Conjunctive Regular Path Queries with Capture Groups[*]

MARKUS L. SCHMID, Humboldt-Universität zu Berlin, Germany

In practice, regular expressions are usually extended by so-called capture groups or capture variables, which allow to capture a subexpression by a variable that can be referenced in the regular expression in order to describe repetitions of subwords. We investigate how this concept could be used for pattern-based graph querying, i. e., we investigate conjunctive regular path queries (CRPQs) that are extended by capture variables.

If capture variables are added to CRPQs in a completely unrestricted way, then Boolean evaluation becomes PSPACE-hard in data complexity, even for single-edge graph patterns. On the other hand, if capture variables do not occur under a Kleene star, then the data complexity drops to NL-completeness. Combined complexity is in EXPSPACE, but drops to PSPACE-completeness if the depth (i. e., the nesting depth of capture variables) is bounded, and it drops to NP-completeness if the size of the images of capture variables is bounded by a constant (regardless of the depth or of whether capture variables occur under a Kleene star).

In the application of regular expressions as string searching tools, references to capture variables only describe exact repetitions of subwords (i. e., they implement the equality relation on strings). Following recent developments in graph database research, we also study CRPQs with capture variables that describe arbitrary regular relations. We show that if the expressions have depth 0, or if the the size of the images of capture variables is bounded by a constant, then we can allow arbitrary regular relations, while staying in the same complexity bounds. We also investigate the problems of checking whether a given tuple is in the solution set, and computing the whole solution set.

On the conceptual side, we add capture variables to CRPQs in such a way that they can be defined in an expression on one arc of the graph pattern, but also referenced in expressions on other arcs. Hence, they add to CRPQs the possibility to define inter-dependencies between different paths, which is a relevant feature of pattern-based graph querying.

CCS Concepts: • **Theory of computation** → *Formal languages and automata theory*; **Database query languages (principles)**; Data structures and algorithms for data management.

Additional Key Words and Phrases: Graph Databases; Conjunctive Regular Path Queries, Regular Expressions with Backreferences

1 INTRODUCTION

The popularity of graph databases (commonly abstracted as directed, edge-labelled graphs) is due to their applicability in a variety of settings where the underlying data is naturally represented as graphs, e. g., Semantic Web and social networks, biological data, chemical structure analysis, pattern recognition, network traffic, crime detection, object oriented data. The problem of querying graph-structured data has been studied over the last

[*]This work represents a substantially extended version of the paper "Conjunctive Regular Path Queries with String Variables" [49] (see Section 1.5 for a discussion of the differences of these two versions).

The author is supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) – project number 416776735 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 416776735).

Author's address: Markus L. Schmid, MLSchmid@MLSchmid.de, Humboldt-Universität zu Berlin, Unter den Linden 6, D-10099, Berlin, Germany.

0362-5915/2022/2-ART $15.00
https://doi.org/10.1145/3514230

three decades and still receives a lot of attention. For more background information, we refer to the introductions of the recent papers [7, 8, 22, 39], and to the survey papers [1, 2, 5, 55].

Many query languages for graph databases (for practical systems as well as those studied in academia) follow an elegant and natural declarative approach: a query is described by a *graph pattern*, i. e., a graph $G = (V, E)$ with edge labels that represent some path-specifications. The evaluation of such a query consists in *matching* it to the graph database $\mathcal{D} = (V_{\mathcal{D}}, E_{\mathcal{D}})$, i. e., finding a mapping $h : V \to V_{\mathcal{D}}$, such that, for every $(x, s, y) \in E$, in $\mathcal{D}$ there is a path from $h(x)$ to $h(y)$ whose edge labels satisfy the path-specification s. In the literature, such query languages are also called *pattern-based*.

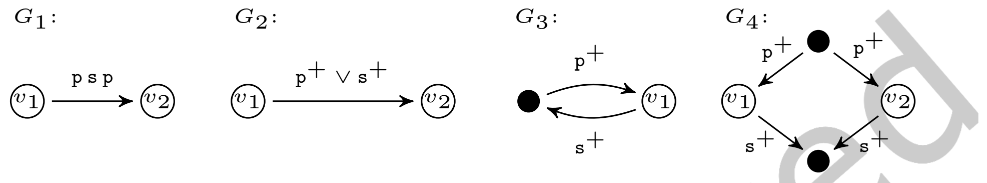


Fig. 1. Examples of RPQ and CRPQ.

The most simple graph-patterns (called *basic* in [1]) have just fixed relations (i. e., edge-labels) from the graph database as their edge labels. In order to implement *navigational features* that can describe more complex connectivities between nodes via longer paths instead of only single arcs, it is common to represent path specifications in graph patterns by regular expressions (over the set of possible edge labels). Navigational features are popular, since they allow to query the *topology* of the data and, if transitivity can be described, exceed the power of the basic relational query languages. The *regular path queries* (RPQs) given by *single-edge* graph patterns $(\{x, y\}, \{(x, s, y)\})$, where s is a regular expression, can be considered the simplest navigational graph patterns. General graph patterns labelled by regular expressions are called *conjunctive regular path queries* (CRPQs).

Since RPQs and CRPQs are central for our work, we discuss some examples. Consider a graph database with nodes representing persons, arcs (u, p, v) meaning "u is a (biological) parent of v" and arcs (u, s, v) meaning "v is u's PhD-supervisor". We consider the graph patterns from Figure 1 (labelled nodes are considered as free variables of the query). Then G_1 describes pairs (v_1, v_2), where v_1's child has been supervised by v_2's parent; G_2 describes pairs (v_1, v_2), where v_1 is a biological ancestor or an academical descendant of v_2; G_3 describes v_1 that have a biological ancestor that is also their academical ancestor; G_4 describes pairs (v_1, v_2), where v_1 and v_2 are biologically related as well as academically. Note that G_1, G_2 represent RPQs, while G_3, G_4 represent CRPQs.

The simple, yet relevant classes of RPQs and CRPQs have been heavily studied in database theory: results on RPQs [3, 12, 19, 21, 40, 41, 44, 45], conjunctive RPQs [4, 10, 11, 23, 46] and extensions thereof [7, 33, 39, 39], questions of static analysis [6, 15, 26–28, 35, 47], experimental analyses [13, 14, 43], and surveys of this research area [1, 5, 16, 55].

Despite their long-standing investigation, these basic classes still pose several challenges that are currently studied. For example, [40–42] provide an in-depth analysis of the complexity of RPQs for different path semantics. So far in this introduction, we implicitly assumed *arbitrary path* semantics, but since there are potentially infinitely many such paths, query languages that also retrieve paths often restrict this by considering simple paths or trails. However, such semantics make the evaluation of RPQs much more difficult (see [40–42] for details). Much effort has also been spent on extending RPQs and CRPQs to the setting where the data-elements stored at nodes of the graph database can also be queried (see [38, 39]). In [8], the authors represent partially defined graph data by graph patterns and then query them with CRPQs (among others). In the recent paper [6], the authors study the boundedness problem for unions of CRPQs (i. e., the problem of finding an equivalent union of (relational) conjunctive queries), and [19] investigates the fine-grained complexity of RPQs.

Also in the practical world, pattern-based query languages for graph databases play a central role. Most prominently, the *W3C Recommendation* for *SPARQL 1.1* "is based around graph pattern matching" (as stated in Section 5 of [36]), and *Neo4J Cypher* also uses graph patterns as a core functionality (see [52]). Moreover, the graph computing framework *Apache TinkerPop*^{TM} contains the graph database query language *Gremlin* [54], which is more based on the navigational graph traversal aspect, but nevertheless supports pattern-based query mechanisms. Note that [1] surveys the main features of these three languages.

1.1 Regular Expressions for Strings and Graphs

The investigation of regular expressions for graph querying is still reasonably new, and the current state-of-the-art of "regular expression based graph querying" comprises a (constantly growing) number of theoretical papers that present results about decidability, upper and lower complexity bounds, expressive power etc., and, on the practical side, academical and industrial implementations of graph database systems (see our pointers from the literature mentioned above). In stark contrast to that, the use of regular expressions as a tool for string searching or string analysis dates back to the beginnings of computer science (milestones are Kleene's introduction in 1956 [37], and Thompson's first implementation in 1968 [53]). Therefore, the classical theoretical results about regular expressions (and regular languages in general) are part of undergraduate courses and textbooks, and, on the practical side, regular expressions are implemented in almost all modern programming languages, and their practical use is a standard expertise of programmers and applied computer scientists (this is also demonstrated by the fact that many textbooks and training courses exist that are particularly tailored to teaching the use of regular expressions in practical scenarios (a standard textbook in this regards is [34])).

This development has also caused a deviation of practical regular expressions from their theoretical origins. More precisely, practical implementations of regular expressions usually cover the theoretical core, but also use many extensions, some of which are just syntactic sugar, while others properly increase expressive power and complexity. Hence, it is nowadays common in academia to distinguish between *theoretical* regular expressions (dating back to Kleene [37]) and *practical* regular expressions (sometimes also called *real* regular expressions) to denote the different varieties that are practically used. In particular, it is an ongoing research effort in formal language theory – arguably initiated by [17] – to catch up with the practically motivated modifications of regular expressions and provide theoretical foundations for them (see, e. g., [17, 18, 29, 32, 48]). In fact, regular expressions form a research area for which the practical and the theoretical work seems to be mutually stimulating.

A famous practical extension of regular expressions, that is central to this work, are *capture variables* (also called capture groups, backreferences, or just variables), and they are (in some way) implemented in most modern regular expressions languages (see [34]).[1] From an intuitive point of view, the concept is quite simple: by a *variable definition* $\mathsf{x}\{r\}$, we can allocate a subexpression r to a *variable* x (or *capture* r by the variable x), and then, by using a *reference* x in the expression, we can specify an exact copy of whatever is matched to the subexpression r captured by x. For example, $\mathsf{x}\{(\mathsf{a} \vee \mathsf{b})^*\}\mathsf{c}\mathsf{x}$ describes the (non-regular) language $\{w\mathsf{c}w \mid w \in \{\mathsf{a}, \mathsf{b}\}^*\}$. Capture variables are not mere syntactic sugar. They significantly increase the expressive power and matching complexity of regular expressions. The theoretical properties of regular expressions with capture variables are well-documented in the literature and we do not discuss them here; the interested reader is referred to the introductions of [29, 32]. In particular, Section 8.2 of [32] gives a comprehensive discussions of different variants of regular expressions with capture variables in theory and practice.

[1]The terms *capture groups* and *backreferences* seem to be the most common in the applied literature. We nevertheless use the term *capture variables*, since it is more convenient to just talk about *variables* and then distinguish between their *definitions* and their *references*.

1.2 Capture Variables for Graph Querying

In database theory, especially in the context of graph databases and regular path queries, regular expressions are usually understood as the clean and simple theoretical concept that was the basis for practical implementations back in 1968 (e. g., the one by Thompson [53]). This is understandable, since theoretical regular expressions are suitable for theoretical investigations (while extensions like capture variables can get quite messy in this regard). However, in the case that regular expressions will become, to some extent, a standard tool for graph querying, we should also keep in mind that for most programmers and IT-administrators the term "regular expressions" has a different meaning than what is usually investigated in the academic literature.

Obviously, it should not be our main objective to add to CRPQs as many concepts from practical regular expressions as possible, but the question which of the practically existing features of regular expression can also be added to CRPQs naturally arises for most people who are familiar with the practical side of regular expressions. If a practitioner, who is familiar with regular expressions, is provided with a graph query language that uses regular expressions (in the theoretical sense), then her first question would probably be why the regular expressions are so severely restricted in comparison to what she knows from practice and what she is trained for. Especially from a theoretical foundations perspective, we should be able to provide some guidance with respect to that question, and it would be good to have theoretical results that suggest that a certain practical extension of regular expressions either probably can, or should not be added to CRPQs. In fact, this is the most valuable support for practical developments that can be provided by theoretical research (as an example of such a situation in the area of graph databases, consider how theoretical investigations led to an understanding of which path semantics may cause problems if practically implemented [40–42]).

This work is dedicated to a theoretical investigation of the following

Main Research Questions:

(1) How can we add capture variables (as found in practical regular expressions) to CRPQs?
(2) To what extent can CRPQs with capture variables be used for graph querying?

The first question has to be answered by a conceptual contribution, i. e., sound definitions of possible query classes, while the second question should be answered by providing theoretical results like upper and lower complexity bounds for their (Boolean) evaluation problem.

With respect to the first question, it is a crucial point that we do not merely replace the regular expressions of CRPQs by a more powerful class of language descriptors, namely regular expressions with capture variables. Instead, we add capture variables in such a way that they can be defined in the expression of one arc and referenced in the expression of another arc. While such a feature is irrelevant in the string case, it is interesting for graph querying, since it provides a natural means of defining inter-dependencies between paths. The graph patterns of Figure 2 serve as first examples of how CRPQs with capture variables may look like (also recall the intuitive explanation of the semantics of capture variables given in Section 1.1). For example, G_1 describes triangles (v_1, v_2, v_3), where v_1 is connected to v_2 by a path labelled with aa or b, v_2 is connected to v_3 by a path labelled with a sequence of arbitrary relations except a or b, and v_3 is connected to v_1 by a path that is labelled *in the same way* as the v_1-to-v_2, or the v_2-to-v_3 path. We will discuss more practically motivated examples in Section 1.4.

With respect to the second question, our insights can be summarised as done in the following Section 1.3 (see also Table 1).

1.3 Main Results

Our main results are as follows:

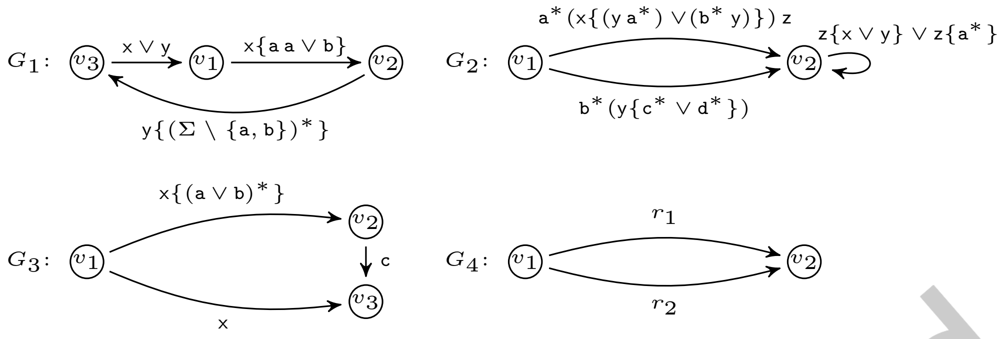


Fig. 2. Examples of CRPQs with capture variables.

- If regular expressions with capture variables are used in CRPQs completely unrestricted, then even single-edge queries (i. e., the graph pattern consists of just two nodes connected by an arc) have a prohibitively high complexity: Boolean evaluation is PSpace-hard in data complexity.
- Restricting the regular expressions such that neither variable definitions nor references are subject to a Kleene star lets the data complexity drop from PSpace-hardness to nondeterministic logarithmic space completeness (NL-completeness).
- Based on the previous observation, we define a class CXRPQ of CRPQs with capture variables. This conceptual contribution is not straightforward, since we also allow that references for the same variable can be distributed over different arc labels of the graph pattern, which allows to describe *inter-path dependencies* (dependencies between labels of different paths). To achieve this, we define a conjunctive variant of regular expressions with capture variables.
- Boolean evaluation for CXRPQs is NL-complete in data complexity, but we can only show an upper bound of ExpSpace for combined complexity. Therefore, we consider further restrictions:
  - If the nesting depth of variable definitions is bounded by a constant, then the combined complexity drops to PSpace-completeness (note that this is precisely the complexity achieved by the extended conjunctive regular path queries from [7]).
  - If the capture variables are restricted such that their images have a constant size bound, then the combined complexity drops to NP-completeness (note that this is precisely the complexity achieved by CRPQs); and this even holds if we allow capture variables to occur under Kleene stars.

In addition to these main results, we also present the following additional results, which mainly follow from variations of the proofs used for the main results (note that these results might nevertheless be practically relevant):

- If the number of variables is bounded by a constant, then Boolean evaluation is PSpace-complete in combined complexity.
- If the variables are restricted such that their images have a size logarithmically bounded in the database size (instead of bounded by a constant), then Boolean evaluation is still NP-complete in combined complexity and can be done in nondeterministic poly-logarithmic space in data complexity.
- If all variable definitions have the form x{r}, where r is a classical regular expression (i. e., r does not contain further variable definitions or references), then we can also replace the string-equality relation represented by variables with arbitrary regular relations (e. g., prefix relation, or equal length relation), and still Boolean evaluation is PSpace-complete in combined complexity and NL-complete in data complexity.

- For CXRPQs with bounded image size, we can use arbitrary regular relations without any further restriction and still Boolean evaluation is NP-complete in combined complexity and NL-complete in data complexity.
- For the full class of CXRPQ, checking tuples (instead of only Boolean evaluation) can also be done within the same complexity bounds, i. e., ExpSpace in combined complexity and NL-complete in data complexity.
- For the full class of CXRPQ, the whole solution set can be computed in nondeterministic logarithmic space[2] in data complexity.
- The expressive power of our CXRPQ fragments is compared to CRPQ and ECRPQ (i. e., the extended conjunctive regular path queries from [7]).

These results provide a comprehensive, although not exhaustive, answer to our main research questions formulated in Section 1.2; open questions are discussed in Section 11.

1.4 Possible Practical Implications

As made clear above, this paper focuses on theoretical results; in particular, we do not try to introduce a specific query class that is meant for practical implementations. Instead, we wish to provide theoretical foundations that explain how capture variables might be added to CRPQs, and what we can expect from such an extension in terms of complexity and expressive power. If, however, an actual query class based on CRPQs with capture variables (or features similar to capture variables) is to be introduced, we believe that our theoretical results, especially with respect to the considered restrictions and fragments, could give some valuable guidance.

Despite our focus on theoretical investigations, we shall now discuss several practical scenarios in which CRPQs with capture variables might be useful. Our examples will demonstrate that there are non-artificial queries that can conveniently be realised with the help of capture variables, that have evaluation complexity comparable to classical CRPQs, and that are not obviously covered by other existing query classes.

We first observe that using capture variables in CRPQs enhances the model in a twofold way: (1) we can define more complex (and non-regular) reachability relations of paths, (2) by defining a variable on one arc of the graph pattern and using references to it on other arcs, we can describe *inter-path dependencies*.

The first point is somewhat generic, since we could always replace regular expressions in CRPQs by some other language description formalism (e. g., context-free languages), and obtain a new class of conjunctive path queries (the question here is whether the actual class of language descriptors make sense in the context of graph database querying). Capture variables in regular expressions are particularly tailored to repetitions of subwords (a feature that is somewhat at odds with the classical language classes of the Chomsky hierarchy (e. g., the simple *copy language* $\{ww \mid w \in \Sigma^*\}$ is not even context-free), but can be found in pattern matching and string searching tasks, and is central to combinatorics on words).

The second point is more interesting, since describing inter-dependencies between paths (other than that they start or end in the same node) goes beyond CRPQ's expressive power. In fact, that is the main motivation for [7] to define extended conjunctive path queries (ECRPQs) by adding regular relations to CRPQs. As it seems, capture variables are an alternative formalism (that is well-established in applied computer science) that naturally yields the power to define some inter-path dependencies. Note that the fact that the possibility to define a variable on one arc of the graph pattern and using references to it on other arcs, requires some more conceptual work than just replacing regular expressions of CRPQs by another language description formalism.

We have made our point in Sections 1.1 and 1.2 that capture variables, being such a widely-used extension of regular expressions, are a natural candidate for an extension to CRPQs, and we have summarised in Section 1.3 our theoretical results that show us what to expect from it complexity-wise. Let us now discuss some practical scenarios in which capture variables might extend CRPQs in a useful way.

[2]More precisely, the function that maps a graph database and a CXRPQ to the solution set is a nondeterministic log-space computable function.

1.4.1 Inter-Path Dependencies. As mentioned above, the need for describing relationships between the arcs of the graph pattern has led to the model of extended conjunctive regular path queries (ECRPQs) of [7]. In Sections 1 and 4 of [7], the authors state several application scenarios, where describing relations between paths is necessary.

A main difference is that ECRPQs allow arbitrary regular relations, while capture variables and references only describe the equality relation. Nevertheless, as mentioned in Section 1.3, if the CXRPQs are simple enough, capture variables can also be interpreted as describing arbitrary regular relations. Hence, just like ECRPQs, CXRPQs might also be interesting for some of the applications mentioned in [7]: finding semantic web associations, pattern matching, approximate matching and sequence alignment.

1.4.2 Simple Copies of Paths. A rather simple way of using capture variables is to capture a whole path in a variable x and then use the variable reference x as some other path label. For example, we might want to find node triples (v_1, v_2, v_3) such that v_3 is a c-successor of v_2, and both v_2 and v_3 can be reached from v_1 by paths labelled with the *same* sequence of a or b relations (see graph pattern G_3 of Figure 2). Note that we could also interpret variable x as describing some regular relation different from the equality relation.

Such queries can be formulated as ECRPQs as well. However, if we use the references not as single arc labels, but inside of more complicated regular expressions, the situation gets more complicated (this is demonstrated by the (artificial) graph patterns G_1 and G_2 from Figure 2). In the following, we shall discuss some more practical examples.

1.4.3 Ordered Execution of Jobs. Assume that we have a graph database that represents a complex state-transition-system (i. e., the states are represented by nodes and the arcs are labelled by transition names from $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$). Now we want to query pairs (u, v) of states, such that v can be reached from u by a sequence of at most 8 state transitions, but there should also be an alternative path that performs the same transitions, but in an ordered way, i. e., first all transition of a certain type are performed, then all transitions of another type and so on. For example, if u can be transformed into v by transition sequence abacab, then suitable ordered alternatives would be sequences aaacbb, or caaabb.

As a more concrete example of such a scenario, we may think of a manufacturing process, where the nodes describe the possible states of a component to be manufactured and arc labels represent manufacturing steps; or the nodes describe the possible internal states of a machine performing some task and arc labels are possible instructions. In such settings, the query described above might be relevant for optimisation – performing all jobs of the same kind before moving on to a different job might allow a more economical use of resources.

The query could be realised by the graph pattern G_4 of Figure 2, where the expressions r_1 (for the transition sequence of length at most 8) and r_2 (for the ordered alternative path) are defined by

$$r_1 = \mathsf{x_a}\{\mathsf{x_b}\{\mathsf{x_c}\{\Sigma^{\leq 8}\}\}\}\,,$$
$$r_2 = \mathsf{x_a x_b x_c} \vee \mathsf{x_a x_c x_b} \vee \mathsf{x_b x_a x_c} \vee \mathsf{x_b x_c x_a} \vee \mathsf{x_c x_a x_b} \vee \mathsf{x_c x_b x_a}\,,$$

but instead of using the string-equality relation for the capture variable $\mathsf{x_a}$, we interpret references $\mathsf{x_a}$ to describe just the subsequence of occurrences of a of whatever is captured by the definition of $\mathsf{x_a}$ (instead of an exact copy). For example, r_1 would match abacab, and the whole string abacab would be captured by $\mathsf{x_a}$, $\mathsf{x_b}$ and $\mathsf{x_c}$, but then in r_2 the reference $\mathsf{x_a}$ corresponds to aaa, $\mathsf{x_b}$ corresponds to bb, and $\mathsf{x_c}$ corresponds to c. Since the length of the strings captured by the variables is bounded by 8, the relations required for the capture variables are regular relations (so the query is among those that have the same evaluation complexity as CRPQs (see Section 1.3)).

