v) \in [\![nexp']\!]_{G_{\mathbf{S}}}$ using the algorithm mentioned in Proposition 2.2.

From Proposition 2.2 the second step can be performed in time $O(|G_{\mathbf{S}}| \cdot |nexp'|)$, and hence from Theorem 4.8 in time $O(|G_{\mathbf{S}}| \cdot |\mathcal{T}|^2 \cdot |nexp|)$. □

4.4 Composition of full RGEDs

The increasing use of schema mappings in data interoperability applications has led to an extensive interest in not only using them as tools for exchanging data, but also manipulating them as first class citizens. One of the most fundamental operations on schema mappings is *composition*. In particular, this operator has been thoughtfully studied for the relational case [31, 33, 22, 35] and, more recently, also for XML schema mappings [3]. Nevertheless, the composition of schema mappings for graph databases has not yet been considered in the literature.

Just as in the relational and XML cases, we define the semantics of the composition of mappings based on the set-theoretical composition of binary relations. That is, given mappings $\mathcal{M}_1$ from Σ_1 to Σ_2, and $\mathcal{M}_2$ from Σ_2 to Σ_3, the composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ is the mapping from Σ_1 to Σ_3 defined by $\mathcal{M}_1 \circ \mathcal{M}_2 = \{(G_1, G_3) \mid \text{there exists } G_2 \text{ over } \Sigma_2 \text{ such that } (G_1, G_2) \in \mathcal{M}_1 \text{ and } (G_2, G_3) \in \mathcal{M}_2\}$ [33, 22]. Given a mapping language $\mathcal{L}$, we say that $\mathcal{L}$ is *closed under composition* if for any two mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ specified in $\mathcal{L}$, the mapping $\mathcal{M}_1 \circ \mathcal{M}_2$ can also be specified in $\mathcal{L}$.

In [22] the authors showed that the language of full stds behaves nicely with respect to composition. In particular, they exploit rewriting techniques to provide an algorithm that composes two relational mappings defined by full stds, and outputs as a result a mapping of the same form. This implies that the language of mappings described by full stds is closed under composition. We show below that essentially the same conceptual idea can be used to provide a composition algorithm for full RGED-mappings. The algorithm relies heavily on the rewriting procedure QRew($nexp$,$\mathcal{M}$) defined in the previous section.

Algorithm: Compose($\mathcal{M}_{12}$,$\mathcal{M}_{23}$)
Input: Two full RGED-mappings $\mathcal{M}_{12} = (\Sigma_1, \Sigma_2, \mathcal{T}_{12})$ and $\mathcal{M}_{23} = (\Sigma_2, \Sigma_3, \mathcal{T}_{23})$.
Output: A full RGED-mapping $\mathcal{M}_{13} = (\Sigma_1, \Sigma_3, \mathcal{T}_{13})$ that defines the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$.

```
1 Return $\mathcal{T}_{13}$ defined as the RGED that is obtained from
  $\mathcal{T}_{23}$ by replacing each occurence of a symbol $(e/a)$, where
  $e \in \text{NREG}(\Sigma_2)$ and $a \in \Sigma_3$, by $(e'/a)$, where $e' :=$
  QRew($e$, $\mathcal{M}_{12}$). □
```

We can show that the algorithm is correct:

Theorem 4.9. *Given full RGED-mappings $\mathcal{M}_{12} = (\Sigma_1, \Sigma_2, \mathcal{T}_{12})$ and $\mathcal{M}_{23} = (\Sigma_2, \Sigma_3, \mathcal{T}_{23})$, the full mapping $\mathcal{M}_{13}$ = Compose($\mathcal{M}_{12}$,$\mathcal{M}_{23}$) specifies $\mathcal{M}_{12} \circ \mathcal{M}_{23}$.*

Theorem 4.9 implies that full RGED-mappings are closed under composition, in the sense that the composition of two full RGED-mappings can always be specified as a full RGED-mapping. Hence full RGED-mappings share over graph databases the good composition properties of mappings specified by full stds over relational databases.

5. EXTENSION TO GENERAL RGEDS

5.1 Universal representatives

In section 4.1 we showed how to obtain minimal representatives for source graph databases under full RGED-mappings. It is then natural to ask wether one can generalize the algorithm Minimal Representative to compute a minimal representative in the scenario when a mapping is specified by an arbitrary (i.e. not necessarily full) RGED-mapping. Unfortunately, as the next result shows, graph databases as minimal representatives do not necessarily exist in this general scenario.

Proposition 5.1. *There exists an RGED-mapping $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$ and a source graph database $G_{\mathbf{S}}$ such that no graph database over $\Sigma_{\mathbf{T}}$ is a minimal representative for $G_{\mathbf{S}}$ under $\mathcal{M}$.*

The problem arises because graph databases do not have enough expressive power to capture the space of solutions generated by arbitrary RGED-mappings. This was first hinted in [21] for relational data exchange, and replicates as well in the XML context [6]. To overcome this limitation, it is customary to allow databases with "incomplete" information as representatives [21, 5].

Thus, in order to represent the set of solutions under a RGED-mapping, we use graph databases that combine missing information both at the data level (missing node ids) and at the structural level (missing the precise relationship between nodes). Objects that combine these two types of incompleteness are usually called *patterns* [10].

In our context, a *graph* pattern is esentially a graph database whose edges are labeled by nested regular expressions and that admits some node ids to be missing and replaced by unknown (null) values.

As usual, we define the semantics of graph patterns in terms of homomorphisms. Let $\pi = (N, D)$ be a pattern over Σ. A *homomomorphism* from π into the graph database $G = (V, E)$ is a mapping $h : N \to V$ such that: (i) h is the identity over the node ids mentioned in N, and (ii) for each edge $(u, nexp, v) \in D$ it is the case that $(h(u), h(v)) \in [\![nexp]\!]_G$. We then define the set of graph databases represented by π over Σ, denoted by $\text{Rep}_{\Sigma}(\pi)$, in such a way that $G \in \text{Rep}_{\Sigma}(\pi)$ iff there is a homomorphism from π into G.

We are now ready to formalize the notion of what is a "good" representative under RGED-mappings. Since patterns are not graph databases, we cannot stick to the notion of a minimal representative (used in the context of full RGED-mappings) that is contained into any other solution. Instead, we will have to borrow the notion of universal representative, widely used in the context of relational and XML data exchange, which is essentially a pattern that represents the whole space of solutions.

Definition 5.2 (Universal representative). *Let $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$ be an RGED-mapping and $G_{\mathbf{S}}$ a* *graph database over $\Sigma_{\mathbf{S}}$. A pattern $\pi_{\mathbf{T}}$ is a* universal representative *of $G_{\mathbf{S}}$ under $\mathcal{M}$ if* $\text{Sol}_{\mathcal{M}}(G_{\mathbf{S}}) = \text{Rep}_{\Sigma_{\mathbf{T}}}(\pi_{\mathbf{T}})$.

It is important to note that if $\pi_{\mathbf{T}}$ is a pattern without incomplete information (in other words, just a graph database), then the above definition collapses to the definition of minimal representative. This is because in this case $\text{Rep}_{\Sigma_{\mathbf{T}}}(\pi_{\mathbf{T}})$ coincides with the set of graph databases that contain $\pi_{\mathbf{T}}$.

Having defined the notion of universal representative, our next step is to present an algorithm UNIVERSAL REPRESENTATIVE for computing them. Interestingly, we are able to do so by slightly modifying the algorithm MINIMAL REPRESENTATIVE presented in Section 4.1. Indeed, the only modification comes in step 2, while constructing the product NFA $G_{\mathbf{S}} \times \mathcal{N}_{\mathcal{T}}$. The modified product automaton $G_{\mathbf{S}} \times N_{\mathcal{T}}$ is defined as follows:

a. The nodes of $G_{\mathbf{S}} \times \mathcal{N}_{\mathcal{T}}$ are $V \times Q$.

b. Include a transition from node (n_1, q_1) to (n_2, q_2) with label $nexp_2 \in \Sigma_2$ iff $\mathcal{N}_{\mathcal{T}}$ contains a transition $(q_1, (nexp_1/nexp_2), q_2)$, and $(n_1, n_2) \in [\![nexp_1]\!]_{G_{\mathbf{S}}}$.

That is, we modify the construction by allowing the product automaton to be labeled with NREs, in order to cope with the extra expressive power provided by RGED-mappings over its full version.

As for the full case, we now state the correctness of the modified algorithm.

THEOREM 5.3. *Let $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$ be an RGED-mapping and $G_{\mathbf{S}}$ a graph database over $\Sigma_{\mathbf{S}}$. Then* UNIVERSAL REPRESENTATIVE*$(G_{\mathbf{S}}, \mathcal{M})$ computes a graph pattern $\pi_{\mathbf{T}}$ that is a universal representative for $G_{\mathbf{S}}$ under $\mathcal{M}$. Moreover, the algorithm runs in time $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{T}|)$.*

Thus, we are able to obtain the same polynomial bound for the case of arbitrary RGED-mappings as for full RGED-mappings, except this time the output may not be a graph database, but a graph pattern. Since every full RGED-mapping is an RGED-mapping, Proposition 4.4 implies that the bound for the above theorem is tight.

5.2 Query Answering

Next we deal with query answering. Unfortunately, the positive results for full RGEDs shown in Section 4.2 do not extend over arbitrary RGED-mappings. In fact, even if we only concentrate on the data complexity of the problem (i.e., the complexity when queries and mappings are assumed to be fixed) the following theorem shows that the certain answers problem may be intractable.

THEOREM 5.4. *There is an RGED-mapping $\mathcal{M}$ from Σ to Σ' and an RPQ r given by a single word in $(\Sigma')^+$, such that the problem* CERTAINANSWERS*$(\mathcal{M}, r)$ is* CONP*-hard.*

Theorem 5.4 implies that computing certain answers under RGED-mappings is intractable even for queries not using recursive navigational features. Further, one can also show that the increased complexity is not due to the choice of using NREs in RGEDs, since the lower bound can be obtained even if disallowing the use of the inverse and the nesting operator inside GREDs.

Interestingly enough, we can recover tractability of query answering for a class of mappings that extends full RGED-mappings. A *semifull* RGED-mapping from Σ to Σ' is an RGED-mapping specified by an RGED that is only only allowed to use symbols from $\mathcal{A}(\Sigma, \Sigma')$ of the form $(nexp/w)$, where w is a word over Σ'. It is easy to prove that semifull GRED-mappings are more expressive than the full ones.