Obviously, it is also possible to describe this query by a union of CRPQs, if we just explicitly spell out all sequences over Σ of length at most 8. This, however, would be rather cumbersome and non-intuitive.

1.4.4 Flight Planning. A common example for a graph database are graphs that represent flight connections (see, e. g., [20]). We assume that places (e. g., "Paris", "Europe", "United Kingdom", "Berlin", "North America" etc.) are represented as nodes connected by arcs labelled with "located in" (e. g., such an arc points from "Paris" to "France" to "Europe", or from "San Francisco" to "California" to "USA" to "North America"). Moreover, there are arcs pointing from cities to cities labelled by acronyms for airlines (e. g., "LH", "AF", "BA", "SK", "UA") if there is such a flight operated by the airline.

A traveler has a certain round-the-world trip in mind: first she wants to fly from Novosibirsk to a city in France, then to New York by a direct flight, followed by a flight travel to San Francisco. After that, she wants to go back to Novosibirsk from some North American airport. In order to take advantage of bonus cards and special deals offered by single airlines, she wants to use few different airlines: all flights except the one from France to New York and those inside of the US should be operated by at most two different airlines, all flights inside of the US should be operated by the same airline, while the flight from France to New York can be operated by any airline. Obviously, a crucial requirement here is that we do not care for the airlines, as long as they are the same.

Assume that $\Sigma = \{LH, AF, BA, SK, UA, \ldots\}$ is the set of all airline acronyms. Then, the graph pattern shown in Figure 3 is a CRPQ with capture variables for the query described above. The expression

$$(\mathsf{x}\{\Sigma\}\mathsf{x}^*) \vee (\mathsf{x}\{\Sigma\}\mathsf{x}^*\mathsf{y}\{\Sigma\}(\mathsf{x} \vee \mathsf{y})^*)$$

on the arc from "Novosibirsk" to the city in "France", using capture variables x and y, describes flight sequences with all flights operated by the same airline (first option of the alternation) or flight sequences with all flights operated by at most two different airlines (second option of the alternation). Similarly, the expression $\mathsf{z}\{\Sigma\}\mathsf{z}^*$ on the arc from "New York" to "San Francisco" describes flight sequences with all flights operated by the same airline. By using z^* on the arc from "San Francisco" to the city in North America, we make sure that the same airline is used that operated all flights from "New York" to "San Francisco"; by using $(\mathsf{x} \vee \mathsf{y})^*$ on the arc back to "Novosibirsk", we make sure that we use the same two airlines that operated our flights to France.[3]

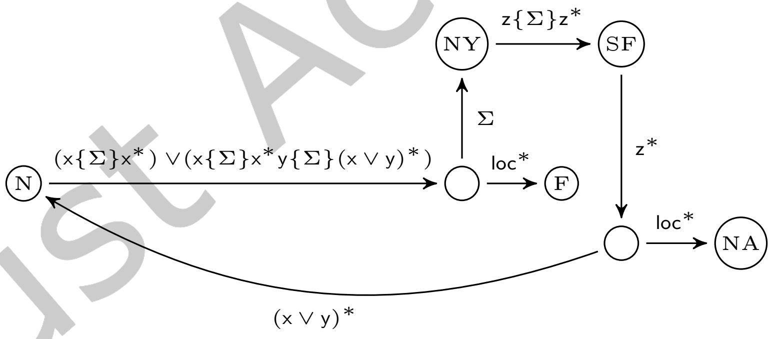


Fig. 3. Graph pattern for the round-the-world trip query. Node labels "Novosibirsk", "New York", "France", "San Francisco" and "North America" are abbreviated as "N", "NY", "F", "SF" and "NA", respectively; edge label "located in" is abbreviated as "loc".

Without going too much into technical details, let us mention a few observations about that query. First, we note that we use here variable references under a Kleene-star (e. g., x^*), which, as explained in Section 1.3, we do not want to allow, since such queries can be PSpace-hard even in data complexity. However, this particularly query is also such that variable images are just single symbols from Σ, and therefore the query is among those

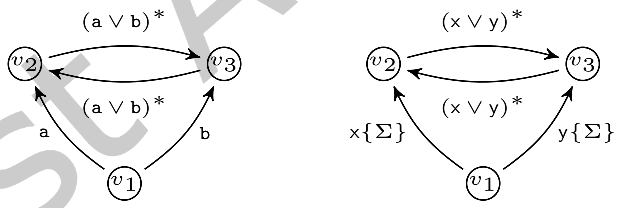
[3]Technically, it can happen that the definition of variable y is not instantiated, which means that references y correspond to the empty word.

with bounded image size, for which evaluation complexity is the same as for CRPQs without capture variables (see Section 1.3). Furthermore, we observe that also due to variable references under a Kleene-star, it seems impossible to describe this query by an ECRPQ (since this would require an unbounded number of arcs with equality relation). Nevertheless, we could just spell out all possible ways of setting the capture variables to some airlines, which means that we can describe the query by unions of classical CRPQs. However, this would yield a huge (exponential in the number of capture variables) and unwieldy collection of CRPQs. Moreover, while it is generally possible to translate CRPQs with capture variables of bounded image size into unions of classical CRPQs, this conversion is not trivial, since we have to deal with nestings of variable definitions and potentially undefined variable definitions.

1.4.5 Similar Data Over Different Signatures. If we are dealing with different databases over slightly different signatures, that all nevertheless represent data from the same domain (e. g., bibliographical data, social networks, biological networks, hereditary information, transportation networks), then a query designed for one signature might also be useful for databases over other signatures, simply because the data can be assumed to be structurally similar. For example, some query for the graph representation of DBLP should also be meaningful for graph representations of some other bibliographical databases. The problem is that different databases might use different names for the same relation (e. g., the relation "creator" of one database is called "author" in the other, "partOf" is called "includedIn" etc.) and therefore we have to translate the query by changing the relation names.

However, such an obvious mapping between signatures does not always exist. For example, databases might represent abstract data produced by empirical methods, and the task is to discover (by human experts or machine learning algorithms) queries that describe relevant patterns and structures of the data.

For example, assume that we are dealing with some huge biological network (e. g., protein networks, gene regulatory network, etc.), which is the result of extensive experimental research, and assume that we discovered that node triples (v_1, v_2, v_3) are of high relevance, if v_2 and v_3 can be reached from v_1 by one a-arc and one b-arc, respectively, and v_2 and v_3 are mutually reachable with some paths that only contain a- or b-arcs. This can be queried by a simple $q \in$ CRPQ (see the left query in Figure 4).



$(\mathtt{a} \vee \mathtt{b})^*$
v_2
v_3
$(\mathtt{a} \vee \mathtt{b})^*$
a
b
v_1
$(\mathtt{x} \vee \mathtt{y})^*$
v_2
v_3
$(\mathtt{x} \vee \mathtt{y})^*$
$\mathtt{x}\{\Sigma\}$
$\mathtt{y}\{\Sigma\}$
v_1

Fig. 4. Queries for the biological network example.

Now, in order to verify some hypotheses, biologists might want to evaluate this query in other databases of the same data domain (protein networks, gene regulatory network, etc.), but that have been produced by experiments of other research groups in other laboratories. So even though the other databases use completely different signatures, the fact that the represented data is similar justifies the hypothesis that in these database there are relations a′ and b′ that play the role of our a and b relations in the relevant query q. If Σ is the set of "unknown" relations of the new database, then the right query in Figure 4 is a suitable CRPQ with capture variables to evaluate our hypothesis.

Another example would be a CRPQ that in routing networks identifies "vulnerable links" (i. e., connections that are more likely to be subject to interruptions as others). Such a query might be the product of simulations or learning algorithms. For using it also for other routing networks with different signatures, it might again be useful to use capture variables in a similar way as described above for the biological network.

Apart from using different relation names, databases from the same data domain can also differ such that single relations in one database are refined into sequences of relations in other databases. A simple example of this would be the situation where in a network of flight connections (like the one described in Section 1.4.4) "located in" arcs are directly pointing from cities to countries, and in another network of flight connections they are first pointing to states, and then to countries. In this case, all single "located in"-arcs would translate into length two paths labelled by "located in · located in".

Again, choosing the example of the biological network from above, it can happen that the single a and b relations are represented by (unknown) chains of at most 3 single relations in another database (e. g., due to more fine-grained experimental studies). In this case, replacing the labels $x\{\Sigma\}$ and $y\{\Sigma\}$ of the right query in Figure 4 by labels $x\{\Sigma^{\leq 3}\}$ and $y\{\Sigma^{\leq 3}\}$, respectively, demonstrates an appropriate use of capture variables.

1.5 Additional Contributions

This work represents a substantially extended version of the paper "Conjunctive Regular Path Queries with String Variables" [49]. In addition to all proofs and technical details that were omitted in the conference version [49], the present paper contains the following new contributions.

In [49], it has only been shown that evaluation for CXRPQs with depth 1 is PSpace-complete in combined complexity. We extend this result and show that for any constant depth bound evaluation becomes PSpace-complete in combined complexity (see Section 6).

We also consider CXRPQs with arbitrary regular relations (Section 8), while [49] only considers CXRPQs with the equality relation. In particular, we show that if the depth is 0, then evaluation for CXRPQs with regular relations is PSpace-complete in combined complexity and NL-complete in data complexity, and if CXRPQs have bounded image size, then evaluation for CXRPQs with regular relations is NP-complete in combined complexity and NL-complete in data complexity (even for abitrary depth).

Another aspect that is not considered in [49] are the upper bounds for checking whether a given tuple is in the solution set and for computing the whole solution set (see Section 9).

1.6 Organisation of the Paper

In Section 2, we give some basic definitions and in Section 3, we define in detail the class of regular expressions with capture variables (called xregex) and we lift this concept to conjunctive xregex, which is the most important building block for CXRPQs. Moreover, as our first result, we show that checking whether a given graph database contains some path described by a xregex is PSpace-hard in data complexity. This negative result is central for our definition of CXRPQs and its fragments. A formal definition of CXRPQs will then be given in Section 4.

We shall spend quite some time on these initial sections, since we illustrate all definitions with comprehensive examples and additional explanations. This is necessary especially with respect to xregex and our new concept of conjunctive xregex, which, in comparison to the general concept of graph databases and conjunctive regular path queries, can be assumed to be less well-known by the intended audience of this paper. On the other hand, the step from conjunctive xregex to the class CXRPQ will then be straightforward.

Our upper and lower complexity bounds for CXRPQs are presented in Section 5. The fragments obtained from bounding the depth of CXRPQs and bounding the image sizes are discussed in Sections 6 and 7, respectively. Then, in Section 8, we investigate how the capture variables of CXRPQs may use arbitrary regular relations instead of only the equality relation, and in Section 9 we discuss the problems of checking whether a given tuple is in the solution set of the query, and the problem of computing the whole solution set. In Section 10, we investigate the expressive power of the query classes introduced in this work. Finally, we conclude the paper and discuss open problems and further research directions in Section 11.

For the sake of accessibility of this paper, we defer some proofs and technical content to an electronic appendix.

2 PRELIMINARIES

Let $\mathbb{N} = \{1, 2, 3, \dots\}$ and $[n] = \{1, 2, \dots, n\}$ for $n \in \mathbb{N}$. A^+ denotes the set of non-empty words over an alphabet A and $A^* = A^+ \cup \{\varepsilon\}$ (where ε is the empty word). For a word $w \in A^*$, $|w|$ denotes its length, and for $k \in \mathbb{N}$, $A^{\leq k} = \{w \in A^* \mid |w| \leq k\}$. For $w_1, w_2, \dots, w_n \in A^*$, we set $\Pi_{i=1}^n w_i = w_1 w_2 \dots w_n$, and if $w = w_i$, for every $i \in [n]$, then we also write w^n instead of $\Pi_{i=1}^n w_i$. For a word $w \in A^*$ and $b \in A$, $|w|_b$ is the number of occurrences of symbol b in w.

We fix a finite *terminal* alphabet Σ and an enumerable set $\mathcal{X}_s$ of *string variables*, where $\mathcal{X}_s \cap \Sigma = \emptyset$. As a convention, we use symbols $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \dots$ for elements from Σ, and $\mathsf{x}, \mathsf{y}, \mathsf{z}, \mathsf{x}_1, \mathsf{x}_2, \dots, \mathsf{y}_1, \mathsf{y}_2, \dots$ for variables from $\mathcal{X}_s$. We consistently use sans-serif font for string variables to distinguish them from *node variables* to be introduced later. We use *regular expressions* and (*nondeterministic*) *finite automata* (NFA for short) as commonly defined in the literature (see Section 3.1 and the remainder of this section for more details).

2.1 Graph-Databases

A *graph-database* (*over* Σ) is a directed, edge labelled multigraph $\mathcal{D} = (V_{\mathcal{D}}, E_{\mathcal{D}})$, where $V_{\mathcal{D}}$ is the set of *vertices* (or *nodes*) and $E_{\mathcal{D}} \subseteq V_{\mathcal{D}} \times \Sigma \times V_{\mathcal{D}}$ is the set of *edges* (or *arcs*). A path from $u \in V_{\mathcal{D}}$ to $v \in V_{\mathcal{D}}$ of length $k \geq 0$ is a sequence

$$p = (w_0, a_1, w_1, a_2, w_2 \dots, w_{k-1}, a_k, w_k)$$

with $(w_{i-1}, a_i, w_i) \in E_{\mathcal{D}}$ for every $i \in [k]$. We say that p is *labelled* with the *word* $a_1 a_2 \dots a_k \in \Sigma^*$. According to this definition, for every $v \in V_{\mathcal{D}}$, (v) is a path from v to v of length 0 that is labelled by ε. Hence, every node of every graph-database has an ε-labelled path to itself (and these are the only ε-labelled paths in $\mathcal{D}$).

Nondeterministic finite automata (NFAs) are just graph databases, the nodes of which are called *states*, and that have a specified *start state* and a set of specified *final states*. Moreover, we allow the empty word as edge label as well (which is not the case for graph databases). The language $\mathcal{L}(M)$ of an NFA M is the set of all labels from paths that lead from the start state to some final state.

In the following, $\mathcal{X}_n$ is an enumerable set of *node-variables*; we shall use symbols $x, y, z, x_1, x_2, \dots, y_1, y_2, \dots$ for node variables (in contrast to the string variables $\mathcal{X}_s$ in sans-serif font).

2.2 Conjunctive Path Queries

Let $\mathfrak{R}$ be a class of language descriptors, and, for every $r \in \mathfrak{R}$, let $\mathcal{L}(r)$ denote the language represented by r. An $\mathfrak{R}$-*graph pattern* is a directed, edge-labelled graph $G = (V, E)$ with $V \subseteq \mathcal{X}_n$ and $E \subseteq V \times \mathfrak{R} \times V$; it is an $\mathfrak{R}$-graph pattern *over alphabet* Σ, if $\mathcal{L}(\alpha) \subseteq \Sigma^*$ for every $(x, \alpha, y) \in E$. For an $\mathfrak{R}$-graph pattern $G = (V, E)$ over Σ and a graph-database $\mathcal{D} = (V_{\mathcal{D}}, E_{\mathcal{D}})$ over Σ, a mapping $h : V \to V_{\mathcal{D}}$ is a *matching morphism* for G and $\mathcal{D}$ if, for every $e = (x, \alpha, y) \in E$, $\mathcal{D}$ contains a path from $h(x)$ to $h(y)$ that is labelled with a word $w_e \in \mathcal{L}(\alpha)$. The tuple $(w_e)_{e \in E}$ is a tuple of *matching words* (*with respect to* h). In particular, a matching morphism can have several different tuples of matching words.

A *conjunctive* $\mathfrak{R}$-*path query* ($\mathfrak{R}$-CPQ for short) is a query $q = \bar{z} \leftarrow G_q$, where $G_q = (V_q, E_q)$ is an $\mathfrak{R}$-graph pattern and $\bar{z} = (z_1, z_2, \dots, z_\ell)$ with $\{z_1, z_2, \dots, z_\ell\} \subseteq V_q$. We say that q is an $\mathfrak{R}$-CPQ *over alphabet* Σ if G_q is an $\mathfrak{R}$-graph pattern over Σ. The query q is a *single-edge* query, if $|E_q| = 1$.

For an $\mathfrak{R}$-CPQ $q = \bar{z} \leftarrow G_q$ over Σ with $\bar{z} = (z_1, z_2, \dots, z_\ell)$, a graph-database $\mathcal{D} = (V_{\mathcal{D}}, E_{\mathcal{D}})$ over Σ and a matching morphism h for G_q and $\mathcal{D}$ (we also call h a *matching morphism for* q *and* $\mathcal{D}$), we define

$q_h(\mathcal{D}) = (h(z_1), h(z_2), \ldots, h(z_\ell))$ and we set

$$q(\mathcal{D}) = \{q_h(\mathcal{D}) \mid h \text{ is a matching morphism for } q \text{ and } \mathcal{D}\}\,.$$

The mapping $\mathcal{D} \mapsto q(\mathcal{D})$ from the set of graph-databases to the set of relations over $V_{\mathcal{D}}$ of arity ℓ that is defined by q shall be denoted by $[\![q]\!]$, and for any class A of conjunctive path queries, we set $[\![A]\!] = \{[\![q]\!] \mid q \in A\}$.

A *Boolean* $\mathfrak{R}$-CPQ has the form $\bar{z} \leftarrow G_q$ where $\bar{z}$ is the empty tuple. In this case, we also denote q just by G_q instead of $() \leftarrow G_q$. For a Boolean $\mathfrak{R}$-CPQ q and a graph-database $\mathcal{D}$, we either have $q(\mathcal{D}) = \{()\}$ or $q(\mathcal{D}) = \emptyset$, which we shall also denote by $\mathcal{D} \models q$ and $\mathcal{D} \not\models q$, respectively. For Boolean queries q, the mapping $[\![q]\!]$ can also be interpreted as the set $\{\mathcal{D} \mid \mathcal{D} \models q\}$. Two $\mathfrak{R}$-CPQs q and q' are *equivalent*, denoted by $q \equiv q'$, if $[\![q]\!] = [\![q']\!]$, i. e., $q(\mathcal{D}) = q'(\mathcal{D})$ for every graph-database $\mathcal{D}$ (or, in the Boolean case, $\mathcal{D} \models q \Leftrightarrow \mathcal{D} \models q'$).

For a class Q of conjunctive path queries, Q-Bool-Eval is the problem to decide, for a given Boolean $q \in Q$ and a graph database $\mathcal{D}$, whether $\mathcal{D} \models q$.

As common in database theory, the *combined complexity* for an algorithm solving Q-Bool-Eval is the time or space needed by the algorithm measured in both $|q|$ and $|\,\mathcal{D}\,|$, while for the *data complexity* the query q is considered constant. For simplicity, we assume $|q| = \mathrm{O}(|\,\mathcal{D}\,|)$ throughout the paper.[4]

Conjunctive regular path queries (CRPQ) are $\mathfrak{R}$-CPQ where $\mathfrak{R}$ is the class of regular expressions (which are defined in Section 3.1). See Figure 1 for examples of CRPQ. For some of our results, we need the following result about CRPQs (this seems to be a folklore result, but it is formally stated in [7]).

Lemma 2.1 ([7]). *CRPQ-Bool-Eval is NP-complete in combined complexity and NL-complete in data complexity.*

2.3 Ref-Words

The following *ref-words* (first introduced in [48]) are convenient for defining the semantics of xregex (Section 3). They have also been used in [32] and for so-called document spanners in [24, 30, 31, 50, 51]. Ref-words will be vital in our definition of conjunctive xregex (Section 4.1), which are the basis of the class CXRPQ. We start with an intuitive explanation of ref-words and then provide a formal definition.

Ref-words are words with terminal symbols from Σ and variables from $\mathcal{X}_s$. For every $\mathsf{x} \in \mathcal{X}_s$, we may use parentheses ${}^{\mathsf{x}}\triangleright \ldots \triangleleft^{\mathsf{x}}$ in order to mark a subword of the ref-word (i. e., by enclosing this subword with the parentheses). Such a subword ${}^{\mathsf{x}}\triangleright \ldots \triangleleft^{\mathsf{x}}$ is called a *definition for variable* x. These definitions must be such that, for every $\mathsf{x} \in \mathcal{X}_s$, there is at most one definition of x, which is not allowed to enclose an occurrence of x, and the definitions are not overlapping, i. e., for $\mathsf{x}, \mathsf{y} \in \mathcal{X}_s$, ${}^{\mathsf{x}}\triangleright\ {}^{\mathsf{y}}\triangleright\ \triangleleft^{\mathsf{x}}\ \triangleleft^{\mathsf{y}}$ must not occur as subsequence, but nestings in the form of ${}^{\mathsf{y}}\triangleright\ {}^{\mathsf{x}}\triangleright\ \triangleleft^{\mathsf{x}}\ \triangleleft^{\mathsf{y}}$ are allowed. The idea is that every occurrence of x describes a *reference* to the definition of x (if it exists). Since we allow nestings of definitions, i. e., definitions may contain itself references or definitions of other variables, there are chains of references, e. g., the definition of x contains references of y, but the definition of y contains references of z and so on. We nevertheless require these nestings to be acyclic. For example,

$$\mathsf{axb}\ {}^{\mathsf{x}}\triangleright\ \mathsf{ab}\ \triangleleft^{\mathsf{x}}\ \mathsf{c}\ {}^{\mathsf{y}}\triangleright\ \mathsf{xaa}\ \triangleleft^{\mathsf{y}}\ \mathsf{y}$$
$$\mathsf{axb}\ {}^{\mathsf{x}}\triangleright\ \mathsf{a}\ {}^{\mathsf{y}}\triangleright\ \mathsf{cc}\ \triangleleft^{\mathsf{y}}\ \mathsf{by}\ \triangleleft^{\mathsf{x}}\ \mathsf{y}$$

are ref-words, while the following are not:

$$\begin{array}{ll}
\mathsf{axb}\ {}^{\mathsf{x}}\triangleright\ \mathsf{ax}\ \triangleleft^{\mathsf{x}}\ \mathsf{b} & \text{(occurrence of x in definition of x)}\,, \\
\mathsf{axb}\ {}^{\mathsf{x}}\triangleright\ \mathsf{a}\ \triangleleft^{\mathsf{x}}\ \mathsf{c}\ {}^{\mathsf{x}}\triangleright\ \mathsf{b}\triangleleft^{\mathsf{x}} & \text{(two definitions of x)}\,, \\
\mathsf{ay}\ {}^{\mathsf{x}}\triangleright\ \mathsf{a}\ {}^{\mathsf{y}}\triangleright\ \mathsf{abc}\ \triangleleft^{\mathsf{x}}\ \mathsf{b}\ \triangleleft^{\mathsf{y}}\ \mathsf{c} & \text{(overlapping definitions)}\,, \\
\mathsf{axa}\ {}^{\mathsf{x}}\triangleright\ \mathsf{ayb}\ \triangleleft^{\mathsf{x}}\ \mathsf{c}\ {}^{\mathsf{y}}\triangleright\ \mathsf{xa}\triangleleft^{\mathsf{y}} & \text{(cyclicity)}\,.
\end{array}$$

[4]This is not without loss of generality, but it fits to the common assumption in databases (in both applied and theoretical considerations) that the query is small in comparison to the data.

We now give a formal definition of ref-words.