It turns out that for mappings specified by semifull RGEDs, we can use rather simple graph patterns as universal representatives. We call a graph pattern *naïve* if it does not make use of regular expressions as edge labels, but do use null values as node ids. We use the term naïve because they can be represented as relational *naïve tables* [27].

PROPOSITION 5.5. *Let $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$ be a semi-full RGED-mapping, and $G_{\mathbf{S}}$ a graph database over $\Sigma_{\mathbf{S}}$. A naïve graph pattern $\pi_{\mathbf{T}}$ which is a universal representative of $G_{\mathbf{S}}$ under $\mathcal{M}$ can be constructed in time $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{T}|)$.*

The proof of this proposition is by a straightforward adaptation of the procedure UNIVERSAL REPRESENTATIVE presented in Section 4.1. This result allows us to prove tractability of query answering under semifull RGED-mappings.

THEOREM 5.6. *The* CERTAINANSWERS *problem for the class of semi-full RGED-mappings and NREs can be solved in time $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{T}| \cdot |nexp|)$.*

To prove this result it is sufficient to combine Proposition 5.5 with the following fact: The certain answers of NREs under semifull RGED-mappings can be obtained by evaluating the NRE over a naïve pattern that works as a universal representative for a source graph database, and then discarding tuples that contain null values (this process is usually known as naïve evaluation in the literature [27]).

Although semifull RGED-mappings preserve the good properties of full RGED-mappings in terms of query answering, they do not preserve the good properties of the latter class in terms of composition. In particular, semifull RGED-mappings are not closed under composition.

PROPOSITION 5.7. *The language of semifull RGED-mappings is not closed under composition. That is, there are semifull RGED-mappings $\mathcal{M}_{12} = (\Sigma_1, \Sigma_2, \mathcal{T}_{12})$ and $\mathcal{M}_{23} = (\Sigma_2, \Sigma_3, \mathcal{T}_{23})$ such that no semifull RGED-mapping $\mathcal{M}_{13}$ from Σ_1 into Σ_3 is equivalent $\mathcal{M}_{12} \circ \mathcal{M}_{23}$.*

The proof of this fact mimicks the proof that relational stds are not closed under composition [22]. This naturally suggests that an extension of the language of semifull RGED-mappings with *second-order* existential quantification may be closed under composition, since similar results have been obtained in the relational context [22]. We plan to deal with this issue in our future work.

6. CONCLUDING REMARKS

We have studied interoperability among graph-structured data stores. In particular, we have proposed the RGED schema-mapping language and studied data exchange and query answering issues for them, showing that RGED-mappings have several good properties. We also presented some interesting tools for full RGED-mappings, such as mapping-based query rewriting, and, on a metadata management level, an algorithm that given two full RGED-mappings returns another RGED-mapping that is equivalent to its composition.

Many interesting problems remain open. A particularly interesting issue has to do with the level of synchronization between source and target paths allowed in RGEDs. For instance, one could think of an RGED capable of forcing that each source path has to be transferred as a target path of the same length. Numerous notions of synchronization have been studied, such as regular relations, or rational relations [20, 26]. It would also be interesting to see how different combinations of target and source languages affect the expressive power and practical uses of RGED-mappings. One could also vary the class of languages defining RGEDs, considering for example context-free grammars instead of regular expressions.

7. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1), 1997.

[3] S. Amano, L. Libkin, and F. Murlak. XML schema mappings. In *PODS*, pages 33–42, 2009.

[4] M. Arenas. Data exchange in the relational and rdf worlds. In *SWIM*, 2011.

[5] M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Relational and XML Data Exchange*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[6] M. Arenas and L. Libkin. XML data exchange: Consistency and query answering. *J. ACM*, 55(2), 2008.

[7] M. Arenas, J. Pérez, and J. L. Reutter. Data exchange beyond complete data. In *PODS*, pages 83–94, 2011.

[8] M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. Composition and inversion of schema mappings. *SIGMOD Record*, 38(3):17–28, 2009.

[9] P. Barceló, C. A. Hurtado, L. Libkin, and P. T. Wood. Expressive languages for path queries over graph-structured data. In *PODS*, pages 3–14, 2010.

[10] P. Barceló, L. Libkin, and J. L. Reutter. Querying graph patterns. In *PODS*, pages 199–210, 2011.

[11] T. Berners-Lee. Linked data – design issues. `http://www.w3.org/DesignIssues/LinkedData.html`, 2006.

[12] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. Containment of conjunctive regular path queries with inverse. In *7th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 176–185, 2000.

[13] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of regular expressions and regular path queries. In *PODS*, pages 194–204, 1999.

[14] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. View-based query processing for regular path queries with inverse. In *PODS*, pages 58–66, 2000.

[15] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Simplifying schema mappings. In *ICDT*, pages 114–125, 2011.

[16] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

[17] M. Consens and A. Mendelzon. GraphLog: A visual formalism for real life recursion. In *9th ACM Symposium on Principles of Database Systems (PODS)*, pages 404–416, 1990.

[18] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A graphical query language supporting recursion. In *SIGMOD Conference*, pages 323–330, 1987.

[19] D2R DBLP bibliography database hosted at L3S research center. `http://dblp.l3s.de/d2r/`.

[20] C. Elgot and J. Mezei. On relations defined by generalized finite automata. IBM J. Res. Develop., (9).

[21] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *TCS*, 336(1):89–124, 2005.

[22] R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *TODS*, 30(4):994–1055, 2005.

[23] S. Flesca and S. Greco. Querying graph databases. In *EDBT*, pages 510–524, 2000.

[24] G. H. L. Fletcher, M. Gyssens, D. Leinders, J. V. den Bussche, D. V. Gucht, S. Vansummeren, and Y. Wu. Relative expressive power of navigational querying on graphs. In *ICDT*, pages 197–207, 2011.

[25] D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *17th ACM Symposium on Principles of Database Systems (PODS)*, pages 139–148, 1998.

[26] C. Frougny and J. Sakarovitch. Synchronized rational relations of finite and infinite words. TCS, (108).

[27] T. Imielinski and W. Lipski. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[28] P. G. Kolaitis, J. Panttaja, and W.-C. Tan. The complexity of data exchange. In *PODS*, pages 30–39, 2006.

[29] M. Lenzerini. Data integration: a theoretical perspective. In *PODS*, pages 233–246, 2002.

[30] A. Y. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS*, pages 95–104, 1995.

[31] J. Madhavan and A. Y. Halevy. Composing mappings among data sources. In *VLDB*, pages 572–583, 2003.

[32] M. S. Martín, C. Gutierrez, and P. T. Wood. Snql: A social networks query and transformation language. In *AMW*, 2011.

[33] S. Melnik. *Generic Model Management: concepts and Algorithms*, volume 2967 of *LNCS*. Springer, 2004.

[34] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comput.*, 24(6):1235–1258, 1995.

[35] A. Nash, P. A. Bernstein, and S. Melnik. Composition of mappings given by embedded dependencies. In *PODS*, pages 172–183, 2005.

[36] J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *J. Web Sem.*, 8(4):255–270, 2010.

[37] M. Y. Vardi. A call to regularity. In *Proceedings of the Paris C. Kanellakis memorial workshop on Principles of computing & knowledge*, PCK50, pages 11–11, 2003.

APPENDIX

A. PROOFS AND INTERMEDIATE RESULTS

A.1 Proof of Proposition 3.6

(1) The proof follows almost immediately from the fact that nested regular expressions extend 2RPQs in expressive power. Let $\mathfrak{R} = \{r_1 \to r'_1, \ldots, r_k \to r'_k)\}$, be a of rules defining a 2RPQ-based mapping. Then consider the RGED

$$\mathcal{T} \ = \ (r_1/r'_1) + (r_2/r'_2) + \cdots + (r_k/r'_k).$$

It is straightforward to prove that $\mathcal{T}$ and $\mathfrak{R}$ specify the same mapping. $\square$

A.2 Proof of Theorem 4.2

The proof follows from a more general result (Theorem 5.3).

A.3 Proof of Proposition 4.4

For every $n \geq 1$ consider the alphabets $\Sigma^n_{\mathbf{S}} = \{a\}$, $\Sigma^n_{\mathbf{T}} = \{b_1, \ldots, b_n\}$, the RGED $\mathcal{T}_n$ from $\Sigma^n_{\mathbf{S}}$ to $\Sigma^n_{\mathbf{T}}$ given by

$$(a^*/b_1) + (a^*/b_2) + \cdots + (a^*/b_n),$$

and let $\mathcal{M}_n = (\Sigma^n_{\mathbf{S}}, \Sigma^n_{\mathbf{T}}, \mathcal{T}_n)$. Notice that $|\mathcal{T}_n|$ is $O(n)$. Now, for every $m \geq 1$ consider the graph database $G^m_{\mathbf{S}}$ over $\Sigma^n_{\mathbf{S}}$ containing the edges

$$(1, a, 2), (2, a, 3), \ldots (m-1, a, m), (m, a, 1),$$

that is, $G^m_{\mathbf{S}}$ is a cycle of length m. Clearly $|G^m_{\mathbf{S}}|$ is $O(m)$. Now notice that for every $1 \leq k \leq n$ and for every pair (i, j) such that $1 \leq i, j \leq m$, it holds that

$$i(a^*/b_k)j \tag{1}$$

is a source matching of $\mathcal{T}_n$ over $G^m_{\mathbf{S}}$. This implies that for every graph database $G_{\mathbf{T}}$ such that $G_{\mathbf{T}} \in \text{Sol}_{\mathcal{M}_n}(G^m_{\mathbf{S}})$ it holds that (1) is also a target matching of $\mathcal{T}_n$ over $G_{\mathbf{T}}$. Thus we have that for every $1 \leq k \leq n$ and for every pair (i, j) such that $1 \leq i, j \leq m$, it holds that $(i, j) \in [\![b_k]\!]_{G_{\mathbf{T}}}$ and thus, for every $1 \leq k \leq n$ and for every pair (i, j) such that $1 \leq i, j \leq m$ we have that $(i, b_k, j) \in G_{\mathbf{T}}$. Therefore, we have that $|G_{\mathbf{T}}| \geq m^2 \cdot n$. Finally, since all the values (i, j) such that $1 \leq i, j \leq m$ are actually values in the source graph $G^m_{\mathbf{S}}$, we have that every pattern that is a universal representative for $G^m_{\mathbf{S}}$ has at least $m^2 \cdot n$ edges. This completes the proof. $\square$

A.4 Proof of Lemma 4.7

Let *exp* be a regular expression over Σ. Similar to what is done with the Glushkov construction, we mark letters from Σ with subscripts to indicate different occurrences of a symbol in *exp*. For example, $ab(a+b)$ becomes $a_1b_1(a_2+b_2)$. We denote by $\gamma(exp)$ the resulting expression. Likewise, let $\Sigma_{\gamma(exp)}$ be the alphabet containing all symbols in $\gamma(exp)$.

Next, given a letter $a \in \Sigma$, let $a_1, \ldots, a_m$ be all the symbols in $\Sigma_{\gamma(exp)}$ that represent a position of the letter a in the expression $\gamma(exp)$. For each such a_i $(1 \leq i \leq m)$, let e_{a_i} be the right quotient of $L(\gamma(exp))$ with respect to $a_i \cdot \Sigma^*_{\gamma(exp)}$; that is, the language

$$\{u \in \Sigma^*_{\gamma(exp)} \mid \text{ there is } v \in L(a_i \cdot \Sigma^*_{\gamma(exp)}) \text{ such that } u \cdot v \in L(\gamma(exp))\},$$

and s_{a_i} be the left quotient of $L(\gamma(exp))$ with respect to $\Sigma^*_{\gamma(exp)} \cdot a_i$ (usually denoted as the *residual* of $\gamma(exp)$ with respect to a_i); that is, the language

$$\{u \in \Sigma^*_{\gamma(exp)} \mid \text{ there is } v \in L(\Sigma^*_{\gamma(exp)} \cdot a_i) \text{ such that } v \cdot u \in L(\gamma(exp))\}.$$

Then, it is easy to see that the pair $\{(e_{a_i}, s_{a_i})\}$ is a remnant of $\gamma(exp)$ with respect to a_i.

Consider now, for each $a_1, \ldots, a_m$, the remnant $\{(e_{a_i}, s_{a_i})\}$ of $\gamma(exp)$ with respect to a_i, as explained above. Then it is not difficult to show, using the Glushkov construction (see e.g. A. Bruggemann-Klein, "Regular expressions into finite automata", Theoretical Computer Science, 120(2):197-213, 1993), that the set

$$\{(\gamma^{-1}(e_{a_i}), \gamma^{-1}(s_{a_i})) \mid 1 \leq i \leq m\}$$

is a remnant of *exp* with respect to a, where γ^{-1} maps each element $a_i \in \Sigma_{\gamma(exp)}$ back to $a \in \Sigma$.

Moreover, for each $1 \leq i \leq m$, the pairs (e_{a_i}, s_{a_i}) can be computed in time $O(n^2)$ (see e.g. Gruber and Holzer, "Language Operations with Regular Expressions of Polynomial Size", Theoretical Computer Science, 410(35): 3281-3289, 2009). Since we need to do this for each occurrence of the letter a in *exp*, the $O(n^2m)$ bound follows. $\square$

A.5 Proof of Theorem 4.8

We first give details on the algorithm we use for computing inverses.

Algorithm: INV(*exp*)
Input: A NRE *exp*.
Output: A NRE exp^{-1} that is an *inverse* of *exp*.

1. If $exp = \varepsilon$, $exp^{-1} = \varepsilon$.
2. If $exp = a$, for $a \in \Sigma$, $exp^{-1} = a^{-1}$.
3. If $exp = a^{-1}$, for $a \in \Sigma$, $exp^{-1} = a$.
4. If $exp = (exp_1 \cdot exp_2)$,
$$exp^{-1} = \textsc{Inv}(exp_2) \cdot \textsc{Inv}(exp_1).$$
5. If $exp = (exp_1 + exp_2)$,
$$exp^{-1} = \textsc{Inv}(exp_1) + \textsc{Inv}(exp_2).$$
6. If $exp = (exp_1)^*$, $exp^{-1} = (\textsc{Inv}(exp_2))^*$.
7. If $exp = [exp_1]$, $exp^{-1} = [\textsc{Inv}(exp_2)]$. □

It is straightforward to prove the correctness of the construction by induction. Moreover, a linear construction can be implemented using usual techniques, by considering any binary parse tree of the expression.

Part 1: The only steps not requiring constant or linear time is the base case. Thus, by a simple recursive analysis we obtain that the algorithm runs in time linear with respect to the base case. Moreover, the complexity of the base case involves computing, for each symbol a in *nexp*, a potential remnant computation of size $O(|\mathcal{T}^3|)$, since in the worst case scenario all pairs in $\mathcal{T}$ produce a. This gives the $O(|\mathcal{T}^3| \cdot |nexp|)$ bound.

Part 2: We give below the full proof, considering ε and kleene star $*$ operators.

Let $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$ be a full RGED-mapping, and *exp* be an NRE over $\Sigma_{\mathbf{T}}$. The *source rewriting* of *exp* w.r.t. $\mathcal{M}$ is the following expression, denoted by $S_{\mathcal{M}}(exp)$, and defined inductively:

- To define $S_{\mathcal{M}}(\varepsilon)$, assume without loss of generality that there are no repeated symbols in $\mathcal{T}$ (if not we can always find equivalent expressions either over $\Sigma_{\mathbf{S}}$ or over $\Sigma_{\mathbf{T}}$ to modify enough symbols of $\mathcal{T}$). Moreover, for each symbol (e/s) in $\mathcal{T}$, assume that $rem_{(e/s)}(\mathcal{T}) = \{(E^{(e/s)}, S^{(e/s)})\}$. Then,
$$S_{\mathcal{M}}(\varepsilon) = \left(\bigcup_{(e/s) \text{ in } \mathcal{T}} [\tau_S(E^{(e/s)})^{-1}][\tau_S((e/s)) \cdot \tau_S(S^{(e/s)})] \right) + [\tau_S(\mathcal{T})^{-1}]$$
- If $exp = a$, for some $a \in \Sigma_{\mathbf{T}}$, let $(e_1/a), \ldots, (e_n/a)$ be the symbols of $\mathcal{T}$ whose target expression is the symbol a. Moreover, assume that $rem_{(e_i/a)}(\mathcal{T}) = \{(E^i_1, S^i_1), \ldots, (E^i_{m_i}, S^i_{m_i})\}$, for each $i \in \{1, \ldots, n\}$. Then define
$$S_{\mathcal{M}}(exp) = \bigcup_{1 \le i \le n} \bigcup_{1 \le j \le m_i} [\tau_S(E^i_j)^{-1}] \cdot e_i \cdot [\tau_S(S^i_j)].$$
- if $exp = a^-$, for some $a \in \Sigma_{\mathbf{T}}$, then $S_{\mathcal{M}}(exp) = (S_{\mathcal{M}}(a))^{-1}$;
- if $exp = exp_1 \cdot exp_2$ then $S_{\mathcal{M}}(exp) = S_{\mathcal{M}}(exp_1) \cdot S_{\mathcal{M}}(exp_2)$;
- if $exp = exp_1 + exp_2$ then $S_{\mathcal{M}}(exp) = S_{\mathcal{M}}(exp_1) + S_{\mathcal{M}}(exp_2)$;
- if $exp = (exp_1)^*$, then $S_{\mathcal{M}}(exp) = S_{\mathcal{M}}(\varepsilon) + S_{\mathcal{M}}(exp_1) \cdot (S_{\mathcal{M}}(exp_1))^*$; and
- if $exp \equiv [exp_1]$ then $S_{\mathcal{M}}(exp) = [S_{\mathcal{M}}(exp_1)]$.

Before continuing with the proof, let us show an example of the application of the rewriting technique that we use later in the appendix.

Example A.1. Let $\Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{d, e\}$. Consider the RGED $\mathcal{T} = (a/d)(b/e)$ and let $\mathcal{M}$ be the mapping from Σ_1 to Σ_2 specified by $\mathcal{T}$. Let *exp* be the NRE d^+. We now show how to compute $S_{\mathcal{M}}(d^+)$. Notice that by the construction it is equivalent to $S_{\mathcal{M}}(d) \cdot (S_{\mathcal{M}}(d))^*$, thus we only need to construct $S_{\mathcal{M}}(d)$. Notice that we have just one symbol with d in the target part, namely, (a/d). It is not difficult to see that $rem_{(a/d)}(\mathcal{T}) = \{(\varepsilon, (b/e))\}$. Thus, given that $\tau_S(\varepsilon) = \varepsilon$ and $\tau_S((b/e)) = b$, we have that
$$S_{\mathcal{M}}(d) \;=\; [\varepsilon^{-1}] \cdot a \cdot [b] \;=\; a \cdot [b]$$
And thus, we have that
$$S_{\mathcal{M}}(d^+) \;=\; S_{\mathcal{M}}(d) \cdot (S_{\mathcal{M}}(d))^* \;=\; (a \cdot [b]) \cdot (a \cdot [b])^* \;=\; (a \cdot [b])^+$$
□

The following lemma is imperative for proving Theorem 4.8.

Lemma A.2. *Let $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$ be a full RGED-mapping, $G_{\mathbf{S}}$ a graph database over $\Sigma_{\mathbf{S}}$, and $G_{\mathbf{T}}$ a universal representative (in fact, a minimal representative) for $G_{\mathbf{S}}$ under $\mathcal{M}$. Consider an NRE exp over $\Sigma_{\mathbf{T}}$. Then, $(n_1, n_2) \in [\![exp]\!]_{G_{\mathbf{T}}}$ if and only if $(n_1, n_2) \in [\![S_{\mathcal{M}}(exp)]\!]_{G_{\mathbf{S}}}$.*

Proof. We prove the lemma using induction. Notice that if the language defined by *exp* does not contain ε, then *exp* can be written as an ε-free expression, and if it does contain ε, then *exp* can be written in the form $\varepsilon + exp'$ were exp' is ε-free. Given this, it is safe to assume that *exp* is either ε or it is ε-free. We assume this in the proof.

- If *exp* is ε, then $(n_1, n_1) \in [\![exp]\!]_{G_{\mathbf{T}}}$ if and only if there is a source matching for $\mathcal{T}$ over $G_{\mathbf{S}}$ that contains the node n_1. It is not difficult to show that this is the case if and only if $(n_1, n_1) \in [\![S_{\mathcal{M}}(exp)]\!]_{G_{\mathbf{S}}}$.