Definition 2.2 (Ref-Words). A *subword-marked* word (*over terminal alphabet* Σ *and variables* $\mathcal{X}_s$) is a word $w \in (\Sigma \cup \{{}^{\mathsf{x}}\triangleright, \triangleleft^{\mathsf{x}} \mid \mathsf{x} \in \mathcal{X}_s\} \cup \mathcal{X}_s)^*$ that, for every $\mathsf{x} \in \mathcal{X}_s$, contains the parentheses ${}^{\mathsf{x}}\triangleright$ and $\triangleleft^{\mathsf{x}}$ at most once and all these parentheses form a well-formed parenthesised expression. For every $\mathsf{x} \in \mathcal{X}_s$, a subword ${}^{\mathsf{x}}\triangleright\ v \triangleleft^{\mathsf{x}}$ in w is called a *definition* (of variable x), and an occurrence of symbol x is called a *reference* (of variable x). For a subword-marked word w over Σ and $\mathcal{X}_s$, the binary relation $\preceq_w$ over $\mathcal{X}_s$ is defined by setting $\mathsf{x} \preceq_w \mathsf{y}$ if in w there is a definition of y that contains a reference or a definition of x. A *ref-word* (*over terminal alphabet* Σ *and variables* $\mathcal{X}_s$) is a subword-marked word over Σ and $\mathcal{X}_s$, such that $\preceq_w$ is acyclic.

Note that any ref-word w over terminal alphabet Σ and variables $\mathcal{X}_s$ is also a ref-word over any terminal alphabet $\Sigma' \supset \Sigma$ and variables $\mathcal{X}'_s \supset \mathcal{X}_s$. A set L of ref-words (over Σ and $\mathcal{X}_s$) is called a *ref-language* (over Σ and $\mathcal{X}_s$).

3 XREGEX

We now define the class of regular expressions with *capture variables* (or simply regular expressions with *variables*). This definition will also provide a definition of classical regular expressions, since they can be considered a special case of regular expressions with variables. In the next two subsections, we define separately the syntax and the semantics of regular expressions with variables. After that, we provide a first insight regarding regular expressions with variables as path queries.

3.1 Syntax

Definition 3.1 (Xregex). The set $\mathsf{XRE}_{\Sigma,\mathcal{X}_s}$ of *regular expressions with variables* (*over* Σ *and* $\mathcal{X}_s$), also denoted by *xregex*, for short, is recursively defined as follows:

(1) $a \in \mathsf{XRE}_{\Sigma,\mathcal{X}_s}$ and $\mathsf{var}(a) = \emptyset$, for every $a \in \Sigma \cup \{\varepsilon\}$,
(2) $\mathsf{x} \in \mathsf{XRE}_{\Sigma,\mathcal{X}_s}$ and $\mathsf{var}(\mathsf{x}) = \{\mathsf{x}\}$, for every $\mathsf{x} \in \mathcal{X}_s$,
(3) $(\alpha \cdot \beta) \in \mathsf{XRE}_{\Sigma,\mathcal{X}_s}$, $(\alpha \vee \beta) \in \mathsf{XRE}_{\Sigma,\mathcal{X}_s}$, and $(\alpha)^+ \in \mathsf{XRE}_{\Sigma,\mathcal{X}_s}$, for every $\alpha, \beta \in \mathsf{XRE}_{\Sigma,\mathcal{X}_s}$; furthermore, $\mathsf{var}((\alpha \cdot \beta)) = \mathsf{var}((\alpha \vee \beta)) = \mathsf{var}(\alpha) \cup \mathsf{var}(\beta)$ and $\mathsf{var}((\alpha)^+) = \mathsf{var}(\alpha)$,
(4) $\mathsf{x}\{\alpha\} \in \mathsf{XRE}_{\Sigma,\mathcal{X}_s}$ and $\mathsf{var}(\mathsf{x}\{\alpha\}) = \mathsf{var}(\alpha) \cup \{\mathsf{x}\}$, for every $\alpha \in \mathsf{XRE}_{\Sigma,\mathcal{X}_s}$ and $\mathsf{x} \in \mathcal{X}_s \setminus \mathsf{var}(\alpha)$.

For technical reasons, we also add $\varnothing$ to $\mathsf{XRE}_{\Sigma,\mathcal{X}_s}$. For $\alpha \in \mathsf{XRE}_{\Sigma,\mathcal{X}_s}$, we use r^* as a shorthand form for $r^+ \vee \varepsilon$, and we usually omit the operator '$\cdot$', i. e., we use juxtaposition. If this does not cause ambiguities, we often omit parenthesis. For example, x, $\mathsf{x}\{\mathsf{ya}\}$ and $\mathsf{x}\{(\mathsf{y}\{\mathsf{z}\{\mathsf{a}^* \vee \mathsf{bc}\}\mathsf{a}\}\mathsf{y})^+\mathsf{b}\}\mathsf{x}$ are all xregex, while neither $\mathsf{x}\{\mathsf{ax}\}\mathsf{b}$ nor $\mathsf{x}\{\mathsf{ax}\{\mathsf{b}^*\}\mathsf{a}\}\mathsf{b}$ is an xregex. The operations $\cdot$, $\vee$, $+$ and $*$ are also called *concatenation*, *alternation*, *plus* and *star*, respectively.

We call an occurrence of $\mathsf{x} \in \mathcal{X}_s$ a *reference of variable* x and a subexpression $\mathsf{x}\{\alpha\}$ a *definition of variable* x. The set $\mathsf{XRE}_{\Sigma,\emptyset}$ (or, equivalently, the set of expressions defined by the first three points of Definition 3.1) is exactly the set of regular expressions over Σ, which shall be denoted by RE_Σ in the following. We also use the term *classical* regular expressions for a clearer distinction from xregex. If the underlying alphabet Σ or set $\mathcal{X}_s$ of variables is clear from the context, we also drop these and simply write XRE and RE.

Example 3.2. The following are examples of xregex:

- $\alpha_1 = (\mathsf{a} \vee \mathsf{b})^*\mathsf{c}(\mathsf{a} \vee \mathsf{b})^*$,
- $\alpha_2 = \mathsf{x}\{(\mathsf{a} \vee \mathsf{b})^*\}\mathsf{c}(\mathsf{a} \vee \mathsf{x})^*$,
- $\alpha_3 = \mathsf{x}\{\mathsf{y}\{\mathsf{a} \vee \mathsf{b}\}\mathsf{b}^*\}\mathsf{cy}\{(\mathsf{a} \vee \mathsf{b})^*\mathsf{x}\}$,
- $\alpha_4 = (\mathsf{x}\{\mathsf{a}^*\} \vee \mathsf{x}\{\mathsf{b}\})\mathsf{c}^*\mathsf{x}$,
- $\alpha_5 = (\mathsf{ax}\{\mathsf{b}^*\})^*\mathsf{cy}\{\mathsf{x}(\mathsf{b} \vee \mathsf{c})\}$.

It can be easily seen that xregex are descriptors for ref-languages. We shall now make this more precise. For any $\alpha \in \mathsf{XRE}_{\Sigma, \mathcal{X}_s}$, let α_{ref} be the classical regular expression over the alphabet $\Sigma \cup \mathcal{X}_s \cup \{{}^{\mathsf{x}}\triangleright, \triangleleft^{\mathsf{x}} \mid \mathsf{x} \in \mathcal{X}_s\}$ that is obtained from α by iteratively replacing all variable definitions $\mathsf{x}\{\beta\}$ by ${}^{\mathsf{x}}\triangleright\, \beta \triangleleft^{\mathsf{x}}$. For example,

$$\alpha = \mathsf{x}\{(\mathsf{y}\{\mathsf{z}\{\mathsf{a}^* \vee \mathsf{bc}\}\mathsf{a}\}\mathsf{y})^+\mathsf{b}\}\mathsf{x}\,,$$
$$\alpha_{\text{ref}} = {}^{\mathsf{x}}\triangleright\, (\,{}^{\mathsf{y}}\triangleright\, {}^{\mathsf{z}}\triangleright\, (\mathsf{a}^* \vee \mathsf{bc}) \triangleleft^{\mathsf{z}} \mathsf{a} \triangleleft^{\mathsf{y}} \mathsf{y})^+\mathsf{b} \triangleleft^{\mathsf{x}} \mathsf{x}\,.$$

We say that α is *valid* if every $w \in \mathcal{L}(\alpha_{\text{ref}})$ contains for every $\mathsf{x} \in \mathcal{X}_s$ at most one occurrence of ${}^{\mathsf{x}}\triangleright$. For example, α is not valid. If not explicitly stated otherwise, we assume from now on that all our xregex are valid. Note that for α to be valid, it is also necessary that variable definitions are not subject to the +-operator. The example xregex α_1, α_2 and α_4 of Example 3.2 are valid, while α_3 and α_5 are not valid: $\mathcal{L}_{\text{ref}}(\alpha_3)$ contains ${}^{\mathsf{x}}\triangleright\ {}^{\mathsf{y}}\triangleright\, \mathsf{a} \triangleleft^{\mathsf{y}} \mathsf{b} \triangleleft^{\mathsf{x}} \mathsf{c}\ {}^{\mathsf{y}}\triangleright\, \mathsf{bx} \triangleleft^{\mathsf{y}}$, and $\mathcal{L}_{\text{ref}}(\alpha_5)$ contains $\mathsf{a}\ {}^{\mathsf{x}}\triangleright\, \mathsf{b} \triangleleft^{\mathsf{x}} \mathsf{a}\ {}^{\mathsf{x}}\triangleright\, \mathsf{b} \triangleleft^{\mathsf{x}} \mathsf{c}\ {}^{\mathsf{y}}\triangleright\, \mathsf{xc} \triangleleft^{\mathsf{y}}$.

If an xregex α is valid, then every $w \in \mathcal{L}(\alpha_{\text{ref}})$ must be a ref-word. Indeed, this directly follows from the fact that the definitions $\mathsf{x}\{\ldots\}$ are always subexpressions. Hence, for valid xregex, $\mathcal{L}(\alpha_{\text{ref}})$ is a ref-language, which we shall denote by $\mathcal{L}_{\text{ref}}(\alpha)$.

If a ref-word $v \in \mathcal{L}_{\text{ref}}(\alpha)$ contains a definition ${}^{\mathsf{x}}\triangleright\, v_{\mathsf{x}} \triangleleft^{\mathsf{x}}$, then we say that the corresponding definition $\mathsf{x}\{\gamma_x\}$ in α is *instantiated* (by v). In particular, we observe that valid xregex can nevertheless have several definitions for the same variable x, but at most one of them is instantiated by any ref-word. This is due to the fact that these definitions can occur in different branches of alternations (see xregex α_4 of Example 3.2). Furthermore, the fourth point of Definition 3.1 makes sure that valid xregex can only generate acyclic ref-words. Also note that in general, in valid xregex no variable definition can occur under a star.

3.2 Semantics

The semantics of the subclass RE_{Σ}, i. e., classical regular expressions, is defined in the usual way:

- $\mathcal{L}(a) = \{a\}$, for every $a \in \Sigma \cup \{\varepsilon\}$,
- $\mathcal{L}(\alpha \cdot \beta) = \mathcal{L}(\alpha) \cdot \mathcal{L}(\beta)$, for every $\alpha, \beta \in \mathsf{RE}_{\Sigma}$,
- $\mathcal{L}(\alpha \vee \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$, for every $\alpha, \beta \in \mathsf{RE}_{\Sigma}$,
- $\mathcal{L}(\alpha^+) = \mathcal{L}(\alpha)^+$, for every $\alpha \in \mathsf{RE}_{\Sigma}$.

In order to define the semantics for $\mathsf{XRE}_{\Sigma, \mathcal{X}_s}$, we first define semantics for ref-words. To this end, we define a dereference function, which maps ref-words over Σ and $\mathcal{X}_s$ to words over Σ. It is helpful to recall that, intuitively, variable references x function as pointers to the subword specified by the definition ${}^{\mathsf{x}}\triangleright \ldots \triangleleft^{\mathsf{x}}$.

Definition 3.3 (Deref-Function). For a ref-word w over Σ and $\mathcal{X}_s$, $\text{deref}_{\Sigma, \mathcal{X}_s}(w) \in \Sigma^*$ is the word obtained from w by the following procedure:

(1) Remove all occurrences of $\mathsf{x} \in \mathcal{X}_s$ without definition in w.
(2) Repeat until we have obtained a word over Σ:
  (a) Let ${}^{\mathsf{x}}\triangleright\, v_{\mathsf{x}} \triangleleft^{\mathsf{x}}$ be a definition such that $v_{\mathsf{x}} \in \Sigma^*$.
  (b) Replace ${}^{\mathsf{x}}\triangleright\, v_{\mathsf{x}} \triangleleft^{\mathsf{x}}$ by v_{x}, and replace each occurrences of x in w by v_{x}.

We lift the function $\text{deref}_{\Sigma, \mathcal{X}_s}$ from ref-words to ref-languages in the obvious way, i. e., for a ref-language L, we define $\text{deref}_{\Sigma, \mathcal{X}_s}(L) = \{\text{deref}_{\Sigma, \mathcal{X}_s}(w) \mid w \in L\}$. If the underlying alphabet Σ or set $\mathcal{X}_s$ of variables is clear from the context, we also write deref instead of $\text{deref}_{\Sigma, \mathcal{X}_s}$.

The following is easy to see (a formal proof is provided in the Appendix).

Proposition 3.4. *The function* $\text{deref}_{\Sigma, \mathcal{X}_s}(w)$ *is well-defined.*

Example 3.5. We consider the following ref-words over $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ and $\mathcal{X}_s = \{\mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3\}$:

$$\begin{aligned} u_1 &= \mathsf{aaaa}\ {}^{\mathsf{x}_1}\!\triangleright\ {}^{\mathsf{x}_2}\!\triangleright\ \mathsf{ba} \triangleleft^{\mathsf{x}_2}\ \mathsf{baa} \triangleleft^{\mathsf{x}_1}\ \mathsf{x}_2\mathsf{x}_2\mathsf{bbax}_1\,, \\ u_2 &= \mathsf{aaa}\ {}^{\mathsf{x}_1}\!\triangleright\ \mathsf{a}\ {}^{\mathsf{x}_2}\!\triangleright\ \mathsf{bab} \triangleleft^{\mathsf{x}_2}\ \mathsf{aa} \triangleleft^{\mathsf{x}_1}\ \mathsf{x}_2\mathsf{abbx}_1\,, \\ u_3 &= {}^{\mathsf{x}_1}\!\triangleright\ \mathsf{cc}\ {}^{\mathsf{x}_2}\!\triangleright\ \mathsf{aa} \triangleleft^{\mathsf{x}_2}\ \triangleleft^{\mathsf{x}_1}\ \mathsf{cx}_2\mathsf{cx}_3\mathsf{bx}_1\,. \end{aligned}$$

By definition, we have

$$\begin{aligned} \mathsf{deref}(u_1) = \mathsf{deref}(u_2) &= \mathsf{aaaababaababababbababaa} \\ \mathsf{deref}(u_3) &= \mathsf{ccaacaacbccaa}\,. \end{aligned}$$

Note that the reference x_3 in u_3 has no definition and is therefore removed by Step 1 of Definition 3.3.

The language described by a valid $\alpha \in \mathsf{XRE}_{\Sigma,\mathcal{X}_s}$ is now defined as $\mathcal{L}(\alpha) = \mathsf{deref}(\mathcal{L}_{\mathsf{ref}}(\alpha))$. As a special case, we also define $\mathcal{L}(\varnothing) = \emptyset$.

Example 3.6. Let $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ and $\mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3 \in \mathcal{X}_s$, let $\alpha, \beta \in \mathsf{XRE}_{\Sigma,\mathcal{X}_s}$ be defined by

$$\begin{aligned} \alpha &= \mathsf{a}^*\mathsf{x}_1\{\mathsf{a}^*\mathsf{x}_2\{(\mathsf{a} \vee \mathsf{b})^*\}\mathsf{b}^*\mathsf{a}^*\}\mathsf{x}_2^*(\mathsf{a} \vee \mathsf{b})^*\mathsf{x}_1\,, \\ \beta &= \mathsf{x}_1\{\mathsf{c}^*(\mathsf{x}_2\{\mathsf{a}^*\} \vee \mathsf{x}_3\{\mathsf{b}^*\})\}\mathsf{cx}_2\mathsf{cx}_3\mathsf{bx}_1\,. \end{aligned}$$

The classical regular expressions that describe $\mathcal{L}_{\mathsf{ref}}(\alpha)$ and $\mathcal{L}_{\mathsf{ref}}(\beta)$ are given by

$$\begin{aligned} \alpha_{\mathsf{ref}} &= \mathsf{a}^*\ {}^{\mathsf{x}_1}\!\triangleright\ \mathsf{a}^*\ {}^{\mathsf{x}_2}\!\triangleright\ (\mathsf{a} \vee \mathsf{b})^* \triangleleft^{\mathsf{x}_2}\ \mathsf{b}^*\mathsf{a}^* \triangleleft^{\mathsf{x}_1}\ \mathsf{x}_2^*(\mathsf{a} \vee \mathsf{b})^*\mathsf{x}_1\,, \\ \beta_{\mathsf{ref}} &= {}^{\mathsf{x}_1}\!\triangleright\ \mathsf{c}^*(\ {}^{\mathsf{x}_2}\!\triangleright\ \mathsf{a}^* \triangleleft^{\mathsf{x}_2} \vee\ {}^{\mathsf{x}_3}\!\triangleright\ \mathsf{b}^* \triangleleft^{\mathsf{x}_3}) \triangleleft^{\mathsf{x}_1}\ \mathsf{cx}_2\mathsf{cx}_3\mathsf{bx}_1\,. \end{aligned}$$

Let u_1, u_2, u_3 be the ref-words from Example 3.5. Then, by definition, $w_1 = \mathsf{a}^4(\mathsf{ba})^2(\mathsf{ab})^3(\mathsf{ba})^3\mathsf{a} \in \mathcal{L}(\alpha)$, since $\mathsf{deref}(u_1) = w_1$ and $u_1 \in \mathcal{L}_{\mathsf{ref}}(\alpha)$; and $w_3 = \mathsf{c}^2\mathsf{a}^2\mathsf{ca}^2\mathsf{cbc}^2\mathsf{a}^2 \in \mathcal{L}(\beta)$, since $\mathsf{deref}(u_3) = w_3$ and $u_3 \in \mathcal{L}_{\mathsf{ref}}(\beta)$.

3.3 Xregex as Path Queries

The membership problem for XRE (i. e., checking whether $w \in \mathcal{L}(\alpha)$ for given $w \in \Sigma$ and $\alpha \in \mathsf{XRE}_{\Sigma,\mathcal{X}_s}$) is NP-complete (but only in combined complexity, i. e., α is part of the input). In other words, adding variables to regular expressions causes a substantial increase in complexity (since the membership problem for classical regular expressions can be solved quite efficiently). This leads to the question of how adding variables changes the complexity of regular path queries.

Any language descriptor r from some class $\mathfrak{R}$ of language descriptors can be interpreted as a simple path query that returns all node pairs (u, v) (or checks whether such a pair exists) that are connected by a path labelled with a word from $\mathcal{L}(r)$; or, in the terminology of Section 2.2, as the single-edge conjunctive $\mathfrak{R}$-path query $q = (x, y) \leftarrow G_q$ with $G_q = (\{x, y\}, \{(x, r, y)\})$. It can be easily seen that if $\mathfrak{R}$ is the class of classical regular expressions, then such queries can be evaluated efficiently (e. g., see [5]); and, since the membership problem for the string case can be easily expressed as a graph query problem, evaluation should become at least NP-hard if we let $\mathfrak{R}$ be the class XRE. Unfortunately, in the graph case, adding variables to regular expressions causes a much more severe increase in complexity: there is a fixed $\alpha \in \mathsf{XRE}$, such that checking whether a given graph contains a path labelled with some word form $\mathcal{L}(\alpha)$ is PSpace-hard.

Let $\Delta = \{\mathsf{a}, \mathsf{b}, \#\}$ and let $\mathcal{X}_s = \{\mathsf{x}\}$. We define the xregex $\alpha_{\mathsf{ni}} = \#\mathsf{x}\{(\mathsf{a} \vee \mathsf{b})^*\}\,(\#\#\,\mathsf{x})^*\#\#\# \in \mathsf{XRE}_{\Delta,\mathcal{X}_s}$.

Theorem 3.7. *Deciding whether a given graph-database over* $\Sigma \supseteq \Delta$ *contains a path labelled with some* $w \in \mathcal{L}(\alpha_{ni})$ *is* PSpace-*hard.*

Proof. We will prove the result by a reduction from the PSpace-complete NFA-intersection problem over binary alphabet $\{\mathsf{a}, \mathsf{b}\}$, which is defined as follows: Given NFA $M_1, \ldots, M_k$ over alphabet $\{\mathsf{a}, \mathsf{b}\}$, decide whether or not $\bigcap_{i=1}^{k} \mathcal{L}(M_i) \neq \emptyset$.

Without loss of generality, we assume that, for every $i \in [k]$, M_i has state set Q_i, transition function δ_i, initial state $q_{0,i}$, only one accepting state $q_{f,i}$, and we also assume that $\bigcap_{i=1}^{k} Q_i = \emptyset$. We transform the NFA $M_1, \ldots, M_k$ into a graph-database $\mathcal{D} = (V_{\mathcal{D}}, E_{\mathcal{D}})$ over alphabet $\Delta = \{\mathsf{a}, \mathsf{b}, \#\}$ as follows. For the sake of convenience, we assume that arcs can also be labelled by the words ## and ### (technically, these would be paths of length 2 and 3, respectively, instead of single arcs). We set $V_{\mathcal{D}} = (\bigcup_{i=1}^{k} Q_i) \cup \{s, t\}$, where $\{s, t\} \cap (\bigcup_{i=1}^{k} Q_i) = \emptyset$, and we set

$$E_{\mathcal{D}} = \left(\bigcup_{i=1}^{k} \delta_i\right) \cup \{(q_{f,i}, \#\#, q_{0,i+1}) \mid 1 \leq i \leq k-1\} \cup \{(s, \#, q_{0,1}), (q_{f,k}, \#\#\#, t)\}\,.$$

We first note that in $\mathcal{D}$ there is a path labelled with a word from $\mathcal{L}(\alpha_{\mathsf{ni}})$ if and only if there is such a path from node s to node t. Let us now assume that there is a path in $\mathcal{D}$ from s to t labelled with a word $w \in \mathcal{L}(\alpha_{\mathsf{ni}})$. Since $w \in \mathcal{L}(\alpha_{\mathsf{ni}})$, we can conclude that $w = \#w'(\#\#w')^{\ell}\#\#\#$ for some $w' \in \{\mathsf{a}, \mathsf{b}\}^*$ and $\ell \geq 0$. By the structure of $\mathcal{D}$, this directly implies that $\ell = k-1$ and, for every $i \in [k]$, there is a path labelled with w' from $q_{0,i}$ to $q_{f,i}$. Consequently, $w' \in \bigcap_{i=1}^{k} \mathcal{L}(M_i)$. On the other hand, if there is some word $w' \in \bigcup_{i=1}^{k} \mathcal{L}(M_i)$, then, for every $i \in [k]$, there is a path labelled with w' from $q_{0,i}$ to $q_{f,i}$, which directly implies that there is a path from s to t labelled with $\#w'(\#\#w')^{k-1}\#\#\# \in \mathcal{L}(\alpha_{\mathsf{ni}})$. □

We shall briefly discuss this negative result. While the evaluation of CRPQ is only NP-hard in combined complexity and can be solved in polynomial-time in data complexity (see Lemma 2.1), using XRE as path queries leads to PSpace-hardness even in data-complexity, and we do not even need *conjunctive* queries for this, i. e., single-edge queries are enough. This seems to rule out variables for regular path queries altogether. However, we shall see next that minor restrictions are enough to obtain classes of conjunctive XRE-path queries that have acceptable complexities. Moreover, the benefit of the negative result of Theorem 3.7 will be that the query α_{ni} indicates which such restrictions are successful.