- It is the case that $exp = a$, for some $a \in \Sigma_{\mathbf{T}}$.
  ($\Longrightarrow$): Assume that $(n_1, n_2) \in [\![a]\!]_{G_{\mathbf{T}}}$, that is, there is an edge in $G_{\mathbf{T}}$ from n_1 to n_2 that is labelled a. We show that $(n_1, n_2) \in [\![S_{\mathcal{M}}(a)]\!]_{G_{\mathbf{S}}}$. From the construction of $G_{\mathbf{T}}$, we have that there is a symbol (e/a) such that $n_1(e/a)n_2$ is part of a source matching of E over $G_{\mathbf{S}}$, of form:
  $$v_0(e_1/s_1)v_1(e_2/s_2)v_2 \cdots (e_p/s_p)n_1(e/a)n_2(e_{p+2}, s_{p+2}) \cdots v_{n-1}(e_n/s_n)v_n, \tag{2}$$
  and where $(e_1/s_1) \cdots (e_p/s_p) \cdot (e/a) \cdot (e_{p+2}, s_{p+2}) \cdots (e_n/s_n)$ is a word in $\mathcal{T}$. Let $rem_{(e/a)}(\mathcal{T}) = \{(E_1, S_1), \ldots, (E_m, S_m)\}$. Then by the definition of remnant, there is a $j \in \{1, \ldots, m\}$ such that (1) the word $(e_1/s_1) \cdots (e_p/s_p)$ belongs to E_j, and (2) the word $(e_{p+2}, s_{p+2}) \cdots (e_n/s_n)$ belongs to S_j. Moreover, given that for each $1 \leq i \leq n$ it holds that $(v_{i-1}, v_i) \in [\![e_i]\!]_{G_{\mathbf{S}}}$, it is easy to see that $(v_0, n_1) \in [\![\tau_S(E_j)]\!]_{G_{\mathbf{S}}}$, $(n_1, n_2) \in [\![e]\!]_{G_{\mathbf{S}}}$, and that $(n_2, v_n) \in [\![\tau_S(S_j)]\!]_{G_{\mathbf{S}}}$. This entails that $(n_1, n_2) \in [\![[\tau_S(E_j)^{-1}] \cdot e \cdot [\tau_S(S_j)]]\!]_{G_{\mathbf{S}}}$, which suffices for the proof that $(n_1, n_2) \in [\![S_{\mathcal{M}}(a)]\!]_{G_{\mathbf{S}}}$.

  ($\Longleftarrow$): Assume that $(n_1, n_2) \in [\![S_{\mathcal{M}}(a)]\!]_{G_{\mathbf{S}}}$. We need to show that there is an edge in $G_{\mathbf{T}}$ from n_1 to n_2 that is labelled a. Let $(e_1/a), \ldots, (e_\ell/a)$ be the symbols of $\mathcal{T}$ whose target expression is the symbol a. Moreover, assume that $rem_{(e_k/a)}(\mathcal{T}) = \{(E_1^k, S_1^k), \ldots, (E_{m_k}^k, S_{m_k}^k)\}$, for each $k \in \{1, \ldots, \ell\}$. Then,
  $$S_{\mathcal{M}}(a) = \bigcup_{1 \leq k \leq \ell} \bigcup_{1 \leq j \leq m_k} [\tau_S(E_j^k)^{-1}] \cdot e_i \cdot [\tau_S(S_j^k)].$$
  Thus, given that $(n_1, n_2) \in [\![S_{\mathcal{M}}(a)]\!]_{G_{\mathbf{S}}}$, there must be $k \in \{1, \ldots, \ell\}$ and $j \in \{1, \ldots, m_k\}$ such that $(n_1, n_2) \in [\![[\tau_S(E_j^k)^{-1}] \cdot e_k \cdot [\tau_S(S_j^k)]]\!]_{G_{\mathbf{S}}}$. Then, there are nodes v_0 and v_n in $G_{\mathbf{S}}$, such that $(v_0, n_1) \in [\![\tau_S(E_j^k)]\!]_{G_{\mathbf{S}}}$ and $(n_2, v_p) \in [\![\tau_S(S_j^k)]\!]_{G_{\mathbf{S}}}$. Furthermore, notice that both $\tau_S(E_j^k)$ and $\tau_S(S_j^k)$ can be considered as regular expressions over the alphabet defined by $\Gamma = \{e \mid \mathcal{T}$ contains a symbol of form $(e/s)\}$. Thus, it is possible to construct a sequence
  $$v_0 e_1 v_1 e_2 v_2 \cdots e_p n_1 e n_2 e_{p+2} \cdots v_{n-1} e_n v_n, \tag{3}$$
  such that $e_1 \cdots e_p$ belong to $\tau_S(E_j^k)$ and $e_{p+1} \cdots e_n$ belongs to $\tau_S(S_j^k)$, where now both $\tau_S(E_j^k)$ and $\tau_S(S_j^k)$ are considered as regular expressions over Γ; and such that, if we let $n_1 = v_{p+1}$, and $n_2 = v_{p+2}$, then $(v_{i-1}, v_i) \in [\![e_i]\!]_{G_{\mathbf{S}}}$, for all $i \in \{1, \ldots, n\}$. From the definition of remnant, we then have that $e_1 \cdots e_n$ must belong to the language of $\tau_S(\mathcal{T})$, if we consider $\tau_S(\mathcal{T})$ as a regular expression over Γ as well. Then notice that the above sequence can be extended into a source matching of $G_{\mathbf{S}}$ over $\mathcal{T}$, as follows:
  $$v_0(e_1/s_1)v_1(e_2/s_2)v_2 \cdots (e_p/s_p)n_1(e/a)n_2(e_{p+2}, s_{p+2}) \cdots v_{n-1}(e_n/s_n)v_n, \tag{4}$$
  Since $G_{\mathbf{T}}$ is a solution for $G_{\mathbf{S}}$, then this sequence needs to be a target matching as well, which proves that there exists an edge of form (n_1, a, n_2) in $G_{\mathbf{T}}$.
- It is the case that $exp = a^-$, for some $a \in \Sigma_{\mathbf{T}}$. Then $S_{\mathcal{M}}(a^-) = (S_{\mathcal{M}}(a))^{-1}$. But we know that $(n_1, n_2) \in [\![a]\!]_{G_{\mathbf{T}}}$ if and only if $(n_1, n_2) \in [\![S_{\mathcal{M}}(a)]\!]_{G_{\mathbf{S}}}$. Moreover, $(n_1, n_2) \in [\![a]\!]_{G_{\mathbf{T}}}$ iff $(n_2, n_1) \in [\![a^-]\!]_{G_{\mathbf{T}}}$, and $(n_1, n_2) \in [\![S_{\mathcal{M}}(a)]\!]_{G_{\mathbf{S}}}$ iff $(n_2, n_1) \in [\![(S_{\mathcal{M}}(a))^{-1}]\!]_{G_{\mathbf{S}}}$. The proof then follows.
- We have that $exp = exp_1 \cdot exp_2$. Then $S_{\mathcal{M}}(exp) = S_{\mathcal{M}}(exp_1) \cdot S_{\mathcal{M}}(exp_2)$.
  ($\Longrightarrow$): Assume that $(n_1, n_2) \in [\![S_{\mathcal{M}}(exp)]\!]_{G_{\mathbf{S}}}$. Then, by definition, there is a node n_3 of $G_{\mathbf{S}}$ such that $(n_1, n_3) \in [\![S_{\mathcal{M}}(exp_1)]\!]_{G_{\mathbf{S}}}$, and $(n_3, n_2) \in [\![S_{\mathcal{M}}(exp_2)]\!]_{G_{\mathbf{S}}}$. By induction hypothesis, we have that $(n_1, n_3) \in [\![exp_1]\!]_{G_{\mathbf{T}}}$ and $(n_3, n_2) \in [\![exp_2]\!]_{G_{\mathbf{T}}}$, which entails that $(n_1, n_2) \in [\![exp]\!]_{G_{\mathbf{T}}}$.
  ($\Longleftarrow$): Along the same line as the previous direction.
- We have that $exp = exp_1 + exp_2$. Then $S_{\mathcal{M}}(exp) = S_{\mathcal{M}}(exp_1) + S_{\mathcal{M}}(exp_2)$. The proof is identical to the case of concatenation.
- It is the case that $exp = (exp_1)^*$. Then $S_{\mathcal{M}}(exp) = S_{\mathcal{M}}(\varepsilon) + S_{\mathcal{M}}(exp_1) \cdot (S_{\mathcal{M}}(exp_1))^*$.
  ($\Longrightarrow$): Assume that $(n_1, n_2) \in [\![S_{\mathcal{M}}(exp)]\!]_{G_{\mathbf{S}}}$. Then either $(n_1, n_2) \in [\![S_{\mathcal{M}}(\varepsilon)]\!]_{G_{\mathbf{S}}}$ or $(n_1, n_2) \in [\![S_{\mathcal{M}}(exp_1) \cdot (S_{\mathcal{M}}(exp_1))^*]\!]_{G_{\mathbf{S}}}$. Assume the latter. Then there is at least a $k \geq 1$ such that $(n_1, n_2) \in [\![S_{\mathcal{M}}(exp_1)^k]\!]_{G_{\mathbf{S}}}$, and thus there are nodes $m_1, \ldots, m_{k-1}$ such that $(n_1, m_1) \in [\![S_{\mathcal{M}}(exp_1)]\!]_{G_{\mathbf{S}}}$, $(m_{k-1}, n_2) \in [\![S_{\mathcal{M}}(exp_1)]\!]_{G_{\mathbf{S}}}$ and $(m_j, m_{j+1}) \in [\![S_{\mathcal{M}}(exp_1)]\!]_{G_{\mathbf{S}}}$, for each $j \in \{1, \ldots, k-2\}$. The proof that $(n_1, n_2) \in [\![exp]\!]_{G_{\mathbf{T}}}$ then follows using the induction hypothesis. Assume that $(n_1, n_2) \in [\![S_{\mathcal{M}}(\varepsilon)]\!]_{G_{\mathbf{S}}}$. Then from induction hypothesis we have that $(n_1, n_2) \in [\![\varepsilon]\!]_{G_{\mathbf{T}}}$, which proves that $(n_1, n_2) \in [\![exp]\!]_{G_{\mathbf{T}}}$.
  ($\Longleftarrow$): If $(n_1, n_2) \in [\![exp]\!]_{G_{\mathbf{T}}}$, then there is a $k \geq 0$ such that $(n_1, n_2) \in [\![S_{\mathcal{M}}(exp)^k]\!]_{G_{\mathbf{T}}}$. Assume first that $k = 0$. Then $(n_1, n_2) \in [\![S_{\mathcal{M}}(\varepsilon)]\!]_{G_{\mathbf{T}}}$, and thus from induction hypothesis we have that $(n_1, n_2) \in [\![S_{\mathcal{M}}(\varepsilon)]\!]_{G_{\mathbf{S}}}$, which proves that $(n_1, n_2) \in [\![S_{\mathcal{M}}(exp)]\!]_{G_{\mathbf{S}}}$. The case when $k > 0$ follows the same lines as the other direction.
- For the last case, $exp \equiv [exp_1]$. Then $S_{\mathcal{M}}(exp) = [S_{\mathcal{M}}(exp_1)]$.
  ($\Longrightarrow$): Assume that $(n_1, n_2) \in [\![S_{\mathcal{M}}(exp)]\!]_{G_{\mathbf{S}}}$. Then (1) $n_1 = n_2$, and (2) there is a node n_3 such that $(n_1, n_3) \in [\![S_{\mathcal{M}}(exp_1)]\!]_{G_{\mathbf{S}}}$. From the induction hypothesis, we have that $(n_1, n_3) \in [\![exp_1]\!]_{G_{\mathbf{T}}}$, and thus $(n_1, n_1) \in [\![exp]\!]_{G_{\mathbf{T}}}$.
  ($\Longleftarrow$): Assume that $(n_1, n_2) \in [\![exp]\!]_{G_{\mathbf{T}}}$. Then (1) $n_1 = n_2$, and (2) there is a node n_3 such that $(n_1, n_3) \in [\![exp_1]\!]_{G_{\mathbf{T}}}$. By induction we have that $(n_1, n_3) \in [\![S_{\mathcal{M}}(exp_1)]\!]_{G_{\mathbf{S}}}$, which shows that $(n_1, n_1) \in [\![S_{\mathcal{M}}(exp)]\!]_{G_{\mathbf{S}}}$.