4 CONJUNCTIVE XREGEX PATH QUERIES

In this section, we define a class of conjunctive path queries based on XRE. Technically, we could consider conjunctive XRE-path queries as defined in Section 2.2, but this would mean that we have CRPQ in which regular expressions are just replaced by the more powerful class XRE, without any possibility to distribute variables references over different arcs of the graph patterns. Consequently, we first have to define the concept of *conjunctive xregex* (Section 4.1), which are then used to as the base of our class of conjunctive XRE-path queries (Section 4.2). Moreover, this has to be done in such a way that we avoid the PSpace-hardness in data complexity pointed out by Theorem 3.7.

4.1 Conjunctive Xregex

Definition 4.1 (Conjunctive Xregex). A tuple $\bar{\alpha} = (\alpha_1, \ldots, \alpha_m) \in (\mathsf{XRE}_{\Sigma, \mathcal{X}_s})^m$ is a *conjunctive xregex of dimension* m, if $\alpha_1\alpha_2 \ldots \alpha_m$ is a valid xregex.

We note that all α_i being valid is necessary, but not sufficient for $\alpha_1\alpha_2 \ldots \alpha_m$ being valid, e. g., two valid α_i and α_j with $i \neq j$ may have both a definition for variable x, which makes $\alpha_1\alpha_2 \ldots \alpha_m$ invalid. Since xregex can have references of undefined variables, undefined references of x in some α_i may nevertheless become defined references in $\alpha_1\alpha_2 \ldots \alpha_m$ due to a definition of x in some α_j with $i \neq j$.

By m-$\mathsf{CXRE}_{\Sigma, \mathcal{X}_s}$, we denote the set of conjunctive xregex of dimension m (over Σ and $\mathcal{X}_s$) and we set $\mathsf{CXRE}_{\Sigma, \mathcal{X}_s} = \bigcup_{m \geq 1} m\text{-}\mathsf{CXRE}_{\Sigma, \mathcal{X}_s}$. Note that $1\text{-}\mathsf{CXRE}_{\Sigma, \mathcal{X}_s} = \mathsf{XRE}_{\Sigma, \mathcal{X}_s}$. If the terminal alphabet Σ or set $\mathcal{X}_s$ of variables are not important or clear from the context, we also drop the corresponding subscripts. We also write $\bar{\alpha}[i]$ to denote the i^{th} element of some $\bar{\alpha} \in m\text{-}\mathsf{CXRE}$.

Let $\# \notin \Sigma \cup \mathcal{X}_s$ be a new symbol. A tuple $\bar{w} = (w_1, w_2, \ldots, w_m) \in (\Sigma^*)^m$ is a (*conjunctive*) *match* for $\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m) \in (\mathsf{XRE}_{\Sigma, \mathcal{X}_s})^m$ if $w_1 \# w_2 \# \ldots \# w_m \in \mathcal{L}(\alpha_1 \# \alpha_2 \# \ldots \# \alpha_m)$.

For an $\bar{\alpha} \in \mathsf{CXRE}$, $\mathcal{L}(\bar{\alpha})$ is the *set of conjunctive matches* for $\bar{\alpha}$. We say that conjunctive xregex $\bar{\alpha}$ and $\bar{\beta}$ are *equivalent* if $\mathcal{L}(\bar{\alpha}) = \mathcal{L}(\bar{\beta})$ (note that $\bar{\alpha}$ and $\bar{\beta}$ having the same dimension is necessary for this). Note that changing the order of the elements of the tuple $\bar{\alpha}$ changes both the conjunctive xregex and its set of conjunctive matches.

Example 4.2. Consider the xregex

$$\alpha_1 = \mathsf{x}_2\{\mathsf{x}_1 \vee \mathsf{a}^*\}\mathsf{b}\,, \qquad \alpha_2 = \mathsf{x}_1\{(\mathsf{a} \vee \mathsf{b})^*\}\mathsf{x}_3\{\mathsf{c}^*\}\mathsf{b}\mathsf{x}_3\,,$$
$$\alpha_3 = \mathsf{x}_2^*\mathsf{a}^*\mathsf{x}_1\,, \qquad \alpha_4 = \mathsf{x}_4\{\mathsf{a}^*\}\mathsf{b}\mathsf{x}_4\mathsf{x}_1\{\mathsf{x}_2\mathsf{a}\}\,.$$

The tuple (α_2, α_4) is not a conjunctive xregex since $\alpha_2\alpha_4$ is not a valid xregex (it has two definitions of x_1). The tuple (α_3, α_4) is a conjunctive xregex (note that there are references, but no definition for x_2, which, by definition, is not a contradiction to $\alpha_3\alpha_4$ being a valid xregex). The tuple $(\alpha_1, \alpha_2, \alpha_3)$ is also a conjunctive xregex (here, all variables with a reference also have a definition).

For $w_1 = \mathsf{bab}$, $w_2 = \mathsf{bacbc}$, $w_3 = \mathsf{babaaba}$, the tuple (w_1, w_2, w_3) is an example of a conjunctive match for $(\alpha_1, \alpha_2, \alpha_3)$, since $w_1 \# w_2 \# w_3 \in \mathcal{L}(\alpha_1 \# \alpha_2 \# \alpha_3)$. This can also be illustrated by considering suitable ref-words as witnesses: $u_1 = {}^{\mathsf{x}_2}\!\triangleright\, \mathsf{x}_1 \triangleleft^{\mathsf{x}_2} \mathsf{b} \in \mathcal{L}_{\mathsf{ref}}(\alpha_1)$, $u_2 = {}^{\mathsf{x}_1}\!\triangleright\, \mathsf{ba} \triangleleft^{\mathsf{x}_1}\ {}^{\mathsf{x}_3}\!\triangleright\, \mathsf{c} \triangleleft^{\mathsf{x}_3} \mathsf{b}\mathsf{x}_3 \in \mathcal{L}_{\mathsf{ref}}(\alpha_2)$, and $u_3 = \mathsf{x}_2\mathsf{x}_2\mathsf{a}\mathsf{x}_1 \in \mathcal{L}_{\mathsf{ref}}(\alpha_3)$, which yield the ref-word $u_1 \# u_2 \# u_3 \in \mathcal{L}_{\mathsf{ref}}(\alpha_1 \# \alpha_2 \# \alpha_3)$ with $\mathsf{deref}(u_1 \# u_2 \# u_3) = w_1 \# w_2 \# w_3$.

On the other hand, $w_1' = \mathsf{aab}$, $w_2' = \mathsf{bbacb}$ and $w_3' = \mathsf{aa}$ are members of the languages $\mathcal{L}(\alpha_1)$, $\mathcal{L}(\alpha_2)$ and $\mathcal{L}(\alpha_3)$, respectively, but (w_1', w_2', w_3') is not a conjunctive match for $(\alpha_1, \alpha_2, \alpha_3)$, since $w_1' \# w_2' \# w_3' \notin \mathcal{L}(\alpha_1 \# \alpha_2 \# \alpha_3)$. This can be easily seen: any ref-word u_2' with $\mathsf{deref} u_2' = \mathsf{bbacb}$ must be such that x_1's definition contains occurrences of b, and since any ref-word $u_3' \in \mathcal{L}_{\mathsf{ref}}(\alpha_3)$ contains a reference of x_1, but $w_3' = \mathsf{aa}$, it is impossible that $\mathsf{deref}(u_1' \# u_2' \# u_3') = w_1' \# w_2' \# w_3'$.

Remark 4.3. *A helpful point of view for a (conjunctive) match is to consider the mapping $\mathcal{X}_s \to \Sigma^*$ induced by the ref-word witnesses and the function* deref, *i. e., the replacement of definitions and references of variables by words over Σ in order to obtain $w_1 \# w_2 \# \ldots \# w_m$ from $u_1 \# u_2 \# \ldots \# u_m$. Since such a mapping is only uniquely determined by the ref-word witnesses, the same conjuncive match can have several different such mappings. This is illustrated by Example 3.5: The ref-words u_1 and u_2 satisfy* $\mathsf{deref}(u_1) = \mathsf{deref}(u_2)$, *but the mapping induced by u_1 is* $\mathsf{x}_1 \mapsto \mathsf{babaa}$, $\mathsf{x}_2 \mapsto \mathsf{ba}$, *while the mapping induced by u_2 is* $\mathsf{x}_1 \mapsto \mathsf{ababaa}$, $\mathsf{x}_2 \mapsto \mathsf{bab}$.

In Example 4.2, the mapping induced by the ref-words (u_1, u_2, u_3) that witness the conjunctive match (w_1, w_2, w_3) is $\mathsf{x}_1 \mapsto \mathsf{ba}$, $\mathsf{x}_2 \mapsto \mathsf{ba}$, $\mathsf{x}_3 \mapsto \mathsf{c}$.

While keeping this mapping in mind is helpful for our intuition, it shall also play a more central role (and shall be defined more formally) in Section 7.1.

A special class of conjunctive xregex is $m\text{-}\mathsf{CXRE}_{\Sigma, \emptyset}$, i. e., the class of all m-dimensional tuples of classical regular expressions. For every $\bar{\alpha} \in m\text{-}\mathsf{CXRE}_{\Sigma, \emptyset}$, $\mathcal{L}(\bar{\alpha}) = \mathcal{L}(\bar{\alpha}[1]) \times \mathcal{L}(\bar{\alpha}[2]) \times \ldots \times \mathcal{L}(\bar{\alpha}[m])$.

The following lemma is straightforward (a formal proof is given in the Appendix), but nevertheless helpful in our proofs. It also points out how the semantics of a conjunctive xregex $\bar{\alpha}$ depends on the ref-languages of the individual components $\bar{\alpha}[i]$.

Lemma 4.4. *Let $\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m) \in m\text{-}\mathsf{CXRE}_{\Sigma, \mathcal{X}_s}$ and let $\beta_1, \beta_2, \ldots, \beta_m \in \mathsf{XRE}_{\Sigma, \mathcal{X}_s}$ such that, for every $i \in [m]$, $\mathcal{L}_{ref}(\alpha_i) = \mathcal{L}_{ref}(\beta_i)$. Then $\bar{\beta} = (\beta_1, \beta_2, \ldots, \beta_m)$ is a conjunctive xregex with $\mathcal{L}(\bar{\alpha}) = \mathcal{L}(\bar{\beta})$.*

4.2 Conjunctive Xregex Path Queries

An xregex α is *variable-star free* (*vstar-free*, for short), if in α no +-operator applies to a variable reference.[5] A conjunctive xregex $\bar{\alpha}$ of dimension m is vstar-free if, for every $i \in [m]$, $\bar{\alpha}[i]$ is vstar-free. We discuss the meaning of this restriction in more detail in Section 5, when we investigate complexity issues.

Finally, we define conjunctive xregex path queries:

*Definition 4.5 (*CXRPQ*).* A *conjunctive xregex path query over* Σ *and* $\mathcal{X}_s$ ($\mathsf{CXRPQ}_{\Sigma,\mathcal{X}_s}$ for short) is an $\mathfrak{R}$-CPQ $q = \bar{z} \leftarrow G_q$, where $\mathfrak{R} = \mathsf{XRE}_{\Sigma,\mathcal{X}_s}$, $G_q = (V_q, E_q)$ with $E_q = \{(x_i, \alpha_i, y_i) \mid i \in [m]\}$ and $\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ is a vstar-free conjunctive xregex (which is also called *the conjunctive xregex of* q).

The semantics are defined by combining the semantics of conjunctive path queries and conjunctive xregex in the obvious way. More precisely, let $q = \bar{z} \leftarrow G_q$ be a $\mathsf{CXRPQ}_{\Sigma,\mathcal{X}_s}$, where $G_q = (V_q, E_q)$ with $E_q = \{(x_i, \alpha_i, y_i) \mid i \in [m]\}$, and let $\mathcal{D}$ be a graph database. Then $h : V_q \to V_{\mathcal{D}}$ is a matching morphism for q and $\mathcal{D}$ if there is a conjuctive match $(w_1, w_2, \ldots, w_m)$ of $\bar{\alpha}$, such that, for every $i \in [m]$, $\mathcal{D}$ contains a path from $h(x_i)$ to $h(y_i)$ labelled with w_i. Note that this definition of a matching morphism h for $\mathsf{CXRPQ}_{\Sigma,\mathcal{X}_s}$ also yields definitions for $q_h(\mathcal{D})$, $q(\mathcal{D})$, $\mathcal{D} \models q$, $[\![q]\!]$ and $[\![\mathsf{CXRPQ}_{\Sigma,\mathcal{X}_s}]\!]$ (see Section 2.2).

The graph patterns of Figure 2 in Section 1 show some examples of CXRPQs.

Since, for a $q \in \mathsf{CXRPQ}_{\Sigma,\mathcal{X}_s}$ and a fixed graph database $\mathcal{D}$, the set $q(\mathcal{D})$ is completely determined by the corresponding graph pattern and the set of conjunctive matches of the corresponding conjunctive xregex, the following can directly be concluded from the definitions.

Proposition 4.6. *Let* $q = \bar{z} \leftarrow G_q$ *be a* $\mathsf{CXRPQ}_{\Sigma,\mathcal{X}_s}$ *with conjunctive xregex* $\bar{\alpha} \in m$-CXRE. *Let* $\bar{\beta} \in m$-CXRE *and let* $q' = \bar{z} \leftarrow G_{q'}$ *be a* $\mathsf{CXRPQ}_{\Sigma,\mathcal{X}_s}$ *where* $G_{q'}$ *is obtained from* G_q *by replacing each edge label* $\bar{\alpha}[i]$ *by* $\bar{\beta}[i]$*, for every* $i \in [m]$*. If* $\mathcal{L}(\bar{\alpha}) = \mathcal{L}(\bar{\beta})$*, then* $q \equiv q'$*.*

In this paper, we mainly concentrate on queries with conjunctive xregex that are variable star-free (i. e., the class CXRPQ as given by Definition 4.5, and fragments thereof), and this decision is well motivated by Theorem 3.7 (a discussion of this issue follows in Section 5). However, we shall briefly consider queries without that restriction in Sections 10 (when we discuss expressive power) and in Sections 7 and 8.3, where upper bounds also hold for the unrestricted variant. Therefore, let us denote by $\mathsf{CXRPQ}^{\mathsf{vs}}$ the class of queries obtained by dropping from Definition 4.5 the requirement that the conjunctive xregex must be variable-star free (since we have defined conjunctive xregex without the restriction of vstar-freeness, this is well-defined).

The query of Figure 3 and the right query of Figure 4 are examples of $\mathsf{CXRPQ}^{\mathsf{vs}}$s that are not CXRPQs.

5 THE COMPLEXITY OF CXRPQ-EVALUATION

We now investigate the complexity of CXRPQ-evaluation, by investigating the problem CXRPQ-Bool-Eval: Given a Boolean $q \in \mathsf{CXRPQ}_{\Sigma,\mathcal{X}_s}$, and a graph database $\mathcal{D}$, check whether $\mathcal{D} \models q$.

The main result of this section is as follows.

Theorem 5.1. CXRPQ-*Bool-Eval is*

- *in* ExpSpace*, but* PSpace*-hard, with respect to combined-complexity and*
- NL*-complete with respect to data complexity.*

Before proving the lower and upper bounds of Theorem 5.1 in Sections 5.1 and 5.2, respectively, we discuss this result in more detail.

[5]Since we use the Kleene-star r^* only as short hand form for $r^+ \vee \varepsilon$, the term "variable-*plus* free" seems more appropriate. We nevertheless use the term "star free", since it is much more common in the literature on regular expressions and languages.

Our lower bound of Theorem 3.7 shows that even single-edge queries with xregex have PSpace-complete data-complexity. However, the reduction of Theorem 3.7 needs an xregex that has a variable reference under a star, and Theorem 5.1 shows that restricting xregex accordingly (i. e., to require the xregex to be vstar-free) is actually sufficient to avoid such a high data complexity, even for queries with arbitrary graph patterns. Consequently, the data complexity of CXRPQ is as good as that of conjunctive regular path queries, i. e., adding variables to CRPQ as done in Section 4 does not increase the data complexity of CRPQ.

The combined complexity of CXRPQ, on the other hand, is problematic, since it is at least PSpace-hard. This justifies to search for further restrictions of CXRPQ that yield fragments with lower combined complexity (see Sections 6 and 7), for which again the lower bound reduction of Theorem 5.1 provides some guidance.

5.1 Lower Bounds

Proving the lower bounds of Theorem 5.1 is much simpler than the upper bounds. In fact, we can prove the following slightly stronger lower bounds:

Theorem 5.2. CXRPQ-*Bool-Eval is* PSpace-*hard in combined complexity, even if restricted to inputs where* $|\Sigma| = 3$, $\mathcal{X}_s = \{\mathsf{x}\}$, *and the query is a single-edge query.*

Proof. We can proceed similarly to the proof of Theorem 3.7. An instance $M_1, \ldots, M_k$ of the NFA-intersection problem over alphabet $\{\mathsf{a}, \mathsf{b}\}$ is transformed into a graph-database $\mathcal{D} = (V_{\mathcal{D}}, E_{\mathcal{D}})$ over alphabet $\Sigma = \{\mathsf{a}, \mathsf{b}, \#\}$ in the same way as in the proof of Theorem 3.7 and into a graph pattern $(x, \alpha^k_{\mathsf{ni}}, y)$ with

$$\alpha^k_{\mathsf{ni}} = \#\mathsf{x}\{(\mathsf{a} \vee \mathsf{b})^*\}\,(\#\#\,\mathsf{x})^{k-1}\#\#\#$$

(i. e., α^k_{ni} is obtained from α_{ni} by replacing $(\#\#\,\mathsf{x})^*$ by $k-1$ copies of $\#\#\,\mathsf{x}$). We note that α^k_{ni} is vstar-free, and therefore $(x, \alpha^k_{\mathsf{ni}}, y)$ is a single-edge CXRPQ. In particular, all the restrictions mentioned in the statement of the theorem are satisfied.

It follows analogously as in the proof of Theorem 3.7 that in $\mathcal{D}$ there is a path labelled with some $w \in \mathcal{L}(\alpha^k_{\mathsf{ni}})$ if and only if $\bigcap_{i=1}^{k} \mathcal{L}(M_i) \neq \emptyset$. □

It is a trivial observation, that we can check reachability between to given vertices in a given graph by a fixed single-edge CRPQ (and that a binary alphabet is sufficient for this reduction). Hence, we can directly conclude the following lower bound.

Theorem 5.3. CXRPQ-*Bool-Eval is* NL-*hard in data complexity, even if restricted to inputs where* $|\Sigma| = 2$ *and* $\mathcal{X}_s = \emptyset$, *and the query is a single-edge query.*

5.2 Upper Bounds

In this section, we prove the following result, which directly implies the upper bounds of Theorem 5.1.

Lemma 5.4. *Given a Boolean* $q \in$ CXRPQ *and a graph database* $\mathcal{D}$, *we can nondeterministically check whether* $\mathcal{D} \models q$ *in space* $\mathrm{O}(2^{poly(|q|)} \log(|\mathcal{D}|))$.

We first need the following definitions (also recall the definition of *vstar-freeness* given at the beginning of Section 4.2).

Definition 5.5. Let α be a valid xregex. A variable definition $\mathsf{x}\{\gamma\}$ of α is *basic*, if γ is a classical regular expression. The xregex α is

- *variable-alternation free* (*valt-free*) if, for every subexpression $(\beta_1 \vee \beta_2)$ of α, neither β_1 nor β_2 contain any variable definition or variable reference.
- *variable-simple* if it is vstar-free and valt-free.

Let us clarify these definitions with some intuitive explanations and examples. A variable definition is basic if it applies to a classical regular expression, like $\mathsf{x}\{\mathsf{a}^* \vee \mathsf{b}\}$ or $\mathsf{x}\{(\mathsf{a} \vee \mathsf{b} \vee \mathsf{c})^*\}$; on the other hand, variable definitions $\mathsf{x}\{\mathsf{ay}\}$ or $\mathsf{x}\{\mathsf{by}\{\mathsf{a}^*\}\}$ are not basic. The condition of being vstar-free or valt-free can also be interpreted as follows: an xregex is vstar-free (or valt-free), if every subtree of its syntax tree rooted by a node for a +-operation (for a $\vee$-operation, respectively) does not contain any nodes for variable definitions or references. If this is true for all +-operation nodes *and* all $\vee$-operation nodes, then the xregex is variable-simple. Equivalently, α is variable-simple if $\alpha = \beta_1\beta_2\ldots\beta_k$, where each β_i is a classical regular expression, a variable reference or a variable definition $\mathsf{x}\{\gamma\}$, where γ is also variable-simple.

Example 5.6.

$\mathsf{x}\{\mathsf{a}^*\}(\mathsf{bx}(\mathsf{c} \vee \mathsf{a}))^*\mathsf{b}$	is not vstar-free, but valt-free,
$\mathsf{x}\{\mathsf{a}^*\}\mathsf{y}((\mathsf{bx}) \vee (\mathsf{ca}))\mathsf{b}^*\mathsf{y}$	is vstar-free, but not valt-free,
$\mathsf{ax}\{(\mathsf{b} \vee \mathsf{c})^*\mathsf{by}\{\mathsf{dxa}^*\}\}\mathsf{bxa}^*\mathsf{z}\{\mathsf{d}^*\}\mathsf{zy}$	is variable-simple, but contains non-basic definitions,
$\mathsf{ax}\{(\mathsf{b} \vee \mathsf{c})^*\mathsf{da}\}\mathsf{bxa}^*\mathsf{y}\{\mathsf{c}^*\}\mathsf{xy}$	is variable-simple and only contains basic definitions.

We extend these restrictions to conjunctive xregex and CXRPQ in the obvious way, i. e., a conjunctive xregex $(\alpha_1, \alpha_2, \ldots, \alpha_m)$ is valt-free or variable-simple if each α_i with $i \in [m]$ is valt-free or variable-simple, respectively. Analogously, a CXRPQ is valt-free or variable-simple if its conjunctive xregex is valt-free or variable-simple, respectively.

Remark 5.7. *A* CXRPQ *has only basic variable definitions if and only if it has a depth of* 0 *in the sense as shall be defined in Section 6. The definition of depth is more general and not required here, so, for the sake of simplicity, we defer it to Section 6. However, since we shall also talk about depth-*0 *queries later on, we keep in mind that depth-*0 *queries are precisely the ones with only basic variable definitions.*

We are now ready to give a formal proof of Lemma 5.4.

Let $q \in$ CXRPQ be Boolean and represented by the graph pattern G_q with $E_{\mathcal{D}} = \{(x_i, \alpha_i, y_i) \mid i \in [m]\}$, and let $\mathcal{D}$ be a graph-database. We now define a nondeterministic procedure to check whether or not $\mathcal{D} \models q$.

In a first step, for every $i \in [m]$, we transform α_i by a nondeterministic procedure as follows. As long as α_i is not valt-free, we choose some subexpression $(\gamma_1 \vee \gamma_2)$ where γ_1 or γ_2 contains a variable definition or a variable reference, and we nondeterministically replace it by γ_1 or γ_2. In this way, we obtain a conjunctive xregex $\bar{\beta} = (\beta_1, \beta_2, \ldots, \beta_m)$ that is variable-simple and that satisfies that every $\mathsf{x} \in \mathcal{X}_s$ has at most one definition in $\bar{\beta}$ (this is due to the fact that, since $\bar{\beta}$ is variable-simple, every variable definition is necessarily instantiated by every tuple of ref-words, so two definitions of a variable would contradict the validity of $\bar{\beta}$). We note that $\bar{\beta}$ can be constructed in space $\mathrm{O}(|\bar{\alpha}|) = \mathrm{O}(|q|)$. The next proposition follows directly from the definitions.