This finishes the proof of the Lemma. $\square$

It is now easy to prove Theorem 4.8 using this Lemma and Proposition 4.5. $\square$

A.6 Proof of Proposition 4.9

Without loss of generality, we show the proof for the case when $\mathcal{M}_{23}$ is an arbitrary mapping. We need to prove that $(G_1, G_3) \in (\Sigma_1, \Sigma_3, \mathcal{T}_{13})$ if and only if there exist a graph G_2 over Σ_2 such that $(G_1, G_2) \in \mathcal{M}_{12}$ and $(G_2, G_3) \in \mathcal{M}_{23}$.

($\Longrightarrow$): Assume that $(G_1, G_3) \in (\Sigma_1, \Sigma_3, \mathcal{T}_{13})$. Moreover, let G_2 be the universal representative for G_1 under $\mathcal{M}_{12}$. Notice that G_2 is a proper graph database over Σ_2, since $\mathcal{M}_{12}$ is full. It is clear that $(G_1, G_2) \in \mathcal{M}_{12}$, since G_2 is a universal representative. Next we show that $(G_2, G_3) \in \mathcal{M}_{23}$. Let

$$v_0(e_1/s_1)v_1(e_2/s_2)v_2 \cdots v_{n-1}(e_n/s_n)v_n,$$

Be an arbitrary source matching for $\mathcal{T}_{23}$ over G_2. We show that it is also a target matching over G_3. To that extent, we claim that

$$v_0(S_{\mathcal{M}_{12}}(e_1)/s_1)v_1(S_{\mathcal{M}_{12}}(e_2)/s_2)v_2 \cdots v_{n-1}(S_{\mathcal{M}_{12}}(e_n)/s_n)v_n,$$

is a source matching for $\mathcal{T}_{13}$ over G_1. Indeed, by the construction of $\mathcal{T}_{13}$ and the fact that $(e_1/s_1)\cdot(e_2/s_2)\cdots(e_n/s_n)$ belongs to the language of $\mathcal{T}_{23}$, we have that $(S_{\mathcal{M}_{12}}(e_1)/s_1)\cdot(S_{\mathcal{M}_{12}}(e_2)/s_2)\cdots(S_{\mathcal{M}_{12}}(e_n)/s_n)$ belongs to the language of $\mathcal{T}_{13}$. Moreover, since for each $i \in \{1, \dots, n\}$ we have that $(v_{i-1}, v_i) \in [\![e_i]\!]_{G_2}$, by Lemma A.2, it is the case that $(v_{i-1}, v_i) \in [\![S_{\mathcal{M}_{12}}(e_i)]\!]_{G_1}$. Thus, it must also be a target matching for $\mathcal{T}_{13}$ over G_3. This clearly implies that

$$v_0(e_1/s_1)v_1(e_2/s_2)v_2 \cdots v_{n-1}(e_n/s_n)v_n,$$

is a target matching for $\mathcal{T}_{23}$ over G_3, which was to be shown.

($\Longleftarrow$): Assume that there exist a graph G_2 over Σ_2 such that $(G_1, G_2) \in \mathcal{M}_{12}$ and $(G_2, G_3) \in \mathcal{M}_{23}$. We have to show that $(G_1, G_3) \in \mathcal{M}_{13}$. Let

$$v_0(S_{\mathcal{M}_{12}}(e_1)/s_1)v_1(S_{\mathcal{M}_{12}}(e_2)/s_2)v_2 \cdots v_{n-1}(S_{\mathcal{M}_{12}}(e_n)/s_n)v_n,$$

be a source matching for $\mathcal{T}_{13}$ over G_1. We need to show that it is a target matching for $\mathcal{T}_{13}$ over G_3. But notice that Lemma A.2 clearly states that

$$v_0(e_1/s_1)v_1(e_2/s_2)v_2 \cdots v_{n-1}(e_n/s_n)v_n,$$

is a source matching for $\mathcal{T}_{23}$ over G_2, and since we assumed that $(G_2, G_3) \in \mathcal{M}_{23}$, we have that the above sequence is a target matching for $\mathcal{T}_{23}$ over G_3. This immediately implies that

$$v_0(S_{\mathcal{M}_{12}}(e_1)/s_1)v_1(S_{\mathcal{M}_{12}}(e_2)/s_2)v_2 \cdots v_{n-1}(S_{\mathcal{M}_{12}}(e_n)/s_n)v_n,$$

if a target matching for $\mathcal{T}_{13}$ over G_3. □

A.7 Proof of Proposition 5.1

Follows directly from Theorem 5.3 and the results about expressibility of patterns in [10]

A.8 Proof of Theorem 5.3

- First we prove that $\text{Sol}(G_{\mathbf{S}}) \subseteq \text{Rep}_{\Sigma_{\mathbf{T}}}(\pi_{\mathbf{T}})$. Take an arbitrary graph $G_{\mathbf{T}} \in \text{Sol}_{\mathcal{M}}(G_{\mathbf{S}})$. To show that G belongs to $\text{Rep}_{\Sigma_{\mathbf{T}}}(\pi_{\mathbf{T}})$, we construct a homomorphism from $\pi_{\mathbf{T}}$ to $G_{\mathbf{T}}$. Clearly, it must be the identity on every nodes of $\pi_{\mathbf{T}}$, since it uses no node variables. Thus, we need to show that for each edge (n_1, exp_2, n_2) it is the case that $(n_1, n_2) \in [\![exp_2]\!]_{G_{\mathbf{T}}}$. Pick then an arbitrary edge (n_1, exp_2, n_2) in $\pi_{\mathbf{T}}$. From the construction of $\pi_{\mathbf{T}}$, we have that $G_{\mathbf{S}} \times N_{\mathcal{T}}$ contains a transition of the form $((n_1, q_1), (exp_1/exp_2), (n_2, q_2))$. Moreover, (n_1, q_1) is reachable from a node (n', q_0), where q_0 is an initial state of $N_{\mathcal{T}}$, and node (n_2, q_2) can reach a node (n'', q_f), where q_f is a final state in $N_{\mathcal{T}}$. Thus, it is easy to see that there is a sequence

$$(v_0, p_0)(e_1/s_1)(v_1, p_1)(e_2/s_2)(v_2, p_2) \cdots (n_1, q_1)(exp_1/exp_2)(n_2, q_2) \cdots (v_{m-1}, p_{m-1})(e_m/s_m)(v_m, p_m),$$

  such that $(v_0, p_0) = (n', q_0)$ and $(v_m, p_m) = (n'', q_f)$. By the construction $N_{\mathcal{T}}$, we have that $(e_1/s_1) \cdots (exp_1/exp_2) \cdots (e_m/s_m)$ is a string in the language defined by $\mathcal{T}$, and thus, by the construction of $G_{\mathbf{S}} \times N_{\mathcal{T}}$, we have that the sequence

$$v_0(e_1/s_1)v_1(e_2/s_2)v_2 \cdots n_1(exp_1/exp_2)n_2 \cdots v_{m-1}(e_m/s_m)v_m,$$

  is a source matching of $\mathcal{T}$ over $G_{\mathbf{S}}$. Thus, since $G_{\mathbf{T}} \in \text{Sol}_{\mathcal{M}}(G_{\mathbf{S}})$ we have that the above sequence is also a target matching of $\mathcal{T}$ over $G_{\mathbf{T}}$, which implies that $(n_1, n_2) \in [\![exp_2]\!]_{G_{\mathbf{T}}}$, completing this part of the proof.

- Now we show that $\text{Rep}_{\Sigma_{\mathbf{T}}}(\pi_{\mathbf{T}}) \subseteq \text{Sol}(G_{\mathbf{S}})$. To that extent, take an arbitrary graph $G_{\mathbf{T}} \in \text{Rep}_{\Sigma_{\mathbf{T}}}(\pi_{\mathbf{T}})$. We need to show that $G_{\mathbf{T}} \in \text{Sol}(G_{\mathbf{S}})$, or, essentially, that every source matching of $\mathcal{T}$ over $G_{\mathbf{S}}$ is also a target matching of $\mathcal{T}$ over $G_{\mathbf{T}}$. Then consider a source matching of $\mathcal{T}$ over $G_{\mathbf{S}}$

$$v_0(e_1/s_1)v_1(e_2/s_2)v_2 \cdots v_{m-1}(e_m/s_m)v_m. \qquad (5)$$

  We next prove that (5) is also a target matching of $\mathcal{T}$ over $G_{\mathbf{T}}$. Since the string $(e_1/s_1)(e_2/s_2)\cdots(e_m/s_m)$ is in the language defined by $\mathcal{T}$ we know that there exists a path in $N_{\mathcal{T}}$ of the form $q_0(e_1/s_1)q_1(e_2/s_2)q_2 \cdots q_{m-1}(e_m/s_m)q_m$

where q_0 is the initial state of $N_{\mathcal{T}}$ and q_m is a final state of $N_{\mathcal{T}}$. Moreover, since (5) is a source matching of $\mathcal{T}$ over $G_{\mathbf{S}}$, we have that $(v_{i-1}, v_i) \in [\![e_i]\!]_{G_{\mathbf{S}}}$. This implies that the following is a path in $G_{\mathbf{S}} \times N_{\mathcal{T}}$

$$(v_0, p_0)(e_1/s_1)(v_1, p_1)(e_2/s_2)(v_2, p_2) \cdots (v_{m-1}, p_{m-1})(e_n/s_n)(v_m, p_m). \tag{6}$$

Pick any arbitrary $i \in \{1, \ldots, m\}$. Given the construction of $\pi_{\mathbf{T}}$ we know that (v_{i-1}, s_1, v_i) is an edge in $\pi_{\mathbf{T}}$. Moreover, since $G_{\mathbf{T}}$ belongs to $\mathrm{Rep}_{\Sigma_{\mathbf{T}}}(\pi_{\mathbf{T}})$, there is a homomorphism from $\pi_{\mathbf{T}}$ to $G_{\mathbf{T}}$, that needs to be the identity on both v_{i-1} and v_i. This entails that $(v_{i-1}, v_i) \in [\![s_i]\!]_{G_{\mathbf{T}}}$. Since i was chosen arbitrarily, this implies that that (5) is a target matching of $\mathcal{T}$ over $G_{\mathbf{T}}$. This concludes the proof that $\mathrm{Rep}_{\Sigma_{\mathbf{T}}}(\pi_{\mathbf{T}}) = \mathrm{Sol}(G_{\mathbf{S}})$.