Proposition 5.8. *Regardless of the actual nondeterministic choices, we have $\mathcal{L}(\bar{\beta}) \subseteq \mathcal{L}(\bar{\alpha})$. Furthermore, for every $\bar{w} \in \mathcal{L}(\bar{\alpha})$, it is possible to perform the nondeterministic steps in such a way that also $\bar{w} \in \mathcal{L}(\bar{\beta})$.*

Let q' be obtained from q by replacing each α_i by β_i. Proposition 5.8 means that we can proceed with q' instead of q, i. e., it remains to check whether $\mathcal{D} \models q'$.

In a next step, we will transform $\bar{\beta}$ into an equivalent $\bar{\gamma}$. First, we define a general modification step, which can be applied to any variable definition in $\bar{\beta}$. This modification step will then be used in order to transform $\bar{\beta}$ into $\bar{\gamma}$.

Main modification step: Let $\mathsf{z}\{\psi\}$ be a variable definition of $\bar{\beta}$. Since ψ is variable-simple, $\psi = \psi_1\psi_2\ldots\psi_p$ such that each ψ_ℓ with $\ell \in [p]$ is either a classical regular expression, a variable definition, or a variable reference.

For every $\ell \in [p]$, we define a ψ'_ℓ as follows. If ψ_ℓ is a variable definition $\mathsf{y}_\ell\{\ldots\}$, then we set $\psi'_\ell = \psi_\ell$. If ψ_ℓ is a classical regular expression or a variable reference, then we set $\psi'_\ell = \mathsf{y}_\ell\{\psi_\ell\}$ for a *new* variable $\mathsf{y}_\ell \notin \mathcal{X}_s$. Note that

$$\psi'_1\psi'_2\ldots\psi'_p = \mathsf{y}_1\{\ldots\}\mathsf{y}_2\{\ldots\}\ldots\mathsf{y}_p\{\ldots\}$$

is a concatenation of variable definitions. Next, we replace the variable definition $\mathsf{z}\{\psi\}$ in $\bar{\beta}$ by $\psi'_1\psi'_2 \dots \psi'_p$. Then, we replace all variable references of z in $\bar{\beta}$ by $\mathsf{y}_1\mathsf{y}_2 \dots \mathsf{y}_p$.

Let the thus modified version of $\bar{\beta}$ be denoted by $\bar{\beta}' = (\beta'_1, \beta'_2, \dots, \beta'_m)$. It can be easily seen that $\bar{\beta}'$ is valid and therefore a conjunctive xregex. Moreover, for every $i \in [m]$, β'_i is still variable-simple.

Lemma 5.9. $\mathcal{L}(\bar{\beta}) = \mathcal{L}(\bar{\beta}')$.

Proof. The difficulty in showing $\mathcal{L}(\bar{\beta}) = \mathcal{L}(\bar{\beta}')$ is due to the fact that we cannot assume, for every $i \in [m]$, that $\mathcal{L}_{\mathrm{ref}}(\beta_i) = \mathcal{L}_{\mathrm{ref}}(\beta'_i)$; i. e., we cannot conveniently apply Lemma 4.4. We need a few more notational preliminaries. We assume that $r \in [m]$ is such that the modified variable definition $\mathsf{z}\{\psi\}$ is in β_r. Moreover, recall that β'_r is obtained from β_r by the construction from above (i. e., β'_r is obtained from β_r by replacing the definition $\mathsf{z}\{\psi\}$ with $\psi'_1\psi'_2 \dots \psi'_p$ and all references z with $\mathsf{y}_1\mathsf{y}_2 \dots \mathsf{y}_p$.

Let $\bar{w} = (w_1, w_2, \dots, w_m)$ be a conjunctive match for $\bar{\beta}$, i. e., there is a ref-word

$$v_\beta = v_{\beta,1}\#v_{\beta,2}\# \dots \#v_{\beta,m} \in \mathcal{L}_{\mathrm{ref}}(\beta_1\#\beta_2\# \dots \#\beta_m)$$

such that $\mathrm{deref}(v_\beta) = w_1\#w_2\# \dots \#w_m$. In order to show that $\bar{w}$ is a conjunctive match for $\bar{\beta}'$, we have to show that there is a ref-word

$$v_{\beta'} = v_{\beta',1}\#v_{\beta',2}\# \dots \#v_{\beta',m} \in \mathcal{L}_{\mathrm{ref}}(\beta'_1\#\beta'_2\# \dots \#\beta'_m)$$

such that $\mathrm{deref}(v_{\beta'}) = w_1\#w_2\# \dots \#w_m$.

Since $\bar{\beta}$ is variable-simple, the variable definition $\mathsf{z}\{\psi\} = \mathsf{z}\{\psi_1\psi_2 \dots \psi_p\}$ (i. e., the variable definition that has been modified by the construction) must be instantiated by $v_{\beta,r}$, i. e., $v_{\beta,r} \in \mathcal{L}_{\mathrm{ref}}(\beta_r)$ and $v_{\beta,r}$ contains a factor ${}^{\mathsf{z}}{\triangleright}\, g_1g_2 \dots g_p {\triangleleft}^{\mathsf{z}}$, where, for every $\ell \in [p]$, $g_\ell \in \mathcal{L}_{\mathrm{ref}}(\psi_i)$. We let $v_{\beta',r}$ be obtained from $v_{\beta,r}$ by replacing ${}^{\mathsf{z}}{\triangleright}\, g_1g_2 \dots g_p {\triangleleft}^{\mathsf{z}}$ by ${}^{\mathsf{y}_1}{\triangleright}\, g_1 {\triangleleft}^{\mathsf{y}_1}\ {}^{\mathsf{y}_2}{\triangleright}\, g_2 {\triangleleft}^{\mathsf{y}_2} \dots {}^{\mathsf{y}_p}{\triangleright}\, g_p {\triangleleft}^{\mathsf{y}_p}$ and all occurrences of z by $\mathsf{y}_1\mathsf{y}_2 \dots \mathsf{y}_p$. By construction, $v_{\beta',r} \in \mathcal{L}_{\mathrm{ref}}(\beta'_r)$ is satisfied. For every $i \in [m] \setminus \{r\}$, we let $v_{\beta',i}$ be obtained from $v_{\beta,i}$ by replacing each variable-reference z by $\mathsf{y}_1\mathsf{y}_2 \dots \mathsf{y}_p$. By construction, it follows that $v_{\beta',i} \in \mathcal{L}_{\mathrm{ref}}(\beta'_i)$. It remains to show that $\mathrm{deref}(v_{\beta'}) = w_1\#w_2\# \dots \#w_m$.

By construction, the factor $v_{\beta,r}$ of v_β contains the definition ${}^{\mathsf{z}}{\triangleright}\, g_1g_2 \dots g_p {\triangleleft}^{\mathsf{z}}$, and $v_{\beta'}$ is obtained from v_β by replacing ${}^{\mathsf{z}}{\triangleright}\, g_1g_2 \dots g_p {\triangleleft}^{\mathsf{z}}$ with the definitions ${}^{\mathsf{y}_1}{\triangleright}\, g_1 {\triangleleft}^{\mathsf{y}_1}\ {}^{\mathsf{y}_2}{\triangleright}\, g_2 {\triangleleft}^{\mathsf{y}_2} \dots {}^{\mathsf{y}_p}{\triangleright}\, g_p {\triangleleft}^{\mathsf{y}_p}$, and every reference z with the references $\mathsf{y}_1\mathsf{y}_2 \dots \mathsf{y}_p$. We can now argue with the definition of the function deref, i. e., we consider the run of deref on v_β and the run of deref on $v_{\beta'}$ and show that these two runs produce the same word over Σ^*, i. e., $\mathrm{deref}(v_\beta) = \mathrm{deref}(v_{\beta'})$. Since $\mathrm{deref}(v_\beta) = w_1\#w_2\# \dots \#w_m$, this will also mean that $\mathrm{deref}(v_{\beta'}) = w_1\#w_2\# \dots \#w_m$.

First, the function deref applied to v_β and $v_{\beta'}$ will make identical replacements up to the point where ${}^{\mathsf{z}}{\triangleright}\, g'_1g'_2 \dots g'_p {\triangleleft}^{\mathsf{z}}$ is a factor of v_β, and ${}^{\mathsf{y}_1}{\triangleright}\, g'_1 {\triangleleft}^{\mathsf{y}_1}\ {}^{\mathsf{y}_2}{\triangleright}\, g'_2 {\triangleleft}^{\mathsf{y}_2} \dots {}^{\mathsf{y}_p}{\triangleright}\, g'_p {\triangleleft}^{\mathsf{y}_p}$ is a factor of $v_{\beta'}$, where $g'_1g'_2 \dots g'_p \in \Sigma^*$. This means that after replacing all references of z in v_β by $g'_1g'_2 \dots g'_p$ and replacing all references of $\mathsf{y}_1, \mathsf{y}_2, \dots, \mathsf{y}_p$ by $g'_1, g'_2, \dots, g'_p$, respectively, we obtain the same words in the run of deref on v_β and $v_{\beta'}$. This shows that every conjunctive match for $\bar{\beta}$ is also a conjunctive match for $\bar{\beta}'$. Moreover, the reverse direction, i. e., showing that every conjunctive match for $\bar{\beta}'$ is also a conjunctive match for $\bar{\beta}$, can be done analogously. More precisely, in order to construct suitable ref-words, we obtain v_β from $v_{\beta'}$ by replacing every $\mathsf{y}_1\mathsf{y}_2 \dots \mathsf{y}_p$ (observe that every reference of some y_ℓ in $v_{\beta'}$ must occur in this form) by z and every ${}^{\mathsf{y}_1}{\triangleright}\, g_1 {\triangleleft}^{\mathsf{y}_2}\ {}^{\mathsf{y}_2}{\triangleright}\, g_2 {\triangleleft}^{\mathsf{y}_2} \dots {}^{\mathsf{y}_p}{\triangleright}\, g_p {\triangleleft}^{\mathsf{y}_p}$ (observe that every definition of some y_ℓ in $v_{\beta'}$ must occur in this form) by ${}^{\mathsf{z}}{\triangleright}\, g_1g_2 \dots g_p {\triangleleft}^{\mathsf{z}}$. The rest of the argument is analogous.

Consequently, $\mathcal{L}(\bar{\beta}) = \mathcal{L}(\bar{\beta}')$, which concludes the proof. ◻

Next (and also as part of the main modification step), for every $\ell \in [p]$ where $\gamma_\ell = \mathsf{u} \in \mathcal{X}_s$ is a single variable reference, we replace $\mathsf{y}_\ell\{\mathsf{u}\}$ and all occurrences of variable reference y_ℓ by u. We denote the thus modified version of $\bar{\beta}'$ by $\bar{\beta}''$.

Lemma 5.10. $\mathcal{L}(\bar{\beta}') = \mathcal{L}(\bar{\beta}'')$.

Proof. Since $\bar{\beta}'$ is variable-simple, every variable $\mathsf{x} \in \mathcal{X}_s$ with a definition in $\bar{\beta}'$ has *exactly* one definition in every ref-word $v \in \mathcal{L}_{\mathrm{ref}}(\beta'_1\#\beta'_2\# \ldots \#\beta'_m)$. Consequently, any definition $\mathsf{y}_\ell\{\mathsf{u}\}$ will necessarily yield the factor ${}^{\mathsf{y}_\ell}\triangleright\, \mathsf{u} \triangleleft^{\mathsf{y}_\ell}$ in every ref-word $v \in \mathcal{L}_{\mathrm{ref}}(\beta'_1\#\beta'_2\# \ldots \#\beta'_m)$ and therefore replacing $\mathsf{y}_\ell\{\mathsf{u}\}$ and all variable references y_ℓ directly by u does not change the set of conjunctive matches of $\bar{\beta}'$. □

Remark 5.11. *Instead of applying this second part of the modification step, we could also change the first part such that we directly leave all $\mathsf{y}_\ell = \mathsf{u} \in \mathcal{X}_s$ unchanged and use u in the place of y_ℓ when substituting the references z by $\mathsf{y}_1\mathsf{y}_2 \ldots \mathsf{y}_p$. This, however, would unnecessarily complicate the proof of Lemma 5.9.*

Applying the main modification step to make all variable definitions basic: We now inductively apply the main modification step to each non-basic variable definition of $\bar{\beta}$. Each such application of the main modification step replaces a non-basic variable definiton $\mathsf{z}\{\psi\}$ by a sequence $\psi'_1\psi'_2 \ldots \psi'_p$ of variable references and variable definitions, such that each variable definition is either basic or it is already a non-basic variable definition of the original conjunctive xregex $\bar{\beta}$. Hence, after applying the main modification step with respect to all original non-basic variable definitions of $\bar{\beta}$, we obtain a conjunctive xregex $\bar{\gamma}$ with only basic variable definitions.

Lemma 5.12. $\mathcal{L}(\bar{\beta}) = \mathcal{L}(\bar{\gamma})$ *and* $|\bar{\gamma}| = \mathrm{O}(|\bar{\beta}|^{|\mathcal{X}_s|+1})$.

Proof. It directly follows from the correctness of the main modification step (Lemmas 5.9 and 5.10) that $\mathcal{L}(\bar{\beta}) = \mathcal{L}(\bar{\gamma})$.

We next estimate the size of $\bar{\gamma}$. In the procedure that transforms $\bar{\beta}$ into $\bar{\gamma}$, there are $k \leq |\mathcal{X}_s|$ applications of the main modification step (note that since $\bar{\beta}$ is variable-simple, every variable can have at most one definition). For every $i \in [k] \cup \{0\}$, let $\bar{\beta}^{(i)}$ be the version of $\bar{\beta}$ after the i^{th} application of the main modification step in the procedure that transforms $\bar{\beta}$ into $\bar{\gamma}$. In particular, $\bar{\beta}^{(0)} = \bar{\beta}$ and $\bar{\beta}^{(k)} = \bar{\gamma}$. For every $\mathsf{x} \in \mathcal{X}_s$, let k_{x} be the number of references of x in $\bar{\beta}$. We claim that, for every $i \in [k] \cup \{0\}$, $|\bar{\beta}^{(i)}| = \mathrm{O}(|\bar{\beta}|^{i+1})$. For $i = 0$, this obviously holds. Now let $i \in [k-1] \cup \{0\}$ and assume that $|\bar{\beta}^{(i)}| = \mathrm{O}(|\bar{\beta}|^{i+1})$. Moreover, let the $(i+1)^{\mathrm{th}}$ application of the main modification step apply to variable x. This means that $\bar{\beta}^{(i+1)}$ is obtained in the $(i+1)^{\mathrm{th}}$ application of the main modification step by replacing k_{x} symbols in $\bar{\beta}^{(i)}$ by at most $|\bar{\beta}^{(i)}|$ symbols. Hence, $|\bar{\beta}^{(i+1)}| = \mathrm{O}(k_{\mathsf{x}}|\bar{\beta}^{(i)}|) = \mathrm{O}(|\bar{\beta}||\bar{\beta}^{(i)}|) = \mathrm{O}(|\bar{\beta}||\bar{\beta}|^{i+1}) = \mathrm{O}(|\bar{\beta}|^{i+2})$. Consequently, $|\bar{\gamma}| = \mathrm{O}(|\bar{\beta}|^{k+1}) = \mathrm{O}(|\bar{\beta}|^{|\mathcal{X}_s|+1})$. □

Finally, let q'' be obtained from q' by replacing each β_i by γ_i. We observe that, $|\bar{\gamma}| = \mathrm{O}(|\bar{\beta}|^{|\mathcal{X}_s|+1}) = \mathrm{O}(2^{\mathsf{poly}(|\bar{\alpha}|)})$. Since, $\bar{\gamma}$ has only basic variable definitions, we can use Lemma 8.4 (which will be proven in Section 8.2) to nondeterministically decide whether $\mathcal{D} \models q''$ in space $\mathrm{O}(|q''| \log(|\mathcal{D}|)) = \mathrm{O}(2^{\mathsf{poly}(|q|)} \log(|\mathcal{D}|))$. Consequently, this whole procedure decides nondeterministically whether or not $\mathcal{D} \models q$ in space $\mathrm{O}(2^{\mathsf{poly}(|q|)} \log(|\mathcal{D}|))$, which concludes the proof of Lemma 5.4.

6 CXRPQ WITH BOUNDED DEPTH

In this section, we consider a restriction of CXRPQ that leads to fragments with combined complexity upper bound of PSpace (i. e., fragments with the same combined complexity as relational algebra).

The only reason why the procedure of Lemma 5.4 only leads to a combined complexity upper bound of ExpSpace (instead of a PSpace upper bound that would match our lower bound) is that the modification that makes all variable definitions basic introduces a seemingly unavoidable exponential size blow-up. Hence, we should investigate this aspect in a bit more detail.

Let us first illustrate the situation by the following example:

$$\alpha = \mathsf{x}_1\{\mathsf{a}\}\mathsf{x}_2\{\mathsf{x}_1\mathsf{x}_1\}\mathsf{x}_3\{\mathsf{x}_2\mathsf{x}_2\}\mathsf{x}_4\{\mathsf{x}_3\mathsf{x}_3\} \ldots \mathsf{x}_n\{\mathsf{x}_{n-1}\mathsf{x}_{n-1}\}\,.$$

This xregex is variable-simple and if we apply the main modification step with respect to the definitions for the variables $\mathsf{x}_2, \mathsf{x}_3, \ldots, \mathsf{x}_n$, then each reference of x_2 is replaced by 2 variable references, each reference of x_4 by 4 variable references, each reference of x_5 by 8 variable references, and so on.

The crucial point here seems to be that if we replace single references of x in a definition of y by longer sequences of references, then we also enlarge the definition of y and therefore the sequences by which single references of y are to be replaced in a following step of the procedure. This obviously leads to an exponential size blow-up. We shall now define the *depth* of (conjunctive) xregex.

First, we recall the definition of the relation $\preceq_v$ on $\mathcal{X}_s$ for ref-words v. For any ref-word v, we define a graph $G_v = (\mathcal{X}_s, \{(\mathsf{x}, \mathsf{y}) \mid \mathsf{x} \preceq_v \mathsf{y}\})$, and we note that since $\preceq_v$ is acyclic, G_v is a directed acyclic graph (DAG). The *depth* of v (denoted by $\mathsf{depth}(v)$) is defined as the length of a longest path from a root to a sink in G_v. For a conjunctive xregex $\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m)$, we define its *depth* (denoted by $\mathsf{depth}(\bar{\alpha})$) as $\max\{\mathsf{depth}(v) \mid v \in \mathcal{L}_{\mathsf{ref}}(\alpha_1\alpha_2 \ldots \alpha_m)\}$.

Next, we prove the following stronger variant of Lemma 5.12.

Lemma 6.1. *Let $\bar{\beta} \in m\text{-}\mathsf{CXRE}_{\Sigma, \mathcal{X}_s}$ be variable simple. Then there is an equivalent $\bar{\gamma} \in m\text{-}\mathsf{CXRE}_{\Sigma, \mathcal{X}'_s}$ such that $\bar{\gamma}$ has only basic variable definitions. Moreover, $|\bar{\gamma}| = \mathrm{O}(|\bar{\beta}|^{\mathsf{depth}(\bar{\beta})})$.*

Proof. We transform $\bar{\beta}$ to $\bar{\gamma}$ in exactly the same way as done in Section 5.2, but we apply the applications of the main modification step in a particular order. Let $\preceq_{\bar{\beta}}$ be the binary relation on $\mathcal{X}_s$ defined by setting $\mathsf{x} \preceq_{\bar{\beta}} \mathsf{y}$ if in $\bar{\beta}$ there is a definition of y that contains a reference or a definition of x. Let $G_{\bar{\beta}}$ be the graph induced by $\preceq_{\bar{\beta}}$, i. e., $G_{\bar{\beta}} = (\mathcal{X}_s, \{(\mathsf{x}, \mathsf{y}) \mid \mathsf{x} \preceq_{\bar{\beta}} \mathsf{y}\})$. Since $\bar{\beta}$ is variable-simple, every variable definition of $\bar{\beta}$ is necessarily instantiated by every ref-word $v \in \mathcal{L}_{\mathsf{ref}}(\beta_1\beta_2 \ldots \beta_m)$. In particular, this also means that $G_{\bar{\beta}}$ is a DAG. We apply the main modification steps according to the structure of $G_{\bar{\beta}}$ as follows.

We repeatedly choose some root x of (the current version of) $G_{\bar{\beta}}$. If the definition of x is basic, then we will just remove the root x from $G_{\bar{\beta}}$ without modifying $\bar{\beta}$. If, on the other hand, the definition $\mathsf{x}\{\gamma\}$ is non-basic, then we apply the main modification step to $\mathsf{x}\{\gamma\}$ (which modifies $\bar{\beta}$), and then we remove the root x from $G_{\bar{\beta}}$. Note that in this procedure, $\mathsf{x}\{\gamma\}$ always refers to the definition of x in the current version of $\bar{\beta}$, which may have been changed by applications of the main modification steps, i. e., variable references in the original definition of x might have been replaced by concatenations of new variable references. Next, we show that $|\bar{\gamma}| = \mathrm{O}(|\bar{\beta}|^{\mathsf{depth}(\bar{\beta})})$.

For every fixed variable definition $\mathsf{x}\{\psi\}$ of $\bar{\beta}$ that is non-basic, the construction will repeatedly replace variable references in ψ by sequences of variable references of size $\mathrm{O}(|\bar{\beta}|)$. Then, the variable definition is replaced by a sequence of basic variable definitions, which does not cause a size-increase. Consequently, the size of the variable definition of x right before it will be replaced is bounded by $|\bar{\beta}|^{k+1}$, where k is the number of main modification steps that replace references in this definition of x. By definition of $G_{\bar{\beta}}$ and $\mathsf{depth}(\bar{\beta})$, we have that $k \leq \mathsf{depth}(\bar{\beta})$. This directly implies that, for every $i \in [m]$, $|\bar{\gamma}[i]| = \mathrm{O}(|\bar{\gamma}[i]|^{\mathsf{depth}(\bar{\beta})+1})$, and therefore $|\bar{\gamma}| = \mathrm{O}(|\bar{\beta}|^{\mathsf{depth}(\bar{\beta})+1})$. □

For a $q \in \mathsf{CXRPQ}_{\Sigma, \mathcal{X}_s}$ with conjunctive xregex $\bar{\alpha}$, we define $\mathsf{depth}(q) = \mathsf{depth}(\bar{\alpha})$. For every $k \in \mathbb{N}$, let $\mathsf{CXRPQ}^{\mathsf{depth} \leq k}_{\Sigma, \mathcal{X}_s}$ be the class of $q \in \mathsf{CXRPQ}_{\Sigma, \mathcal{X}_s}$ with $\mathsf{depth}(q) \leq k$.

If the depth of a query is 0, then it can only have basic variable definitions (see Remark 5.7) and therefore Lemma 8.4 (which will be proven in Section 8.2) directly implies the following result.

Theorem 6.2. $\mathsf{CXRPQ}^{\mathsf{depth} \leq 0}$*-Bool-Eval can be solved in nondeterministic space* $\mathrm{O}(|q| \log(|\mathcal{D}|))$.

Given a $q \in \mathsf{CXRPQ}^{\mathsf{depth} \leq k}$ with $k > 0$, we can proceed analogously as done on Section 5.2. We only have to note that the initial nondeterministic transformation does not increase the depth of q. Consequently, the application of the procedure of Section 5.2 yields a CXRPQ with a conjunctive xregex whose components are variable-simple, and that is of size polynomial in the initial query. Hence, for every fixed k, we obtain the following analogue of Lemma 5.4.

Lemma 6.3. *Let $k \in \mathbb{N}$ be a fixed constant. Given a Boolean $q \in \mathsf{CXRPQ}^{\mathsf{depth} \leq k}$ and a graph database $\mathcal{D}$, we can nondeterministically check whether $\mathcal{D} \models q$ in space* $\mathrm{O}(poly(|q|)\log(|\mathcal{D}|))$.

Since the vstar-free xregex used in order to show PSpace-hardness of CXRPQ-Bool-Eval, i. e., the xregex α^k_{ni} from the proof of Theorem 5.2, has only basic variable definitions and therefore a depth of 0, we can conclude the following result.

Theorem 6.4. *For every fixed constant $k \in \mathbb{N} \cup \{0\}$, $\mathsf{CXRPQ}^{\mathsf{depth} \leq k}$-Bool-Eval is* PSpace*-complete in combined complexity.*

For every $k \in \mathbb{N}$, let $\mathsf{CXRPQ}^{\mathsf{var} \leq k}_{\Sigma}$ be the class of conjunctive xregex path queries with at most k variables, i. e., $\mathsf{CXRPQ}^{\mathsf{var} \leq k}_{\Sigma} = \bigcup_{|\mathcal{X}_s| \leq k} \mathsf{CXRPQ}_{\Sigma, \mathcal{X}_s}$. Since, for every $q \in \mathsf{CXRPQ}^{\mathsf{var} \leq k}$, we have $\mathsf{depth}(q) \leq k - 1$, and since the xregex α^k_{ni} from the proof of Theorem 5.2 has only one variable, we can conclude the following corollary from Theorem 6.4.

Corollary 6.5. *For every fixed constant $k \in \mathbb{N}$, $\mathsf{CXRPQ}^{\mathsf{var} \leq k}$-Bool-Eval is* PSpace*-complete in combined complexity.*

Note that CXRPQs with no variables are classical CRPQs, for which Boolean evaluation is NP-complete in combined complexity.

7 CXRPQ WITH BOUNDED IMAGE SIZE

In this section, we consider another restriction of CXRPQ that leads to fragments with a better combined complexity upper bound, this time even of NP (i. e., fragments with the same combined complexity as CRPQ). The idea is to restrict the size of the words that can be referenced by variable references.

7.1 Conjunctive Xregex with Fixed Images

We recall the definition of the function deref (see Definition 3.3) that computes $\mathsf{deref}(w)$ for a ref-word w and note that it also uniquely allocates a subword v_{x} of $\mathsf{deref}(w)$ to each variable x that has a definition in w (i. e., the subwords v_{x} defined in the iterations of Step 2a of Definition 3.3). In this way, a ref-word w over terminal alphabet Σ and variables $\mathcal{X}_s$ describes a *variable mapping* $\mathsf{vmap}_{w, \mathcal{X}_s} : \mathcal{X}_s \to \Sigma^*$, i. e., we set $\mathsf{vmap}_{w, \mathcal{X}_s}(x) = v_{\mathsf{x}}$ if x has a definition in w and we set $\mathsf{vmap}_{w, \mathcal{X}_s}(x) = \varepsilon$ otherwise. The elements $\mathsf{vmap}_{w, \mathcal{X}_s}(x)$ for $x \in \mathcal{X}_s$ are called *variable images*. See also Remark 4.3 of Section 4.1 for explanations and examples concerning the variable mapping.

For every $\alpha \in \mathsf{XRE}_{\Sigma, \mathcal{X}_s}$ with $\mathcal{X}_s = \{\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n\}$ and $\bar{v} = (v_1, v_2, \ldots, v_n) \in (\Sigma^*)^n$, we define

$$\mathcal{L}^{\bar{v}}_{\mathsf{ref}}(\alpha) = \{u \in \mathcal{L}_{\mathsf{ref}}(\alpha) \mid \forall i \in [n] : \mathsf{vmap}_{u, \mathcal{X}_s}(\mathsf{x}_i) = v_i\},$$
$$\mathcal{L}^{\bar{v}}(\alpha) = \mathsf{deref}(\mathcal{L}^{\bar{v}}_{\mathsf{ref}}(\alpha)).$$

The notion $\mathcal{L}^{\bar{v}}(\alpha)$ also extends to conjunctive xregex in the following way. Let $\bar{\alpha} \in m\text{-}\mathsf{CXRE}_{\Sigma, \mathcal{X}_s}$ with $\mathcal{X}_s = \{\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n\}$ and $\bar{v} = (v_1, v_2, \ldots, v_n) \in (\Sigma^*)^n$. A tuple $\bar{w} = (w_1, w_2, \ldots, w_m) \in (\Sigma^*)^m$ is a (*conjunctive*) *$\bar{v}$-match* for $\bar{\alpha} \in (\mathsf{XRE}_{\Sigma, \mathcal{X}_s})^m$ if $w_1 \# w_2 \# \ldots \# w_m \in \mathcal{L}^{\bar{v}}(\alpha_1 \# \alpha_2 \# \ldots \# \alpha_m)$. By $\mathcal{L}^{\bar{v}}(\bar{\alpha})$, we denote the set of $\bar{v}$-matches for $\bar{\alpha}$.

We now give an informal description of the restricted classes of CXRPQ and the corresponding evaluation algorithm. The main observation is that if we have fixed some variable mapping $\bar{v} = (v_1, v_2, \ldots, v_m)$, then the subset $\mathcal{L}^{\bar{v}}(\bar{\alpha})$ of $\mathcal{L}(\bar{\alpha})$ can be represented by a conjunctive xregex without variables, i. e., a tuple of classical regular expressions (see Lemma 7.1). However, the corresponding procedure is not as simple as "replace each $\mathsf{x}_i\{\ldots\}$ and x_i by v_i". We shall now illustrate this with an example and some intuitive explanations. Let $\alpha = (\alpha_1, \alpha_2) \in \mathsf{CXRE}_{\Sigma, \mathcal{X}_s}$

be defined by

$$\alpha_1 = \mathsf{x}_3\{\mathsf{x}_1\{\mathsf{ca}^*\mathsf{c}\}\mathsf{x}_2^*\} \vee [\,(\mathsf{x}_1\{\mathsf{cb}^*\} \vee \mathsf{x}_1\{\mathsf{x}_4\mathsf{c}^*\})(\mathsf{b} \vee \mathsf{x}_2^*)\mathsf{x}_3\{\mathsf{x}_1\mathsf{x}_2\mathsf{x}_1^*\}\,]\,,$$
$$\alpha_2 = (\mathsf{x}_1 \vee \mathsf{x}_2)^*\mathsf{x}_4\{(\mathsf{b} \vee \mathsf{c})^*\mathsf{x}_2^*\}\mathsf{x}_2\{(\mathsf{a} \vee \mathsf{b})^*\mathsf{a}\}\,,$$

and let $\bar{v} = (v_1, \ldots, v_4) = (\mathsf{ca}, \mathsf{a}, \mathsf{caaca}, \mathsf{ca})$.

For computing $\beta = (\beta_1, \beta_2) \in \mathsf{CXRE}_{\Sigma,\emptyset}$ with $\mathcal{L}(\bar{\beta}) = \mathcal{L}^{\bar{v}}(\bar{\alpha})$, replacing $\mathsf{x}_i\{\gamma\}$ by v_i can only be correct, if γ can produce v_i (in a match with variable mapping $\bar{v}$). So we should treat the variable definitions and references of γ as their intended images and then check whether the thus obtained classical regular expression can produce v_i. For example, we transform $\mathsf{x}_3\{\mathsf{x}_1\{\mathsf{ca}^*\mathsf{c}\}\mathsf{x}_2^*\}$ in α_1 to $\mathsf{x}_3\{v_1 v_2^*\} = \mathsf{x}_3\{\mathsf{caa}^*\}$ and check that $v_3 = \mathsf{ca} \in \mathcal{L}(\mathsf{caa}^*)$. However, this assumes that $\mathsf{x}_1\{\mathsf{ca}^*\mathsf{c}\}$ can really produce v_1, which is not the case, so we should rather remove $\mathsf{x}_3\{\mathsf{x}_1\{\mathsf{ca}^*\mathsf{c}\}\mathsf{x}_2^*\}$ altogether, since it can never be involved in a conjunctive match from $\mathcal{L}^{\bar{v}}(\bar{\alpha})$. Hence, we also need to cut whole alternation branches in $\bar{\alpha}$, and, moreover, we have to make sure that if $\mathsf{x}_i \neq \varepsilon$, then at least one definition of x_i is necessarily instantiated, while this is not required if $\mathsf{x}_i = \varepsilon$. Let us illustrate the correct transformations with the above example.

We pick an unmarked variable definition $\mathsf{x}_i\{\gamma\}$, where γ does not contain any unmarked variable definition (this means that initially we must choose definitions such that γ does not contain any other variable definition). Let γ' be obtained from γ by interpreting each variable definition and reference for x_j as v_j. If $v_i \in \mathcal{L}(\gamma')$, then we mark the variable definition $\mathsf{x}_i\{\gamma\}$, and $v_i \notin \mathcal{L}(\gamma')$, then we cut all branches that necesssarily produce this definition. This step is repeated until all variable derfinitions are marked or removed. With respect to our example (α_1, α_2), this means that we proceed as follows.

For α_1, we first observe that the definitions $\mathsf{x}_1\{\mathsf{ca}^*\mathsf{c}\}$ and $\mathsf{x}_1\{\mathsf{cb}^*\}$ should not be marked, but $\mathsf{x}_1\{\mathsf{x}_4\mathsf{c}^*\}$ should be marked. This leads to the following expression (markings are represented by overline):

$$\overline{\mathsf{x}_1\{\mathsf{x}_4\mathsf{c}^*\}}(\mathsf{b} \vee \mathsf{x}_2^*)\mathsf{x}_3\{\mathsf{x}_1\mathsf{x}_2\mathsf{x}_1^*\}\,.$$

Then, we investigate the remaining unmarked variable definition $\mathsf{x}_3\{\mathsf{x}_1\mathsf{x}_2\mathsf{x}_1^*\}$ and observe that it should be marked as well, which yields:

$$\overline{\mathsf{x}_1\{\mathsf{x}_4\mathsf{c}^*\}}(\mathsf{b} \vee \mathsf{x}_2^*)\overline{\mathsf{x}_3\{\mathsf{x}_1\mathsf{x}_2\mathsf{x}_1^*\}}\,.$$

With respect to α_2, we get

$$(\mathsf{x}_1 \vee \mathsf{x}_2)^*\overline{\mathsf{x}_4\{(\mathsf{b} \vee \mathsf{c})^*\mathsf{x}_2^*\}\mathsf{x}_2\{(\mathsf{a} \vee \mathsf{b})^*\mathsf{a}\}}\,.$$

Finally, we replace all definitions and references by the intended images to obtain

$$(\beta_1, \beta_2) = (\mathsf{ca}(\mathsf{b} \vee \mathsf{a}^*)\mathsf{caaca}, ((\mathsf{ca}) \vee \mathsf{a})^*\mathsf{caa})\,.$$

In the general case, the situation can be slightly more complicated, since we also have to make sure that every ref-word necessarily instantiates a definition for x_i if $v_i \neq \varepsilon$, while for $v_i = \varepsilon$ ref-words without definition for x_i should still be possible. It can also happen that this procedure reduces an xregex to $\varnothing$, which means that $\mathcal{L}^{\bar{v}}(\bar{\alpha}) = \emptyset$.

By turning these exemplary observations into a general procedure, we can prove the following result.

Lemma 7.1. *For every $\bar{\alpha} \in m\text{-}\mathsf{CXRE}_{\Sigma,\mathcal{X}_s}$ with $\mathcal{X}_s = \{\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n\}$ and every $\bar{v} \in (\Sigma^*)^n$, there is a $\bar{\beta} \in m\text{-}\mathsf{CXRE}_{\Sigma,\emptyset}$ such that $\mathcal{L}(\bar{\beta}) = \mathcal{L}^{\bar{v}}(\bar{\alpha})$. Furthermore, $|\bar{\beta}| = \mathrm{O}(|\bar{\alpha}|k)$, where $k = \max\{|\bar{v}[i]| \mid i \in [n]\}$, and $\bar{\beta}$ can be computed in time polynomial in $|\bar{\alpha}|$ and $|\bar{v}|$.*

Proof. Let $\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ and let $\bar{v} = (v_1, v_2, \ldots, v_n) \in \Sigma^*$. We give an algorithm that transforms $\bar{\alpha}$ into $\bar{\beta}$ with the desired property.

The general idea of the procedure is as follows. For every considered variable definition $\mathsf{x}_i\{\gamma\}$ in some α_j, we want to determine whether γ can produce v_i under the assumption that all variable references and variable definitions in γ can produce their corresponding images (according to $\bar{v}$). If this is not the case, then we delete the alternation branch that instantiates the variable definition $\mathsf{x}_i\{\gamma\}$ to make sure that it cannot be instantiated. When this procedure terminates, we can replace all remaining variable definitions and variable references by their corresponding images in order to obtain the components β_i of $\bar{\beta}$, which are then classical regular expressions. We now describe this procedure in more detail.

- **Step 1:** We assume that the variable definitions in $\bar{\alpha}$ can be either *marked* or *unmarked* and that they are initially all *unmarked*. The algorithm considers each variable definition separately in such an order that every considered variable definition does only contain other variable definitions (if any) that are already marked. We repeat the following step until all variable definitions are *marked* (recall that initially all variable definitions are *unmarked*). Let $\mathsf{x}_i\{\gamma\}$ be some *unmarked* variable definition in some α_j such that γ does not contain any *unmarked* variable definition. Let γ' be the classical regular expression obtained from γ by replacing each reference and definition for a variable $\mathsf{x}_{i'}$ by $v_{i'}$. If $v_i \in \mathcal{L}(\gamma')$, then we *mark* $\mathsf{x}_i\{\gamma\}$. If $v_i \notin \mathcal{L}(\gamma')$, then we have to modify α_j in such a way that $\mathsf{x}_i\{\gamma\}$ is never instantiated. This is achieved as follows. We start at the node of the syntax-tree of α_j that represents $\mathsf{x}_i\{\gamma\}$, we move up in the syntax tree and simply delete every node that we encounter (including the node that represents $\mathsf{x}_i\{\gamma\}$ where we started, which also means that the whole subtree rooted by this node is deleted), and we stop as soon as we encounter an alternation node, which is then replaced by its other child (i. e., the sibling of the node from which we entered the alternation node). In particular, if no alternation node is encountered, then we can conclude that the definition $\mathsf{x}_i\{\gamma\}$ under consideration will necessarily be instantiated by every ref-word of α_j, which immediately implies that there is no conjunctive match of $\bar{\alpha}$ with variable mapping $(v_1, v_2, \ldots, v_n)$. In this case, the procedure will replace α_j by $\varnothing$ (the unique regular expressions with $\mathcal{L}(\varnothing) = \emptyset$). Let $\bar{\alpha}' = (\alpha'_1, \alpha'_2, \ldots, \alpha'_m)$ be the conjunctive xregex obtained when the procedure of this step terminates.
- **Step 2:** Next, for every $i = 1, 2, \ldots, n$, if $v_i \neq \varepsilon$, then we have to modify $\bar{\alpha}'$ as follows. If, for some $j \in [m]$, α'_j contains a definition of x_i (note that this is possible for at most one α'_j), then we have to modify it such that it necessarily instantiates a definition for x_i, which is done as follows. For every node of the syntax tree for α'_j that corresponds to a definition for x_i, we mark it and then we move from this node up to the root and mark every visited node along the way. Next, for every marked alternation-node, we remove its unmarked child nodes (note that every such marked alternation-node has either one or two marked child nodes).
  In particular, we note that this modification is only necessary for $i \in [n]$ with $v_i \neq \varepsilon$, since ref-words that have no definition for x_i correspond to variable mappings with image ε for variable x_i, which should not be excluded if $v_i = \varepsilon$. Let $\bar{\alpha}'' = (\alpha''_1, \alpha''_2, \ldots, \alpha''_m)$ be the conjunctive xregex obtained when the procedure of this step terminates.

After these two modification steps, it is possible that, for some $i \in [m]$ with $v_i \neq \varepsilon$, there is no definition of x_i in $\bar{\alpha}''$. If this is the case, we replace each α''_j by $\varnothing$.

Finally, for every $i \in [n]$, we replace each definition and each occurrence of x_i by v_i in order to obtain a $\bar{\beta} \in m$-$\mathsf{CXRE}_{\Sigma,\emptyset}$. It can be verified with moderate effort that $\mathcal{L}(\bar{\beta}) = \mathcal{L}^{\bar{v}}(\bar{\alpha})$. Moreover, this procedure can obviously be carried out in time polynomial in $|\bar{\alpha}|$ and $|\bar{v}|$, and also $|\bar{\beta}| = O(|\bar{\alpha}|k)$, where $k = \max\{|\bar{v}[i]| \mid i \in [n]\}$. □

Remark 7.2. *We observe that the intuitive explanations and examples mentioned at the beginning of this subsection, as well as the statement of Lemma 7.1 and its proof do not require the xregex to be vstar-free. This explains why for the concepts and results of the following subsections we can consider the class* $\mathsf{CXRPQ}^{\mathsf{vs}}$ *(without the restriction of vstar-freeness), instead of only* CXRPQ.

7.2 CXRPQ$^{\text{vs}}$ with Bounded Image Size

We next lift the definition $\mathcal{L}^{\bar{v}}(\bar{\alpha})$ for conjunctive xregex $\bar{\alpha}$ to CXRPQ$^{\text{vs}}$ in the obvious way. Let $q \in \text{CXRPQ}^{\text{vs}}{}_{\Sigma,\mathcal{X}_s}$ with conjunctive xregex $\bar{\alpha} \in \text{CXRE}_{\Sigma,\mathcal{X}_s}$ and $\mathcal{X}_s = \{\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n\}$, let $\bar{v} \in (\Sigma^*)^n$, and let $\mathcal{D}$ be a graph database. By $q^{\bar{v}}(\mathcal{D})$ we denote the subset of $q(\mathcal{D})$ that contains those $q_h(\mathcal{D}) \in q(\mathcal{D})$, where h is a matching morphism with respect to some matching words $\bar{w} = (w_1, w_2, \ldots, w_m) \in \mathcal{L}^{\bar{v}}(\bar{\alpha})$. For Boolean $q \in \text{CXRPQ}^{\text{vs}}{}_{\Sigma,\mathcal{X}_s}$, we also write $\mathcal{D} \models^{\bar{v}} q$ to denote $() \in q^{\bar{v}}(\mathcal{D})$.

A direct consequence of Lemma 7.1 is that for every $\bar{v} \in (\Sigma^*)^n$ we can also transform any $q \in \text{CXRPQ}^{\text{vs}}$ with n variables into a $q' \in \text{CRPQ}$ with $q^{\bar{v}}(\mathcal{D}) = q'(\mathcal{D})$.

Lemma 7.3. *For every $q \in \text{CXRPQ}^{\text{vs}}$ with conjunctive xregex $\bar{\alpha} \in m\text{-CXRE}_{\Sigma,\mathcal{X}_s}$ with $\mathcal{X}_s = \{\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n\}$ and every $\bar{v} \in (\Sigma^*)^n$, there is a $q' \in \text{CRPQ}$, such that, for every database $\mathcal{D}$, we have that $q^{\bar{v}}(\mathcal{D}) = q'(\mathcal{D})$. Furthermore, $|q'| = \mathrm{O}(|q|k)$, where $k = \max\{|\bar{v}[i]| \mid i \in [n]\}$, and q' can be computed in time polynomial in $|q|$ and $|\bar{v}|$.*

Proof. According to Lemma 7.1, we can compute in time polynomial in $|\bar{\alpha}|$ and $|\bar{v}|$ (and therefore in time polynomial in $|q|$ and $|\bar{v}|$) a $\bar{\beta} \in m\text{-CXRE}_{\Sigma,\emptyset}$ such that $\mathcal{L}(\bar{\beta}) = \mathcal{L}^{\bar{v}}(\bar{\alpha})$. Thus, q' can be obtained from q by replacing each edge label α_i by β_i. In particular, we have $|\bar{\beta}| = \mathrm{O}(|\bar{\alpha}|k)$, where $k = \max\{|\bar{v}[i]| \mid i \in [n]\}$, and therefore also $|q'| = \mathrm{O}(|q|k)$. □

We slightly change the syntax of xregex by associating with every $\mathsf{x} \in \mathcal{X}_s$ a function $f_{\mathsf{x}} : \mathbb{N} \to \mathbb{N}$. This class of *xregex with bounded image size* is denoted by $\text{XRE}^{\text{bi}}_{\Sigma,\mathcal{X}_s}$. The idea is that when a CXRPQ$^{\text{vs}}$ q is evaluated over a graph database $\mathcal{D}$, then we are only interested in matching words for which each variable x is defined with a word of length at most $f_{\mathsf{x}}(|\,\mathcal{D}\,|)$. If f_{x} is a constant $k \in \mathbb{N}$, then we also write k instead of f_{x}. Let us now formally define the semantics of this modification.

For an $\alpha \in \text{XRE}^{\text{bi}}_{\Sigma,\mathcal{X}_s}$ with $\mathcal{X}_s = \{\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n\}$, and $k \in \mathbb{N}$, a tuple $\bar{v} \in (\Sigma^*)^n$ is a *valid tuple of variable images for α with respect to k*, if, for every $i \in [n]$, $|\bar{v}[i]| \leq f_{\mathsf{x}_i}(k)$. The *ref-language of α with respect to k* (denoted by $\mathcal{L}_{\text{ref}}(\alpha, k)$) is defined as the union of all $\mathcal{L}^{\bar{v}}_{\text{ref}}(\alpha)$, where $\bar{v}$ is a valid tuple of variable images for α with respect to k. The *language of α with respect to k* is defined by $\mathcal{L}(\alpha, k) = \text{deref}(\mathcal{L}_{\text{ref}}(\alpha, k))$.

The definition of xregex with bounded image size gives also rise to the class of conjunctive xregex with bounded image size, denoted by $\text{CXRE}^{\text{bi}}_{\Sigma,\mathcal{X}_s}$, and therefore the class of conjunctive xregex path queries with bounded image size, denoted by $\text{CXRPQ}^{\text{vs,bi}}_{\Sigma,\mathcal{X}_s}$. The semantics extend in a straightforward way: For an $\bar{\alpha} \in m\text{-CXRE}^{\text{bi}}_{\Sigma,\mathcal{X}_s}$ and $k \in \mathbb{N}$, a tuple $\bar{w} = (w_1, w_2, \ldots, w_m) \in (\Sigma^*)^m$ is a (*conjunctive*) *k-match* for $\bar{\alpha}$ if $w_1\#w_2\#\ldots\#w_m \in \mathcal{L}(\alpha_1\#\alpha_2\#\ldots\#\alpha_m, k)$. We denote the class of conjunctive k-matches by $\mathcal{L}(\bar{\alpha}, k)$.

For a $q \in \text{CXRPQ}^{\text{vs,bi}}_{\Sigma,\mathcal{X}_s}$ with conjunctive xregex with bounded image size $\bar{\alpha} \in \text{CXRE}^{\text{bi}}_{\Sigma,\mathcal{X}_s}$ with $\mathcal{X}_s = \{\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n\}$, and graph database $\mathcal{D}$, $q(\mathcal{D})$ is defined as the union of all $q^{\bar{v}}(\mathcal{D})$, where $\bar{v}$ is a valid tuple of variable images for $\bar{\alpha}$ with respect to k.