Next we prove that the process can be done in time $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{T}|)$. To measure $G_{\mathbf{S}}$ we consider the number of edges, that is, the number of triples $(u, a, v) \in G_{\mathbf{S}}$. Notice that $\mathcal{T}$ is an expression over the alphabet $\mathcal{A}_{\mathcal{T}}(\Sigma_1, \Sigma_2)$. Thus, let $k_{\mathcal{T}}$ be the size of the largest expressions of the form (exp_1/exp_2) in $\mathcal{A}_{\mathcal{T}}(\Sigma_1, \Sigma_2)$ that occurs in $\mathcal{T}$. Moreover, let $M_{\mathcal{T}}$ be the size of $\mathcal{T}$ as a regular expression over $\mathcal{A}_{\mathcal{T}}(\Sigma_1, \Sigma_2)$, that is, as just counting the occurrences of symbols and regular-expression operators in $\mathcal{T}$. Then we have that $|\mathcal{T}|$ is $\Theta(k_{\mathcal{T}} \cdot M_{\mathcal{T}})$. Now, in the analysis of the algorithm we have:

- Step 1: we know that given a regular expression, an NFA for the expression can be constructed in time linear in the size of the expression. That is, Step 1 in our algorithm takes time $O(M_{\mathcal{T}})$.
- Step 2: it was shown in [36] that given a NRE *exp* and a graph G computing the set $[\![exp]\!]_G$ can be done in time $O(|G|^2 \cdot |exp|)$ (see Theorem 3.3 in [36][2]). Thus, since $|exp_1|$ is bounded by $k_{\mathcal{T}}$, for every edge $(q_1, (exp_1/exp_2), q_2)$ in $N_{\mathcal{T}}$ we need time $O(|G_{\mathbf{S}}|^2 \cdot k_{\mathcal{T}})$ to construct the edges of the form $((v_1, q_1), (exp_1/exp_2), (v_2, q_2))$ that belong to $G_{\mathbf{S}} \times N_{\mathcal{T}}$. Since this need to be repeated for every edge we obtain a bound of $O(|G_{\mathbf{S}}|^2 \cdot k_{\mathcal{T}} \cdot M_{\mathcal{T}}) = O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{T}|)$. Notice that the size of $G_{\mathbf{S}} \times N_{\mathcal{T}}$ is $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{T}|)$.
- Step 3: this step can be done in linear time in the size of $G_{\mathbf{S}} \times N_{\mathcal{T}}$ by first doing a breath-first search of all the reachable nodes from nodes of the form (q_0, n'), and then doing a breath-first search from all the nodes (q_f, n'') with $q_f \in F$ by following the edges of $G_{\mathbf{S}} \times N_{\mathcal{T}}$ backwards, and then finding all the nodes that can reach a node (q_f, n'') with $q_f \in F$. Both process can be done in time proportional to the size of $G_{\mathbf{S}} \times N_{\mathcal{T}}$ and thus, in time $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{T}|)$.
- Step 4: this step is just a projection of the nodes and edges of $G_{\mathbf{S}} \times N_{\mathcal{T}}$ and thus can be done in time $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{T}|)$.

Thus we obtain that the complete process can be accomplished in time $O(|G|^2 \cdot |\mathcal{T}|)$. This completes the proof of the theorem. $\square$

A.9 Proof of Theorem 5.4

Theorem 4 in [10] proves that there is a CRPQ Q of the form $\exists x \exists y (x, w, y)$, where $w \in \Sigma^*$, such that computing the certain answers over the set $\mathrm{Rep}(\pi)$, for a graph pattern π over Σ, is coNP-complete. It is not difficult to construct, for each pattern π, a mapping $\mathcal{M}$ and a source graph $G_{\mathbf{S}}$ such that π is a universal representative for $G_{\mathbf{S}}$ under $\mathcal{M}$. This shows that computing certain answers in data exchange is as hard as computing certain answers for graph patterns, which suffices for the Theorem. $\square$

A.10 Proof of Theorem 5.6

The algorithm starts by constructing in time $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{T}|)$ a naïve graph pattern π that is a universal representative for $G_{\mathbf{S}}$ under $\mathcal{M}$. Then $(u, v) \in \textsc{certain}_{\mathcal{M}}(nexp, G_{\mathbf{S}})$ if and only if $(u, v) \in \bigcap\{[\![nexp]\!]_G \mid G \in \mathrm{Rep}(\pi)\}$. We show that the latter can be checked in time $O(|\pi| \cdot |nexp|)$, which implies that checking whether $(u, v) \in \textsc{certain}_{\mathcal{M}}(nexp, G_{\mathbf{S}})$ can be solved in time $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{T}| + |\pi| \cdot |nexp|)$, and hence in time $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{T}| \cdot |nexp|)$.

The *naïve* evaluation of a query Q over a naïve pattern π is as follows: Evaluate Q directly over π, treating nulls as if they were constants, and then remove all tuples in the output that contain null values. We claim that $(u, v) \in \bigcap\{[\![nexp]\!]_G \mid G \in \mathrm{Rep}(\pi)\}$ if and only if (u, v) belongs to the naïve evaluation of *nexp* over π. Notice that this immediately implies our desired result that checking whether $(u, v) \in \bigcap\{[\![nexp]\!]_G \mid G \in \mathrm{Rep}(\pi)\}$ can be done in time $O(|\pi| \cdot |nexp|)$. Indeed, from the previous claim we only have to check whether (u, v) belongs to the naïve evaluation of *nexp* over π. But the latter is the same than checking whether $(u, v) \in [\![nexp]\!]_G$, where G is the graph database over $\Sigma_{\mathbf{T}}$ that is obtained from π by replacing each null value with a fresh node id. From Proposition 2.2 the latter can be done in time $O(|G| \cdot |nexp|)$, and hence in time $O(|\pi| \cdot |nexp|)$, Further, G can be constructed in linear time from π, which implies that checking whether $(u, v) \in \bigcap\{[\![nexp]\!]_G \mid G \in \mathrm{Rep}(\pi)\}$ can be done in time $O(|\pi| \cdot |nexp|)$.

Now we prove our claim. First, $(u, v) \in \bigcap\{[\![nexp]\!]_G \mid G \in \mathrm{Rep}(\pi)\}$ implies that (u, v) belongs to the naïve evaluation of *nexp* over G. Indeed, the graph database G over $\Sigma_{\mathbf{T}}$ that is obtained from π by replacing each null value by a fresh node id belongs to $\mathrm{Rep}(\pi)$, and hence $(u, v) \in [\![nexp]\!]_G$. But this is precisely stating that (u, v) belongs to the naïve evaluation of *nexp* over π.

Assume, on the other hand, that (u, v) belongs to the naïve evaluation of *nexp* over π. Take an arbitrary $G \in \mathrm{Rep}(\pi)$. Then there is a homomorphism from π into G that is the identity on u and v. We prove below that NREs are preserved under this kind of homomorphisms, and hence that $(u, v) \in [\![nexp]\!]_G$. We conclude that $(u, v) \in \bigcap\{[\![nexp]\!]_G \mid G \in \mathrm{Rep}(\pi)\}$, since G was arbitrarily chosen.

In order to prove that NREs are preserved under this kind of homomorphisms, we embed NREs into a well-established logic, Datalog, that we know it is preserved under homomorphisms. However, we have to be a little bit careful since Datalog

[2]Actually, in [36], the following stronger property was proved from which the bound $|G|^2 \cdot |exp|$ follows directly. Given a node u, computing the set of all elements v such that $(u, v) \in [\![exp]\!]_G$ can be done in time $O(|G| \cdot |exp|)$ [36].

is not evaluated directly over the class of naïve graph patterns over Σ, but over its standard relational representations given as follows: The vocabulary of these relational representations consists of binary relation symbols E_a, for each $a \in \Sigma$. A naïve graph pattern π over Σ can be interpreted in the standard way as a first-order structure $\mathcal{S}_\pi$ over this vocabulary: The interpretation of symbol E_a in this structure contains all pairs (p,q) of nodes in π such that there is an edge labeled a from p to q in π.