For every $k \in \mathbb{N}$, by $\text{CXRPQ}^{\text{vs,bi}\leq k}_{\Sigma,\mathcal{X}_s}$, we denote the class of $\text{CXRPQ}^{\text{vs,bi}}_{\Sigma,\mathcal{X}_s}$, where, for every $\mathsf{x} \in \mathcal{X}_s$, f_{x} is a constant $k' \leq k$; by $\text{CXRPQ}^{\text{vs,bi}\leq k\log}_{\Sigma,\mathcal{X}_s}$, we denote the class of $\text{CXRPQ}^{\text{vs,bi}}_{\Sigma,\mathcal{X}_s}$, where, for every $\mathsf{x} \in \mathcal{X}_s$, $f_{\mathsf{x}}(n)$ is a constant $k' \leq k$ or the function $k' \log(n)$ for some constant $k' \leq k$.

7.3 Evaluation of CXRPQ with Bounded Image Size

We are now ready to show that the combined complexity of evaluating CXRPQ$^{\text{vs}}$ with image size bounded by constants is in NP.

Theorem 7.4. *For every $k \in \mathbb{N}$, $\text{CXRPQ}^{\text{vs,bi}\leq k}$-Bool-Eval can nondeterministically be solved in polynomial time in combined complexity and in logarithmic space in data complexity.*

Proof. Let $q \in \text{CXRPQ}^{\text{vs, bi} \leq k}$ be Boolean with a conjunctive xregex with bounded image size $\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ over Σ and $\mathcal{X}_s = \{\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n\}$, and let $\mathcal{D}$ be a graph-database. We note that $\mathcal{D} \models q$ if and only if there is a valid tuple $\bar{v} = (v_1, v_2, \ldots, v_n)$ of variable images for $\bar{\alpha}$ with respect to $|\mathcal{D}|$ such that $\mathcal{D} \models^{\bar{v}} q$. Consequently, the following is a nondeterministic algorithm that checks whether $\mathcal{D} \models q$.

(1) Nondeterministically guess a valid tuple $\bar{v}$ of variable images for $\bar{\alpha}$ with respect to $|\mathcal{D}|$.
(2) Compute $q' \in \text{CRPQ}$ such that, for every database $\mathcal{D}'$, we have that $q^{\bar{v}}(\mathcal{D}') = q'(\mathcal{D}')$.
(3) Check whether $\mathcal{D} \models q'$.

Point 2 can be done according to Lemma 7.3, and Point 3 can be done according to Lemma 2.1. It remains to show that this nondeterministic algorithm requires polynomial time in combined complexity and logarithmic space in data complexity.

We first note that since f_{x} is a constant $k' \leq k$ for every $\mathsf{x} \in \mathcal{X}_s$, Point 1 can be done in time $O(nk) = O(n)$, which is polynomial in combined complexity. Moreover, the required space for this does only depend on q and k, which means that it is constant in data complexity.

According to Lemma 7.3, q' can be computed in time that is polynomial in $|q|$ and $|\bar{v}|$ in combined complexity, which, since k is a constant, is polynomial in $|q|$. Again, the required space for this does only depend on q and k, which means that it is constant in data complexity.

According to Lemma 2.1, we can nondeterministically check $\mathcal{D} \models q'$ in time polynomial in $|q'|$ and $|\mathcal{D}|$ in combined complexity. Moreover, according to Lemma 7.3, $|q'| = O(|q|k) = O(|q|)$, which means that we can nondeterministically check $\mathcal{D} \models q'$ in time polynomial in $|q|$ and $|\mathcal{D}|$, so polynomial in combined complexity. Finally, we also note that Lemma 2.1 implies that $\mathcal{D} \models q'$ can be checked in nondeterministic space that is logarithmic in $\mathcal{D}$ with respect to data complexity. □

By applying more or less the same algorithm, we can also show that for logarithmic size bounds, the combined complexity is still in NP, while the data complexity increases to poly-logarithmic space, more precisely, to nondeterministic space $O(\log^2(|\mathcal{D}|))$.

Theorem 7.5. *For every $k \in \mathbb{N}$, $\text{CXRPQ}^{\text{vs, bi} \leq k \log}$-Bool-Eval can nondeterministically be solved in polynomial time in combined complexity and in space $O(\log^2(|\mathcal{D}|))$ in data complexity.*

Proof. We can apply the same nondeterministic algorithm from the proof of Theorem 7.4:

(1) Nondeterministically guess a valid tuple $\bar{v}$ of variable images for $\bar{\alpha}$ with respect to $|\mathcal{D}|$.
(2) Compute $q' \in \text{CRPQ}$ such that, for every database $\mathcal{D}$, we have that $q^{\bar{v}}(\mathcal{D}) = q'(\mathcal{D})$.
(3) Check whether $\mathcal{D} \models q'$.

For every $\mathsf{x} \in \mathcal{X}_s$, we have $f_{\mathsf{x}} = k' \log(n)$ for some constant $k' \leq k$; thus, Point 1 can be done in time and space $O(n \log(|\mathcal{D}|))$. According to Lemma 7.3, q' can be computed in time that is polynomial in $|q|$ and $|\bar{v}|$, which, since $|\bar{v}| = O(n \log(|\mathcal{D}|))$, is polynomial in $|q|$ and $|\mathcal{D}|$. Moreover, according to Lemma 7.3, $|q'| = O(|q| \log(|\mathcal{D}|))$.

We can use Lemma 2.1 in order to conclude that we can nondeterministically check $\mathcal{D} \models q'$ in polynomial time in combined complexity. For data complexity, we observe that since $q' \in \text{CRPQ}$, we also have that q' is a CXRPQ with only basic variable definitions. Hence, we can conclude with Lemma 8.4 that we can nondeterministically check $\mathcal{D} \models q'$ in space $O(|q'| \log(|\mathcal{D}|)) = O(|q| \log^2(|\mathcal{D}|))$. □

Our algorithm is essentially a brute-force algorithm: it exhaustively tries all possible tuples of variable images that are within the given size bounds. Hence, the question arises whether we can achieve better upper bounds by some more sophisticated approach. However, we can answer this in the negative by complementing the upper bounds of Theorem 7.4 with matching lower bounds.

Theorem 7.6. *For every fixed constant $k \in \mathbb{N}$, $\text{CXRPQ}^{\text{vs, bi} \leq k}_{\Sigma, \mathcal{X}_s}$-Bool-Eval is*

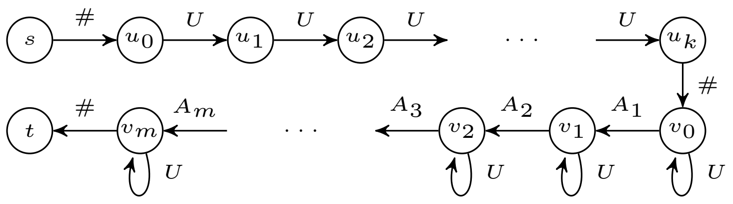


Fig. 5. Sketch of the Hitting Set Reduction from Theorem 7.6. An arc labelled with U or A_i stands for arcs labelled by $\langle z \rangle$ for every $z \in U$ or $z \in A_i$, respectively.

- NP-*hard in combined complexity, even for single-edge queries with only basic variable definitions and* $|\Sigma| = 3$,
- NL-*hard in data complexity, even for single-edge queries with conjunctive xregex* $\alpha \in \mathsf{RE}_\Sigma$ *and* $|\Sigma| = 2$.

Proof. The NL-hardness for data complexity follows in the same way as for Theorem 5.3.

In order to prove the NP-hardness for combined complexity, we devise a reduction from the problem Hitting Set, which is defined as follows. Given subsets $A_1, A_2, \ldots, A_m$ of some universe U and $k \in \mathbb{N}$, decide whether there is a *hitting set* of size at most k, i. e., a set $B \subseteq U$ with $|B| \leq k$ and $B \cap A_i \neq \emptyset$ for every $i \in [m]$.

Now let $A_1, A_2, \ldots, A_m \subseteq U = \{z_1, z_2, \ldots, z_n\}$ and $k \in \mathbb{N}$ be an instance of Hitting Set. Let $\Sigma = \{\mathsf{a}, \mathsf{b}, \#\}$ and, for every $z_i \in U$, we define $\langle z_i \rangle = \mathsf{b}\mathsf{a}^i\mathsf{b}$.

Next, we define a graph database $\mathcal{D} = (V_\mathcal{D}, E_\mathcal{D})$ over Σ as follows. For the sake of convenience, we also allow words as edge labels in graph databases, which stand for paths in the obvious way, i. e., for some $w \in \Sigma^*$, an arc (u, w, v) in a graph-database represents a path from u to v labelled with w (using some distinct intermediate nodes). We set

$$V_\mathcal{D} = \{s, u_0, u_1, \ldots, u_k, v_0, v_1, \ldots, v_m, t\}$$

and the set $E_\mathcal{D}$ is defined as follows.

- There are arcs $(s, \#, u_0)$, $(u_k, \#, v_0)$ and $(v_m, \#, t)$,
- For every $i \in [k]$ and $z \in U$ there is an arc $(u_{i-1}, \langle z \rangle, u_i)$,
- for every $i \in [m]$ and $z \in A_i$ there is an arc $(v_{i-1}, \langle z \rangle, v_i)$,
- for every $i \in [m] \cup \{0\}$ and every $z \in U$ there is an arc $(v_i, \langle z \rangle, v_i)$.

See Figure 5 for an illustration of $\mathcal{D}$. Next, we define the xregex

$$\alpha = \# \prod_{i=1}^{(n+2)k} \mathsf{x}_i\{\mathsf{a} \vee \mathsf{b} \vee \varepsilon\} \, \# \left(\prod_{i=1}^{(n+2)k} \mathsf{x}_i \right)^m \#,$$

and the Boolean $q \in \mathsf{CXRPQ}^{\mathsf{vs}, \mathsf{bi} \leq 1}$ is defined by the single edge graph pattern $\{\{x, y\}, (x, \alpha, y)\}$, where f_{x_i} is the constant 1 for every $i \in [(n+2)k]$. In order to prove the correctness, we first make some observations:

(1) Due to the occurrences of #, there is a path in $\mathcal{D}$ from some node u to some node v labelled with a word from $\mathcal{L}(\alpha)$ if and only if there is such a path in $\mathcal{D}$ from s to t.
(2) The language $\mathcal{L}(\alpha)$ contains exactly the words $w = \#w_1\#w_2\#$, where $w_1 \in \{\mathsf{a}, \mathsf{b}\}^*$ with $|w_1| \leq (n+2)k$, and $w_2 = (w_1)^m$.
(3) Every path in $\mathcal{D}$ from s to t is labelled by $w = \#w_1\#w_2\#$, where

$$w_1 = \langle z_{j_1} \rangle \langle z_{j_2} \rangle \ldots \langle z_{j_k} \rangle,$$

for some $\{j_1, j_2, \ldots, j_k\} \subseteq [n]$, and

$$w_2 = u_1 \langle z_{r_1} \rangle u_2 \langle z_{r_2} \rangle u_3 \ldots u_m \langle z_{r_m} \rangle u_{m+1},$$

such that $\{r_1, r_2, \ldots, r_m\} \subseteq [n]$ and, for every $i \in [m]$, $z_{r_i} \in A_i$; in particular, this means that $\{z_{r_1}, z_{r_2}, \ldots, z_{r_m}\}$ is a hitting set (with respect to the considered problem-instance).

Next, we assume that in $\mathcal{D}$ there is a path p labelled with a word $w \in \mathcal{L}(\alpha)$. With Point 1, we conclude that p is a path from s to t. Moreover, with Point 3, we have that $w = \#w_1\#w_2\#$, where $w_1 = \langle z_{j_1}\rangle\langle z_{j_2}\rangle \ldots \langle z_{j_k}\rangle$ and $w_2 = u_1\langle z_{r_1}\rangle u_2\langle z_{r_2}\rangle u_3 \ldots u_m\langle z_{r_m}\rangle u_{m+1}$, such that $\{z_{r_1}, z_{r_2}, \ldots, z_{r_m}\}$ is a hitting set. Finally, Point 2 means that $w_2 = (w_1)^m$, which directly implies that $\{z_{r_1}, z_{r_2}, \ldots, z_{r_m}\} \subseteq \{z_{j_1}, z_{j_2}, \ldots, z_{j_k}\}$ and therefore $|\{z_{r_1}, z_{r_2}, \ldots, z_{r_m}\}| \leq k$.

On the other hand, if $\{z_{j_1}, z_{j_2}, \ldots, z_{j_k}\}$ is a hitting set of size k, then we can construct a word $w \in \mathcal{L}(\alpha)$ and a path from s to t that is labelled with w as follows. We set $w = \#w_1\#w_2\#$ with $w_1 = \langle z_{j_1}\rangle\langle z_{j_2}\rangle \ldots \langle z_{j_k}\rangle$ and $w_2 = (w_1)^m$. We note that, according to Point 2, $w \in \mathcal{L}(\alpha)$ and we have to show that there is a path from s to t labelled with w. There is obviously a path from s to v_0 labelled with $\#w_1\#$. The set $\{z_{j_1}, z_{j_2}, \ldots, z_{j_k}\}$ is a hitting set and each of the m occurrences of factor w_1 in w_2 contains a factor $\langle z_{j_i}\rangle$ for every $i \in [k]$. Thus, w_2 can be factorised into $w_2 = u_1\langle z_{r_1}\rangle u_2\langle z_{r_2}\rangle u_3 \ldots u_m\langle z_{r_m}\rangle u_{m+1}$, such that, for every $i \in [m]$, $\langle z_{r_i}\rangle \in A_i$. This means that there is a path from v_0 to t labelled with $w_2\#$ and therefore a path from s to t labelled with w.

This proves that $\mathsf{CXRPQ}^{\mathsf{vs},\mathsf{bi}\leq 1}_{\Sigma,\mathcal{X}_s}$-Bool-Eval is NP-hard in combined complexity, even for single-edge queries with only basic variable definitions and $|\Sigma| = 3$. Since we can interpret the constructed query as a $\mathsf{CXRPQ}^{\mathsf{vs},\mathsf{bi}\leq k}_{\Sigma,\mathcal{X}_s}$ for any $k \in \mathbb{N}$, it follows that, for every $k \in \mathbb{N}$, $\mathsf{CXRPQ}^{\mathsf{vs},\mathsf{bi}\leq k}_{\Sigma,\mathcal{X}_s}$-Bool-Eval is NP-hard in combined complexity, even for single-edge queries with only basic variable definitions and $|\Sigma| = 3$. □

Remark 7.7. *Theorem 7.6 also points out a noteworthy difference between* $\mathsf{CXRPQ}^{\mathsf{vs},\mathsf{bi}\leq k}$ *and* CRPQ. *While in combined complexity* CRPQ *can be evaluated in polynomial-time if the underlying graph pattern is acyclic (see [5, 7, 10]),* $\mathsf{CXRPQ}^{\mathsf{vs},\mathsf{bi}\leq k}$ *evaluation remains* NP*-hard in combined complexity even for single-edge graph patterns (and* $k = 1$*).*

8 CXRPQ WITH REGULAR RELATIONS

Variable definitions and variable references of ref-words or xregex check equality between certain subwords. Similarly, they could also be interpreted as checking some other binary string relation, e. g., the prefix relation or subsequence relation. In this section, we shall investigate the question whether CXRPQ can be extended in this way (the formal definitions are provided in Section 8.1), but we restrict ourselves to so-called *regular relations*.[6]

Our main observation (Lemma 8.4 presented in Section 8.2) is that if CXRPQ have only basic variable definitions (or, equivalently, are of depth 0), then even with arbitrary regular relations instead of only equality relations, evaluation is PSpace-complete in combined complexity and NL-complete with respect to data complexity. In particular, we recall that we have used this result already in the proof of Lemma 5.4 (see Page 22), but for the special case of only equality relations; thus, by proving this, we also fill the gap of the proof of Lemma 5.4.

Finally, in Section 8.3, we revisit the fragments of CXRPQ with bounded image size presented in Section 7, and we show that for them regular relations can be added while staying within the same complexity bounds, i. e., the ones stated by Theorems 7.4 and 7.5. In particular, we do not need to require all variable definitions to be basic.

8.1 Adding Regular Relations to CXRPQ

We first need some general definitions.

Let $u, v \in \Sigma^*$, let $m = \max(|u|, |v|)$, and let $\perp$ be some symbol not in A. The *convolution* of u and v is defined by

$$u \otimes v = ((x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)),$$

[6]This decision is explained below in Section 8.1; see Remark 8.2.

where $m = \max\{|u|, |v|\}$, and, for every $i \in [m]$, $x_i = u[i]$ if $i \leq |u|$ and $x_i = \bot$ otherwise, and $y_i = v[i]$ if $i \leq |v|$ and $y_i = \bot$, otherwise. Note that the convolution is a word over the alphabet $(\Sigma \cup \{\bot\})^2$. For example, $\mathsf{abca} \otimes \mathsf{acacbab} = (\mathsf{a}, \mathsf{a})(\mathsf{b}, \mathsf{c})(\mathsf{c}, \mathsf{a})(\mathsf{a}, \mathsf{c})(\bot, \mathsf{b})(\bot, \mathsf{a})(\bot, \mathsf{b})$. A binary relation $\sqsubset$ over A^* is *regular* if the language $\{u \otimes v \mid u, v \in A^*, u \sqsubset v\}$ is a regular language over the alphabet $(A \cup \{\bot\})^2$.

With every $\mathsf{x} \in \mathcal{X}_s$, we associate a binary regular relation $\sqsubset_{\mathsf{x}}$ over Σ^*. We modify the definition of the deref-function (Definition 3.3) to the case of regular relations as follows:

Definition 8.1 (Deref-Function with Regular Relations). For a ref-word w over Σ and $\mathcal{X}_s$, $\mathsf{deref}_{\Sigma, \mathcal{X}_s}(w) \subseteq \Sigma^*$ contains exactly the words that can be obtained from w by the following procedure:

(1) Remove all occurrences of $\mathsf{x} \in \mathcal{X}_s$ without definition in w.
(2) Repeat until we have obtained a word over Σ:
    (a) Let ${}^{\mathsf{x}}\!\triangleright\, v_{\mathsf{x}} \triangleleft^{\mathsf{x}}$ be a definition such that $v_{\mathsf{x}} \in \Sigma^*$.
    (b) Replace ${}^{\mathsf{x}}\!\triangleright\, v_{\mathsf{x}} \triangleleft^{\mathsf{x}}$ by v_{x}, and replace each occurrences of x in w by some word u_{x} with $v_{\mathsf{x}} \sqsubset_{\mathsf{x}} u_{\mathsf{x}}$.

We note that in contrast to the case where every $\sqsubset_{\mathsf{x}}$ is the equality relation, $\mathsf{deref}_{\Sigma, \mathcal{X}_s}(w)$ is not a unique word, but a set of words. The language described by a valid $\alpha \in \mathsf{XRE}_{\Sigma, \mathcal{X}_s}$ can nevertheless be defined analogously, i. e.,

$$\mathcal{L}(\alpha) = \bigcup_{w \in \mathcal{L}_{\mathsf{ref}}(\alpha)} \mathsf{deref}_{\Sigma, \mathcal{X}_s}(w)\,.$$

As a special case, we also define $\mathcal{L}(\varnothing) = \emptyset$.

Remark 8.2. *The restriction to regular relations is a reasonable choice, since surprisingly simple non-regular relations can make evaluation problems undecidable. For example, in [7] it is shown that* ECRPQ *evaluation becomes undecidable if we allow rational relations (i. e., relations that can be defined by non-deterministic transducers) instead of only regular relations.*

With respect to CXRPQ*, we can show a slightly weaker result: if we allow* alphabet reduction relations *and the* palindrome relation *(to be explained below), then* CXRPQ-*Bool-Eval is already undecidable.*

For an alphabet Σ, let the *palindrome* relation be defined by $u \sqsubset_P v$ if $u = v = w(w)^R$ for some $w \in \Sigma^*$; for any $\Delta \subseteq \Sigma$, let the Δ*-restriction relation* $\sqsubset_\Delta$ be defined by $u \sqsubset_\Delta v$ if u restricted to alphabet Δ equals v. A relation is called an *alphabet restriction relation* if it is the Δ-restriction relation for some $\Delta \subseteq \Sigma$.

We note that alphabet restriction relations are not regular, but they are *rational relations* (i. e., relations that can be defined by nondeterministic transducers (e. g., see [9])). The palindrome relation is not even rational, but context-free, i. e., it can be defined by a pushdown transducer.

Theorem 8.3. CXRPQ-*Bool-Eval with only alphabet reduction relations and the palindrome relation is undecidable.*

This result can be proven by a reduction from the post correspondence problem that is similar to the one used in [7]. Full proof details are given in the Appendix.

8.2 Evaluating CXRPQ with Regular Relations and Basic Variable Definitions

In this section, we show that CXRPQ with regular relations and only basic variable definitions can nondeterministically be evaluated in space $\mathrm{O}(|q| \log(|\mathcal{D}|))$. We recall that we have already used this result (in the case of only equality relations) in the proof of Lemma 5.4 (see Page 22).

Lemma 8.4. *Given a set $\mathcal{X}_s$ of variables (with relations $\sqsubset_{\mathsf{x}}$ for every $\mathsf{x} \in \mathcal{X}_s$ given as* NFA*s), a Boolean $q \in$* CXRPQ *with only basic variable definitions, and a graph-database $\mathcal{D}$, we can nondeterministically decide whether or not $\mathcal{D} \models q$ in space* $\mathrm{O}(|q| \log(|\mathcal{D}|))$.

Proof. For every $\mathsf{x} \in \mathcal{X}_s$, let R_{x} be the NFA that describes the relation $\sqsubset_{\mathsf{x}}$. Let q be defined by a graph pattern G_q with edge set $E_q = \{(x_i, \alpha_i, y_i) \mid i \in [m]\}$ and conjunctive xregex $\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m) \in \mathsf{CXRE}_{\Sigma, \mathcal{X}_s}$. We first nondeterministically transform q into a query q' with a variable-simple conjunctive xregex $\bar{\beta}$, by the procedure described before Proposition 5.8. Now, for every $i \in [m]$, $\beta_i = \beta_{i,1}\beta_{i,2}\ldots\beta_{i,t_i}$, where, for every $j \in [t_i]$, $\beta_{i,j}$ is a classical regular expression, a variable reference, or a variable definition $\mathsf{x}\{\psi\}$, where ψ is a classical regular expression.

We next modify $G_{q'}$ such that every (x_i, β_i, y_i) is replaced by edges $(z_{i,0}, \beta_{i,1}, z_{i,1}), (z_{i,1}, \beta_{i,2}, z_{i,2}), \ldots, (z_{i,t_i-1}, \beta_{i,t_i}, z_{i,t_i})$, where $z_{i,0} = x_i$ and $z_{i,t_i} = y_i$. We denote this modified CXRPQ by q'' and let $\bar{\gamma} = (\gamma_1, \ldots, \gamma_{m'})$ be its conjunctive xregex, i. e., $m' = \sum_{i=1}^{m} t_i$ and $(\gamma_1, \ldots, \gamma_{m'})$ equals

$$(\beta_{1,1}, \ldots, \beta_{1,t_1}, \beta_{2,1}, \ldots, \beta_{2,t_2}, \ldots, \beta_{m,1}, \ldots, \beta_{m,t_m}) .$$

It can be easily seen that $q' \equiv q''$.