By following a trivial inductive construction, one can show that for each NRE *nexp* over Σ one can construct Datalog program $P(\mathit{nexp})$ such that for each graph pattern π over Σ and pair of nodes (u,v) in π, it is the case that (u,v) belongs to the naïve evaluation of *nexp* over π if and only if (u,v) belongs to the naïve evaluation of $P(\mathit{nexp})$ over S_π. Assuming that Ans, Ans_1, and Ans_2 are special binary symbols that define the output of our Datalog programs, we can easily define this translation as follows:

- If $\mathit{nexp} = \varepsilon$ then $P(\mathit{nexp})$ consists of the single rule $\mathit{Ans}(x,y) \leftarrow x = y$.
- If $\mathit{nexp} = a$ then $P(\mathit{nexp})$ consists of the single rule $\mathit{Ans}(x,y) \leftarrow E_a(x,y)$.
- If $\mathit{nexp} = a^{-1}$ then $P(\mathit{nexp})$ consists of the single rule $\mathit{Ans}(x,y) \leftarrow E_a(y,x)$.
- If $\mathit{nexp} = \mathit{nexp}_1 \cdot \mathit{nexp}2$ then $P(\mathit{nexp})$ consists of the union of the rules of the programs $P(\mathit{nexp}_1)$ and $P(\mathit{nexp}_2)$, together with the rule $\mathit{Ans}(x,y) \leftarrow \mathit{Ans}_1(x,z), \mathit{Ans}_2(z,y)$, where we assume that Ans_1 and Ans_2 are the predicates that define the output of programs $P(\mathit{nexp}_1)$ and $P(\mathit{nexp}_2)$, respectively.
- If $\mathit{nexp} = \mathit{nexp}_1 + \mathit{nexp}2$ then $P(\mathit{nexp})$ consists of the union of the rules of the programs $P(\mathit{nexp}_1)$ and $P(\mathit{nexp}_2)$, together with rules $\mathit{Ans}(x,y) \leftarrow \mathit{Ans}_1(x,y)$ and $\mathit{Ans}(x,y) \leftarrow \mathit{Ans}_2(x,y)$, where we assume that Ans_1 and Ans_2 are the predicates that define the output of programs $P(\mathit{nexp}_1)$ and $P(\mathit{nexp}_2)$, respectively.
- If $\mathit{nexp} = (\mathit{nexp}_1)^*$ then $P(\mathit{nexp})$ consists of the rules of the program $P(\mathit{nexp}_1)$, together with rules $T(x,y) \leftarrow \mathit{Ans}_1(x,y)$, $T(x,y) \leftarrow T(x,z), T(z,y)$, and $\mathit{Ans}(x,y) \leftarrow T(x,y)$, where we assume that Ans_1 is the predicate that defines the output of the program $P(\mathit{nexp}_1)$, and T is an auxiliary and fresh binary symbol that defines the transitive closure of $\mathit{Ans}(x,y)$.
- If $\mathit{nexp} = [\mathit{nexp}_1]$ then $P(\mathit{nexp})$ consists of the union of the rules of the program $P(\mathit{nexp}_1)$, together with a rule $\mathit{Ans}(x,y) \leftarrow \mathit{Ans}_1(x,z), x = y$, where we assume that Ans_1 is the predicate that defines the output of the program $P(\mathit{nexp}_1)$.

It is trivial to prove that this translation satisfies our desired property: For each graph pattern π over Σ and pair of nodes (u,v) in π, it is the case that (u,v) belongs to the naïve evaluation of *nexp* over π if and only if (u,v) belongs to the naïve evaluation of $P(\mathit{nexp})$ over S_π.

Assume that h is a homomorphism from naïve graph pattern $\pi = (N,D)$ into another naïve graph pattern $\pi' = (N',D')$, where both π and π' are over Σ. That is, h maps N into N' in such a way that for each $(p,a,q) \in D$ it is the case that $(h(p),a,h(q)) \in D'$. Assume that the pair (p,q) belongs to the naïve evaluation of *nexp* over π. Then (p,q) also belongs to the naïve evaluation of $P(\mathit{nexp})$ over S_π. It is easy to see that the mapping h also defines a usual relational homomorphism from S_π into $S_{\pi'}$. But Datalog programs are preserved under homomorphisms, which implies that $(h(p),h(q))$ belongs to the naïve evaluation of $P(\mathit{nexp})$ over $S_{\pi'}$. We conclude that $(h(p),h(q))$ belongs to the naïve evaluation of *nexp* over S_π. This shows that NREs are preserved under this kind of homomorphisms, that was to be proved.

Essentially the same proof shows that certain answers of C2RPQs can be evaluated in polynomial time, with respect to data complexity (that is, assuming the query to be fixed), over semi-full RGED-mappings. $\square$

A.11 Proof of Proposition 5.7

We begin by proving the following result:

LEMMA A.3. *There exists a plain and semi-full RGED-mapping $\mathcal{M}_{12}$, and a plain and full RGED-mapping $\mathcal{M}_{23}$ such that the problem of testing if a pair (G_1,G_3) belongs to $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ is* NP*-hard.*

PROOF. This proof is based on a proof for a similar result in the relational case by Fagin et al. [22]. Let $\Sigma_1 = \{e\}$, $\Sigma_2 = \{a,b,c\}$, and $\Sigma_3 = \{d\}$. Consider the RGEDs

$$\begin{aligned}\mathcal{T}_{12} &= (e/(a \cdot b)) + (e/c)\\ \mathcal{T}_{23} &= ((a^- \cdot c \cdot a)/d)\end{aligned}$$

from Σ_1 to Σ_2 and from Σ_2 to Σ_3. Let $\mathcal{M}_{12} = (\Sigma_1, \Sigma_2, \mathcal{T}_{12})$ and $\mathcal{M}_{23} = (\Sigma_2, \Sigma_3, \mathcal{T}_{23})$. We shall reduce the 3-Colorability problem to the problem of checking given a pair of graphs (G_1,G_3), whether $(G_1,G_3) \in \mathcal{M}_{12} \circ \mathcal{M}_{23}$. Consider an undirected graph G and construct a graph database G_1 over Σ_1 such that $(u,e,v) \in G_1$ iff (u,v) is an edge in G. Moreover, let G_3 be the graph composed of the labeled edges $(1,d,2), (2,d,1), (1,d,3), (3,d,1), (2,d,3), (3,d,2)$. We claim that $(G_1,G_3) \in \mathcal{M}_{12} \circ \mathcal{M}_{23}$ iff G is 3-colorable.

($\Rightarrow$) Assume that $(G_1,G_3) \in \mathcal{M}_{12} \circ \mathcal{M}_{23}$. Then there exists a graph database over Σ_2 such that $(G_1,G_2) \in \mathcal{M}_{12}$ and $(G_2,G_3) \in \mathcal{M}_{23}$. Given the semantics of RGEDs, we know that for every $(u,e,v) \in G_1$ we have that $(u,c,v) \in G_2$. Moreover, for every $(u,e,v) \in G_1$ there exists a node w such that $(u,a,w),(w,b,v) \in G_2$. Thus for every node u in G_1

choose an arbitrary node $f(u)$ in G_2 such that $(u, a, f(u)) \in G_2$. Notice that since for every edge $(u, e, v) \in G_1$ we have that $(u, a, f(u)), (u, c, v), (v, a, f(v)) \in G_2$. Thus we have that

$$f(u)((a^- \cdot c \cdot a)/d)f(v)$$

is a source matching of $\mathcal{T}_{23}$ over G_2, which since $(G_2, G_3) \in \mathcal{M}_{23}$ implies that $f(u)((a^- \cdot c \cdot a)/d)f(v)$ is a target matching, and then $(f(u), d, f(v)) \in G_3$. Given the construction of G_3, we have that f is a function from the nodes in G to the set $\{1, 2, 3\}$ such that for every edge (u, v) in G it holds that $f(u) \neq f(v)$, and thus f is a 3-coloring of G.

($\Leftarrow$) Let f be a function from the nodes of G to $\{1, 2, 3\}$ such that if (u, v) is an edge in G, then $f(u) \neq f(v)$. Let G_2 be the following instance of Σ_2. For every $(u, e, v) \in G_1$ we have that $(u, c, v), (u, a, f(u)), (f(u), b, v) \in G_2$. Notice that since G is undirected, if $(u, e, v) \in G_1$ then $(v, e, u) \in G_1$ and thus, $(v, c, u), (v, a, f(v)), (f(v), b, u) \in G_2$. G_2 does not contain any additional edge. It is clear that $(G_1, G_2) \in \mathcal{M}_{12}$. We prove now that $(G_2, G_3) \in \mathcal{M}_{23}$. Thus, let

$$x((a^- \cdot c \cdot a)/d)y$$

be a source matching of $\mathcal{T}_{23}$ over G_2. Thus we have that $(u, a, x), (u, c, v), (v, a, y) \in G_2$. By the construction of G_2 we have that (u, v) is an edge in G and $x = f(u)$ and $y = f(v)$, which implies that $x, y \in \{1, 2, 3\}$ and $x \neq y$, and thus $x((a^- \cdot c \cdot a)/d)y$ is also a target matching of $\mathcal{T}_{23}$ over G_3. Thus we have that every source matching of $\mathcal{T}_{23}$ over G_2 is also a target matching of $\mathcal{T}_{23}$ over G_3 which implies that $(G_2, G_3) \in \mathcal{M}_{23}$ and thus $(G_1, G_3) \in \mathcal{M}_{12} \circ \mathcal{M}_{23}$. This completes the proof. □

Next we prove the Proposition. The classical result by A. Dawar (see "A restricted second order logic for finite structures. Inf. Comput. 1998. 143, 2, 154-174.") states that 3-colorability cannot be expressed in the infinitary logic $\mathcal{L}^{\omega}_{\infty\omega}$. We can deduce from the proof of the above Lemma that if the composition of two semi-full RGEDs can be expressed in $\mathcal{L}^{\omega}_{\infty\omega}$, then 3-colorability can also be expressed in $\mathcal{L}^{\omega}_{\infty\omega}$. Just notice that the reduction in the above Lemma belongs to the class of quantifier free reductions (see "N. Immerman, Descriptive Complexity. Springer, 1999" for a precise definition of quantifier free reductions). Thus the proof follows from the following lemma that states that every RGED can be expressed in $\mathcal{L}^{\omega}_{\infty\omega}$, thus implying that the composition of the two RGEDs in the above Lemma is not expressible as a RGED.

LEMMA A.4. *Let $\mathcal{T}$ be a RGED from Σ_1 to Σ_2. Then there is a formula Φ in $\mathcal{L}^{\omega}_{\infty\omega}$ such that, for all graph databases G_1 and G_2 over Σ_1 and Σ_2, respectively, we have that $(G_1, G_2) \models \mathcal{T}$ iff $(G_1, G_2) \models \Phi$.*

PROOF. Given an NRE *exp* let $\alpha_{exp}(x, y)$ denote a formula in $\mathcal{L}^{\omega}_{\infty\omega}$ that is equivalent to *exp*. We know that this formula exists as in Theorem 5.6 we have shown how to construct a Datalog program which is equivalent to every NRE (and Datalog is contained in $\mathcal{L}^{\omega}_{\infty\omega}$). We use an inductive argument to show that every RGED $\mathcal{T}$ can be represented as a formula in $\mathcal{L}^{\omega}_{\infty\omega}$. For every RGED $\mathcal{T}$ we construct two binary formulas $\text{source}_{\mathcal{T}}(x, y)$ and $\text{check}_{\mathcal{T}}(x, y)$. Before showing the construction we make an important observation about RGEDs. Notice that the empty string in RGEDs has no effect, as in the definition of source and target matchings we only consider strings with at least one symbol. Thus, w.l.o.g. we can assume that regular expressions defining RGEDs does not use the empty string ε and does not use $(\cdot)^*$ but only $(\cdot)^+$. Now we construct formulas $\text{source}_{\mathcal{T}}(x, y)$ and $\text{check}_{\mathcal{T}}(x, y)$ inductively as follows:

- if $\mathcal{T} = (e/s)$ we have that

$$\begin{aligned} \text{source}_{\mathcal{T}}(x, y) &: \alpha_e(x, y) \\ \text{check}_{\mathcal{T}}(x, y) &: \alpha_e(x, y) \rightarrow \alpha_s(x, y) \end{aligned}$$

- if $\mathcal{T} = \mathcal{T}_1 \cdot \mathcal{T}_2$ we have that

$$\begin{aligned} \text{source}_{\mathcal{T}}(x, y) &: \exists u\big(\ \exists y(\text{source}_{\mathcal{T}_1}(x, y) \wedge y = u) \wedge \exists x(u = x \wedge \text{source}_{\mathcal{T}_2}(x, y))\ \big) \\ \text{check}_{\mathcal{T}}(x, y) &: \forall u\big(\exists y(\text{source}_{\mathcal{T}_1}(x, y) \wedge y = u) \wedge \exists x(u = x \wedge \text{source}_{\mathcal{T}_2}(x, y)) \rightarrow \\ &\qquad \exists y(\text{check}_{\mathcal{T}_1}(x, y) \wedge y = u) \wedge \exists x(u = x \wedge \text{check}_{\mathcal{T}_2}(x, y))\ \big) \end{aligned}$$

- if $\mathcal{T} = \mathcal{T}_1 + \mathcal{T}_2$ we have that

$$\begin{aligned} \text{source}_{\mathcal{T}}(x, y) &: \text{source}_{\mathcal{T}_1}(x, y) \vee \text{source}_{\mathcal{T}_2}(x, y) \\ \text{check}_{\mathcal{T}}(x, y) &: \text{check}_{\mathcal{T}_1}(x, y) \wedge \text{check}_{\mathcal{T}_2}(x, y) \end{aligned}$$

- if $\mathcal{T} = (\mathcal{T}_1)^+$ we have that

$$\begin{aligned} \text{source}_{\mathcal{T}}(x, y) &: \bigvee_{k=1}^{\infty} \text{source}_{(\mathcal{T}_1)^k}(x, y) \\ \text{check}_{\mathcal{T}}(x, y) &: \bigwedge_{k=1}^{\infty} \text{check}_{(\mathcal{T}_1)^k}(x, y) \end{aligned}$$

  where $(\mathcal{T}_1)^1 = \mathcal{T}_1$ and $(\mathcal{T}_1)^{k+1} = (\mathcal{T}_1)^k \cdot \mathcal{T}_1$.

With the above construction we can finally define a formula Φ in $\mathcal{L}^{\omega}_{\infty\omega}$ for a RGED $\mathcal{T}$ given by $\forall x \forall y\, \text{check}_{\mathcal{T}}(x, y)$. Clearly Φ is in $\mathcal{L}^{\omega}_{\infty\omega}$ as it only uses variables x, y and u (plus a finite number of variables used in every formula defining a nested regular expression in the base case).

We now prove by induction that $(G_1, G_2) \models \Phi$ if and only if $(G_1, G_2) \models \mathcal{T}$. In fact, we prove something stronger. We prove that the following two properties hold:

1. There is a target matching of $\mathcal{T}$ over G_1 that begins in node n_1 and ends in node n_2 if and only if $G_1 \models \text{source}_{\mathcal{T}}(n_1, n_2)$.
2. Every source matching of $\mathcal{T}$ over G_1 that begins in node n_1 and ends in node n_2 is also a target matching of $\mathcal{T}$ over G_2, if and only if, $(G_1, G_2) \models \text{check}_{\mathcal{T}}(n_1, n_2)$.

From this, it clearly holds that $(G_1, G_2) \models \mathcal{T}$ if and only if $(G_1, G_2) \models \Phi$, since Φ is defined as $\forall x \forall y\, \text{check}_{\mathcal{T}}(x, y)$. We proceed by induction on the construction of RGEDs.

- $\mathcal{T} = (e/s)$: It is clear that $n_1(e/s)n_2$ is a source matching if and only if $G_1 \models \alpha_e(n_1, n_2) = \text{source}_{\mathcal{T}}(n_1, n_2)$. It is also clear that every source mathcing $n_1(e/s)n_2$ is also a target matching, if and only if $(G_1, G_2) \models \alpha_e(n_1, n_2) \rightarrow \alpha_s(n_1, n_2)$ which occurs if and only if $(G_1, G_2) \models \text{check}_{\mathcal{T}}(n_1, n_2)$.
- $\mathcal{T} = \mathcal{T}_1 \cdot \mathcal{T}_2$: Assume that there exists a source matching of $\mathcal{T}$ over G_1 that stars in n_1 and ends in n_2. Then clearly there exists a node n and a source matching of $\mathcal{T}_1$ over G_1 that starts in n_1 and ends in n, and a source matching of $\mathcal{T}_2$ over G_1 that starts in n and ends in n_2. This implies by induction hypothesis that $G_1 \models \text{source}_{\mathcal{T}_1}(n_1, n)$ and $G_1 \models \text{source}_{\mathcal{T}_2}(n, n_2)$ and thus $G_1 \models \text{source}_{\mathcal{T}}(n_1, n_2)$. The other direction is similar.

  Now assume that every source matching of $\mathcal{T}$ over G_1 that starts in n_1 and ends in n_2 is also a target matching of $\mathcal{T}$ over G_2. We need to prove that $(G_1, G_2) \models \text{check}_{\mathcal{T}}(n_1, n_2)$. Then assume that n is an arbitrary node such that $G_1 \models \text{source}_{\mathcal{T}_1}(n_1, n)$ and $G_1 \models \text{source}_{\mathcal{T}_2}(n, n_2)$. By the construction of $\text{check}_{\mathcal{T}}(x, y)$, we need to prove that $(G_1, G_2) \models \text{check}_{\mathcal{T}_1}(n_1, n)$ and $(G_1, G_2) \models \text{check}_{\mathcal{T}_2}(n, n_2)$. Now, since $G_1 \models \text{source}_{\mathcal{T}_1}(n_1, n)$ and $G_1 \models \text{source}_{\mathcal{T}_2}(n, n_2)$, by using the induction hypothesis, we know that there exists a source matching of $\mathcal{T}_1$ that starts in n_1 and ends in n, and a source matching of $\mathcal{T}_2$ that starts in n and ends in n_2. Notice that the concatenation of this two source matchings generates a source matching for $\mathcal{T}$ over G_1, and thus, this source matching is also a target matching over G_2 (recall that we are assuming that every source matching of $\mathcal{T}$ over G_1 that starts in n_1 and ends in n_2 is also a target matching of $\mathcal{T}$ over G_2). Moreover, every possible source matching of $\mathcal{T}_1$ that starts in n_1 and ends in n should also be a target matching of $\mathcal{T}_1$ over G_2, since otherwise, there would exists a source matching of $\mathcal{T}$ over G_1 that is not a target matching of $\mathcal{T}$ over G_2. Symmetrically, we can argue that every source matching of $\mathcal{T}_2$ over G_1 that starts in n and ends in n_2 is also a target matching of $\mathcal{T}_2$ over G_2. Thus, by using the induction hypothesis, we have that $(G_1, G_2) \models \text{check}_{\mathcal{T}_1}(n_1, n)$ and $(G_1, G_2) \models \text{check}_{\mathcal{T}_2}(n, n_2)$, and thus $(G_1, G_2) \models \text{check}_{\mathcal{T}}(n_1, n_2)$.

  Similarly, we can prove that $(G_1, G_2) \models \text{check}_{\mathcal{T}}(n_1, n_2)$ implies that every source matching of $\mathcal{T}$ over G_1 that starts in n_1 and ends in n_2 is also a target matching of $\mathcal{T}$ over G_2.
- $\mathcal{T} = \mathcal{T}_1 + \mathcal{T}_2$: Assume that there exists a source matching of $\mathcal{T}$ over G_1 from n_1 to n_2. If the source matching is a source matching of $\mathcal{T}_1$ over G_1 then by induction hypothesis we know that $G_1 \models \text{source}_{\mathcal{T}_1}(n_1, n_2)$ and thus we have $G_1 \models \text{source}_{\mathcal{T}}(n_1, n_2)$. The case in which the source matching is a source matching of $\mathcal{T}_2$ is similar. Now assume that $G_1 \models \text{source}_{\mathcal{T}}(n_1, n_2)$ then clearly $G_1 \models \text{source}_{\mathcal{T}_1}(n_1, n_2)$ or $G_1 \models \text{source}_{\mathcal{T}_2}(n_1, n_2)$ and by induction hypothesis the property holds.

  Now assume that every source matching of $\mathcal{T}$ over G_1 that starts in n_1 and ends in n_2 is also a target matching of $\mathcal{T}$ over G_2. We need to prove that $(G_1, G_2) \models \text{check}_{\mathcal{T}}(n_1, n_2)$. By the construction of $\text{check}_{\mathcal{T}}(x, y)$, we need to prove that $(G_1, G_2) \models \text{check}_{\mathcal{T}_1}(n_1, n_2)$ and $(G_1, G_2) \models \text{check}_{\mathcal{T}_2}(n_1, n_2)$. Thus consider an arbitrary source matching of $\mathcal{T}_1$ over G_1 from n_1 to n_2, then this source matching is also a source matching of $\mathcal{T}$ over G_1 and thus it should be a target matching of $\mathcal{T}$ over G_2. It is easy to see that this implies that it is also a target matching of $\mathcal{T}_1$ over G_2 and thus by induction hypothesis, we have $(G_1, G_2) \models \text{check}_{\mathcal{T}_1}(n_1, n_2)$. The case for a source matching of $\mathcal{T}_2$ over G_1 is similar. For the other direction we can use a similar argument.
- $\mathcal{T} = (\mathcal{T}_1)^+$: this case follows from the case of the concatenation and using the definition of $\text{source}_{\mathcal{T}}(x, y)$ and $\text{check}_{\mathcal{T}}(x, y)$. Just notice that there is a source matching of $\mathcal{T}$ over G_1 from n_1 to n_2 if and only if there is either a source matching of $\mathcal{T}_1$ or of $\mathcal{T}_1 \cdot \mathcal{T}_1$ or $\mathcal{T}_1 \cdot \mathcal{T}_1 \cdot \mathcal{T}_1$, etc. Similarly, we have that every source matching of $\mathcal{T}$ over G_1 from n_1 to n_2 is also a target matching of $\mathcal{T}$ over G_2 if and only if, for every k it holds that every source matching of $(\mathcal{T}_1)^k$ over G_1 from n_1 to n_2 is also a target matching of $(\mathcal{T}_1)^k$ over G_2.

This completes the proof of the lemma. $\square$