We now show how to decide $\mathcal{D} \models q''$ for a given graph database $\mathcal{D}$.

For every $i \in [m']$, we define an NFA M_i with state-set Q_i and transition function δ_i as follows:

- If $\gamma_i \in \{\psi, \mathsf{x}\{\psi\}\}$ for some classical regular expression ψ, then M_i accepts $\mathcal{L}(\psi)$.
- If $\gamma_i = \mathsf{x}$ for some $\mathsf{x} \in \mathcal{X}_s$, then $M_i = R_{\mathsf{x}}$ (note that R_{x} accepts words over Σ^2).

For technical reasons, we first add an ε-labelled self-loop to every node in $\mathcal{D}$ and to every state in every M_i (or an $(\varepsilon, \varepsilon)$-labelled self-loop, if $M_i = R_{\mathsf{x}}$, for some $\mathsf{x} \in \mathcal{X}_s$). Next, we define a graph $G_{q'',\mathcal{D}} = (V_{q'',\mathcal{D}}, E_{q'',\mathcal{D}})$, where $V_{q'',\mathcal{D}} = (V_{\mathcal{D}})^{m'} \times Q_1 \times Q_2 \times \ldots \times Q_{m'}$ and $E_{q'',\mathcal{D}}$ contains an edge

$$((u_1, \ldots, u_{m'}, p_1, \ldots, p_{m'}), (v_1, \ldots, v_{m'}, r_1, \ldots, r_{m'}))$$

if and only if there are $b_1, b_2, \ldots, b_{m'} \in \Sigma \cup \{\varepsilon\}$ such that,

- for every $i \in [m']$, $(u_i, b_i, v_i) \in E_{\mathcal{D}}$,
- for every $i \in [m']$ with $\gamma_i = \{\psi, \mathsf{x}\{\psi\}\}$ for some classical regular expression ψ, $r_i \in \delta_i(p_i, b_i)$, and
- for every $i, j \in [m']$ with $\gamma_i = \mathsf{x}$ and $\gamma_j = \mathsf{x}\{\psi\}$, for some $\mathsf{x} \in \mathcal{X}_s$ and a classical regular expression ψ, $r_i \in \delta_j(p_i, (b_j, b_i))$.

Let $\bar{u} = (u_1, \ldots, u_{m'}, p_1, \ldots, p_{m'})$ and $\bar{v} = (v_1, \ldots, v_{m'}, r_1, \ldots, r_{m'})$ be two vertices from $G_{q'',\mathcal{D}}$. We observe that in $G_{q'',\mathcal{D}}$ there is a path from $\bar{u}$ to $\bar{v}$ if and only if there is a tuple $(w_1, w_2, \ldots, w_{m'}) \in (\Sigma^*)^{m'}$ such that, for every $i \in [m']$, there is a w_i-labelled path from u_i to v_i in $\mathcal{D}$, a w_i-labelled path from p_i to r_i in M_i if $\gamma_i \in \{\psi, \mathsf{x}\{\psi\}\}$, and a $w_j \otimes w_i$ labelled path from p_i to r_i in M_i if $\gamma_i = \mathsf{x}$ and $\gamma_j = \mathsf{x}\{\psi\}$.

We say that a vertex $\bar{u}$ of $G_{q'',\mathcal{D}}$ is *initial*, if, for every $i \in [m']$, p_i is the initial state of M_i, and, for every $i, i' \in [m']$, $x_i = x_{i'}$ implies $u_i = u_{i'}$. Analogously, we say that a vertex $\bar{v}$ of $G_{q'',\mathcal{D}}$ is *final*, if, for every $i \in [m']$, r_i is the final state of M_i, and, for every $i, i' \in [m']$, $y_i = y_{i'}$ implies $v_i = v_{i'}$. With the observation from above, we can conclude that in $G_{q'',\mathcal{D}}$ there is a path from some initial vertex $\bar{u} = (u_1, \ldots, u_{m'}, p_1, \ldots, p_{m'})$ to some final vertex $\bar{v} = (v_1, \ldots, v_{m'}, r_1, \ldots, r_{m'})$ if and only if there is a tuple $(w_1, w_2, \ldots, w_{m'}) \in (\Sigma^*)^{m'}$ that is a conjunctive match of $\bar{\gamma}$ (this follows from the definition of the edge relation of $G_{q'',\mathcal{D}}$), and, for every $i \in [m]$, $\mathcal{D}$ contains a path from u_i to v_i labelled with w_i; note that this latter property means that $\mathcal{D} \models q''$ (this follows from the definition of initial and final vertices of $G_{q'',\mathcal{D}}$). For simplicity, we add two additional vertices s and t to $G_{q'',\mathcal{D}}$ with an edge $(s, \bar{u})$ and an edge $(\bar{v}, t)$ for every initial vertex $\bar{u}$ and final vertex $\bar{u}$. Now $\mathcal{D} \models q''$ if and only if there is a path from s to t in $G_{q'',\mathcal{D}}$.

It remains to describe how this can be done in the claimed time and space bound. To this end, we first observe that the initial replacement of variable definition $\mathsf{x}\{\mathsf{y}\}$ by y can obviously be carried out in space $\mathrm{O}(|q'|)$, and it does not increase the size of q'. Then, transforming $\bar{\beta}$ into $\bar{\gamma}$ can also be done in space $\mathrm{O}(|q''|)$, where $|q''| = \mathrm{O}(|q'|)$. Moreover, we can transform each γ_i into an NFA M_i of size $\mathrm{O}(|\gamma_i|)$ in space $\mathrm{O}(|\gamma_i|)$. Consequently, we can obtain all M_i in space $\mathrm{O}(|q''|)$. In particular, we note that removing ε-transitions may result in NFA of size $\mathrm{O}(|\gamma_i|^2)$, but our construction of $G_{q'',\mathcal{D}}$ from above can also handle possible ε-transitions of the M_i.

Now a vertex $\bar{u}$ from $G_{q'',\mathcal{D}}$ can be represented by $\mathrm{O}(m')$ pointers to $\mathcal{D}$ and $\mathrm{O}(1)$ pointers to each of the M_i. Moreover, evaluating the edge relation of $G_{q'',\mathcal{D}}$, i. e., checking for fixed vertices $\bar{u}, \bar{v}$ of $G_{q'',\mathcal{D}}$ whether there is an edge $(\bar{u}, \bar{v})$, can be done as follows. Checking whether there are $b_1, b_2, \ldots, b_{m'} \in \Sigma \cup \{\varepsilon\}$ such that, for every $i \in [m']$, $r_i \in \delta_i(p_i, b_i)$ and $(u_i, b_i, v_i) \in E_{\mathcal{D}}$, can be done by consulting the pointers that represent $\bar{u}$ and $\bar{v}$, the transition functions of the M_i and the edge-relation of $\mathcal{D}$. Moreover, checking for every $i, j \in [m']$ with $\gamma_i = \mathsf{x}$ and $\gamma_j = \mathsf{x}\{\psi\}$, for some $\mathsf{x} \in \mathcal{X}_s$ and a classical regular expression ψ, whether $r_i \in \delta_j(p_i, (b_j, b_i))$ can be done by using two pointers to q''. Consequently, we can nondeterministically check whether there is a path from s to t in $G_{q'',\mathcal{D}}$ in space $\mathrm{O}(|q''|(\log(|\mathcal{D}|) + \log(|q''|)) = \mathrm{O}(|q'|(\log(|\mathcal{D}|) + \log(|q'|)) = \mathrm{O}(|q'|\log(|\mathcal{D}|))$. □

This result yields the following complexity bounds for CXRPQ with regular relations and only basic variable definitions.

THEOREM 8.5. CXRPQ-BOOL-EVAL *with regular relations and only basic variable definitions is*

- PSpace-*complete, with respect to combined-complexity and*
- NL-*complete with respect to data complexity.*

PROOF. The upper bounds follow directly from Lemma 8.4. The lower bounds follow from Theorems 5.2 and 5.3. □

We can note that the lower bounds of Theorem 8.5 also hold for the more restricted cases mentioned in Theorems 5.2 and 5.3: The combined complexity lower bounds also holds if $|\Sigma| = 3$, $\mathcal{X}_s = \{\mathsf{x}\}$, and the query is a single-edge query; the data complexity lower bound also holds if $|\Sigma| = 2$, $\mathcal{X}_s = \emptyset$, and the query is a single-edge query.

8.3 Bounded Image Size

It can be seen with moderate effort, that the results about CXRPQ$^{\mathsf{vs}}$ with bounded image size from Section 7 also hold if we allow regular relations. More precisely, we associate with every $\mathsf{x} \in \mathcal{X}_s$ a regular relation $\sqsubset_{\mathsf{x}}$ *and* a function f_{x} that bounds the image size in terms of the graph database. However, since $\sqsubset_{\mathsf{x}}$ can be an arbitrary regular relation, the images of the definition of x and the references of x can have different lengths, so we require not only the word the variable is defined with to be bounded by f_{x}, but also all words the references are substituted with to be bounded by f_{x}.

First, we extend the definition of xregex with fixed images (see Section 7.1) by stating an explicit image not only for every variable, but for every variable definition and each of its references in the xregex. In this way, we get definitions for $\mathcal{L}^{\bar{v}}_{\mathrm{ref}}(\alpha)$ and $\mathcal{L}^{\bar{v}}(\alpha)$. Technically, this requires a more refined version of the mapping $\mathsf{vmap}_{u,\mathcal{X}_s}$ for ref-words $u \in \mathcal{L}_{\mathrm{ref}}(\alpha)$, which also depends on the xregex α, since it needs to describe exactly which variable references of the xregex are mapped to which words (it is straightforward to derive a formal definition of this from Section 7.1). At this point, we do not care about the “correctness” of the images, i. e., whether they satisfy the inter-dependency postulated by the regular relations $\sqsubset_{\mathsf{x}}$, or the length bounds given by f_{x}. For example, if each $\sqsubset_{\mathsf{x}}$ is the equality relation like in Section 7.1, then a tuple of images is only valid if all images for the definitions and all references of the same variable are the same word. Also note that even though there might be several definitions of the same variable x in a xregex, it is sufficient to only state one image for the definitions, since in every match at most one such definition can be instantiated.

We can now prove an analogue of Lemma 7.1: a conjunctive xregex $\bar{\alpha}$ and a tuple $\bar{v}$ of images (i. e., images for each definition of a variable and each reference of a variable) can be transformed into a tuple $\bar{\beta}$ of classical regular expressions such that $\mathcal{L}(\bar{\beta}) = \mathcal{L}^{\bar{v}}(\bar{\alpha})$. The proof is analogous to Lemma 7.1. We only have to keep in mind that whenever we consider a variable definition (that only contains already marked variable definitions), the images of the variable definitions *and* the variable references are now explicitly given by $\bar{v}$. Note that also Step 2 of the proof of Lemma 7.1 is necessary and can be performed in the same way.

Consequently, if we choose a tuple $\bar{v}$ of images for every variable definition and every variable reference, we can transform a $q \in \mathsf{CXRPQ}^{\mathsf{vs}}$ into a $q' \in \mathsf{CRPQ}$ that is equivalent to q with respect to the matchings that use the chosen images, i. e., for every database $\mathcal{D}$, we have that $q^{\bar{v}}(\mathcal{D}) = q'(\mathcal{D})$. Moreover, we also have that $|q'| = \mathrm{O}(|q|k)$, where $k = \max\{|\bar{v}[i]| \mid i \in [n]\}$. We note that this is an analogue of Lemma 7.3.

We now consider conjunctive xregex with bounded image size and regular relations, i. e., for every variable x we have a regular relation $\sqsubset_{\mathsf{x}}$ and a function $f_{\mathsf{x}} : \mathbb{N} \to \mathbb{N}$. Analogously to Section 7.2, we say that a tuple of images is valid with respect to $k \in \mathbb{N}$ if every image v corresponding to the definition or a reference of x satisfies $|v| \leq f_{\mathsf{x}}(k)$, and each two images v and v' such that v corresponds to the definition and v' corresponds to a reference of x satisfy $v \sqsubset_{\mathsf{x}} v'$. This also gives rise to the definition $\mathcal{L}_{\mathsf{ref}}(\alpha, k)$ of the union of all $\mathcal{L}^{\bar{v}}_{\mathsf{ref}}(\alpha)$, where $\bar{v}$ is a valid tuple of variable images for α with respect to k, and $\mathcal{L}(\alpha, k) = \mathsf{deref}(\mathcal{L}_{\mathsf{ref}}(\alpha, k))$. In particular, analogously to Section 7, this gives rise to the class $\mathsf{CXRPQ}^{\mathsf{vs,bi}}_{\Sigma,\mathcal{X}_s}$ of conjunctive xregex path queries with bounded image size and regular relations. We can also define $\mathsf{CXRPQ}^{\mathsf{vs,bi}\leq k}_{\Sigma,\mathcal{X}_s}$ as the class of $\mathsf{CXRPQ}^{\mathsf{vs,bi}}_{\Sigma,\mathcal{X}_s}$, where, for every $\mathsf{x} \in \mathcal{X}_s$, f_{x} is a constant $k' \leq k$, and $\mathsf{CXRPQ}^{\mathsf{vs,bi}\leq k\log}_{\Sigma,\mathcal{X}_s}$ as the class of $\mathsf{CXRPQ}^{\mathsf{vs,bi}}_{\Sigma,\mathcal{X}_s}$, where, for every $\mathsf{x} \in \mathcal{X}_s$, $f_{\mathsf{x}}(n)$ is a constant $k' \leq k$ or the function $k' \log(n)$ for some constant $k' \leq k$.

The same nondeterminstic algorithms as used for the proofs of Theorem 7.4 and 7.5 can now be used in order to solve $\mathsf{CXRPQ}^{\mathsf{vs,bi}\leq k}$-Bool-Eval with regular relations. We nondeterministically guess a valid tuple $\bar{v}$ of variable images for $\bar{\alpha}$ with respect to $|\mathcal{D}|$, we construct $q' \in \mathsf{CRPQ}$ such that, for every database $\mathcal{D}'$, we have that $q^{\bar{v}}(\mathcal{D}') = q'(\mathcal{D}')$, and then we check whether $\mathcal{D} \models q'$. The last point is done with Lemma 2.1, and the second point can be done with the analogue of Lemma 7.3 sketched above. For the first point, we note that even though we have to guess more images compared to the situation where each $\sqsubset_{\mathsf{x}}$ is the equality relation, this step can still be done in time $\mathrm{O}(nk)$. Furthermore, for checking the validity of a tuple, we have to check the relations $\sqsubset_{\mathsf{x}}$ for certain elements of the guessed tuple. This, however, can be done in logarithmic space, since the relations $\sqsubset_{\mathsf{x}}$ are given as NFAs.

These considerations yield the following result.

Theorem 8.6. *For every $k \in \mathbb{N}$, $\mathsf{CXRPQ}^{\mathsf{vs,bi}\leq k}$-Bool-Eval with regular relations can nondeterministically be solved in polynomial time in combined complexity and in logarithmic space in data complexity.*

Moreover, the argument for the case of $\mathsf{CXRPQ}^{\mathsf{vs,bi}\leq k\log}$-Bool-Eval is analogous.

Theorem 8.7. *For every $k \in \mathbb{N}$, $\mathsf{CXRPQ}^{\mathsf{vs,bi}\leq k\log}$-Bool-Eval with regular relations can nondeterministically be solved in polynomial time in combined complexity and in space $\mathrm{O}(\log^2(|\mathcal{D}|))$ in data complexity.*

Remark 8.8. *For $\mathsf{CXRPQ}^{\mathsf{vs,bi}\leq k}$, we could also allow more powerful relations, while staying within the complexity bounds of Theorem 8.6. This is due to the fact that we explicitly guess a tuple of images within the given length bounds, and for checking its validity, we only have to check the relations $\sqsubset_{\mathsf{x}}$. Consequently, instead of requiring relations to be regular, it is also sufficient to require that the complexity of checking the relations does not exceed the bounds of nondeterministic logarithmic space in data complexity, or nondeterministic polynomial time in combined complexity. For example, we could also allow rational relations instead of regular relations and the complexity bounds of Theorem 8.6 would remain unchanged. In fact, since for $\mathsf{CXRPQ}^{\mathsf{vs,bi}\leq k}$ the elements of the tuple of images have constant size, virtually all binary word relations should be checkable even in constant time and space. This means that as long as we have a constant upper bound for variable images, we can afford to use arbitrary relations instead of only regular ones (although the question of how these relations are to be represented needs to be discussed).*

9 CHECKING TUPLES AND COMPUTING THE WHOLE SOLUTION SET

In order to classify the complexity of CXRPQ (and its fragments), we have investigated the decision problem CXRPQ-Bool-Eval. This is common in the theoretical literature, since upper and lower bounds for the Boolean evaluation problem give a first indication of whether a certain query class might also be practically relevant. For example, the fact that non-vstar-free xregex path queries lead to PSpace-hard Boolean evaluation even in data complexity (see Lemma 3.7) strongly suggests that they must be further restricted. On the other hand, the NL data complexity of CXRPQ and its considered fragments justify hope that practical implementations are possible (if the queries are not too large). Nevertheless, for practical settings Boolean evaluation is less relevant and therefore we shall also briefly consider the problem of checking whether a given tuple is in the query result, and the problem of computing the whole solution set.

These results also demonstrate how the algorithmic approaches developed for Boolean evaluation yield algorithms for checking tuples and computing the whole solution set.

9.1 Checking Tuples

For a class Q of conjunctive xregex path queries, we denote by Q-Check the problem to check $\bar{t} \in q(\mathcal{D})$ for a given $q \in Q$, a graph database $\mathcal{D}$ and a tuple $\bar{t}$.

We now lift Lemma 8.4 to the case of checking tuples.

Lemma 9.1. *Given a set $\mathcal{X}_s$ of variables (with relations $\sqsubset_{\mathsf{x}}$ for every $\mathsf{x} \in \mathcal{X}_s$ given as* NFA*s), a $q = \bar{z} \leftarrow G_q \in$* CXRPQ$_{\Sigma, \mathcal{X}_s}$ *with $|\bar{z}| = \ell$ and with only basic variable definitions, a graph-database $\mathcal{D}$, and a tuple $\bar{t} \in (V_{\mathcal{D}})^{\ell}$, we can nondeterministically decide whether or not $\bar{t} \in q(\mathcal{D})$ in space* $\mathrm{O}(|q| \log(|\mathcal{D}|))$.

Proof. Let q be defined by a graph pattern G_q with edge set $E_q = \{(x_i, \alpha_i, y_i) \mid i \in [m]\}$ and conjunctive xregex $\bar{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_m) \in \mathsf{CXRE}_{\Sigma, \mathcal{X}_s}$. We proceed analogously as in the proof of Lemma 8.4 and first transform q into q''. Recall that, for every $i \in [m]$, $G_{q''}$ has edges $(z_{i,0}, \beta_{i,1}, z_{i,1})$, $(z_{i,1}, \beta_{i,2}, z_{i,2})$, …, $(z_{i,t_i-1}, \beta_{i,t_i}, z_{i,t_i})$, where $z_{i,0} = x_i$ and $z_{i,t_i} = y_i$. Then, we define the graph $G_{q'',\mathcal{D}} = (V_{q'',\mathcal{D}}, E_{q'',\mathcal{D}})$, but we declare a vertex $\bar{u} = (u_1, \dots, u_{m'}, p_1, \dots, p_{m'})$ as *initial*, if,

- for every $i \in [m']$, p_i is the initial state of M_i,
- for every $i, i' \in [m']$, $x_i = x_{i'}$ implies $u_i = u_{i'}$,
- for every $i \in [m']$ and $j \in [\ell]$, $x_i = \bar{z}[j]$ implies $u_i = \bar{t}[j]$.

Analogously, a vertex $\bar{v} = (v_1, \dots, v_{m'}, r_1, \dots, r_{m'})$ is *final*, if,

- for every $i \in [m']$, r_i is the final state of M_i,
- for every $i, i' \in [m']$, $y_i = y_{i'}$ implies $v_i = v_{i'}$,
- for every $i \in [m']$ and $j \in [\ell]$, $y_i = \bar{z}[j]$ implies $v_i = \bar{t}[j]$.

Analogously to the proof of Lemma 8.4, there is a path in $G_{q'',\mathcal{D}}$ from some initial vertex $\bar{u} = (u_1, \dots, u_{m'}, p_1, \dots, p_{m'})$ to some final vertex $\bar{v} = (v_1, \dots, v_{m'}, r_1, \dots, r_{m'})$ if and only if

- there is a tuple $(w_1, w_2, \dots, w_{m'}) \in (\Sigma^*)^{m'}$ that is a conjunctive match of $\bar{\gamma}$,
- for every $i \in [m]$, $\mathcal{D}$ contains a path from u_i to v_i labelled with w_i,
- for every $i \in [m']$ and $j \in [\ell]$, $x_i = \bar{z}[j]$ implies $u_i = \bar{t}[j]$, and $y_i = \bar{z}[j]$ implies $v_i = \bar{t}[j]$.

For simplicity, we add two additional vertices s and t to $G_{q'',\mathcal{D}}$ with an edge $(s, \bar{u})$ and an edge $(\bar{v}, t)$ for every initial vertex $\bar{u}$ and final vertex $\bar{v}$. Now $\bar{t} \in q(\mathcal{D})$ if and only if there is a path from s to t in $G_{q'',\mathcal{D}}$.

It follows in exactly the same way as in the proof of Lemma 8.4 that checking whether there is a path from s to t can be done in nondeterministic space $\mathrm{O}(|q| \log(|\mathcal{D}|))$. □

Analogously as Theorem 5.1 can be obtained by using Lemma 8.4 and the transformation of Section 5.2, we can now prove an equivalent of Theorem 5.1 with respect to checking tuples.

Theorem 9.2. CXRPQ-*Check is*

- *in* ExpSpace*, but* PSpace*-hard, with respect to combined-complexity and*
- NL*-complete with respect to data complexity.*

Proof. The lower bounds follow in the same way as for CXRPQ-Bool-Eval. This is due to the fact that the reduction from Theorem 5.2 is such that checking $\mathcal{D} \models q$ is equivalent to checking $(s, t) \in q(\mathcal{D})$. It is easy to see that also Theorem 5.3 has this property. This shows the lower bounds.

For the upper bounds, we can proceed analogously as in the proof of Lemma 5.4, i. e., as described in Section 5.2. Let $q \in \text{CXRPQ}_{\Sigma, \mathcal{X}_s}$ with conjunctive xregex $\bar{\alpha}$. Let $\mathcal{D}$ be a graph database and let $\bar{t} \in (V_{\mathcal{D}})^{\ell}$ be a tuple.

We first nondeterministically transform the conjunctive xregex $\bar{\alpha}$ into a variable simple conjunctive xregex $\bar{\beta}$. Then we construct an equivalent conjunctive xregex $\bar{\gamma}$ with only basic variable definitions, and we let q'' be the corresponding CXRPQ. Finally, we check whether $\bar{t} \in q''(\mathcal{D})$ with Lemma 9.1. ◻

9.2 Computing the Whole Solution Set

For a class Q of conjunctive xregex path queries, we denote by Q-Compute the problem to produce some enumeration without repetitions of the elements of the set $q(\mathcal{D})$ for a given $q \in Q$ and a graph database $\mathcal{D}$.

As common in theoretical computer science, when we talk about the space complexity of an algorithm for Q-Compute, we only refer to the working space needed by the algorithm, i. e., we do not count the size of the input or of the output.

Theorem 9.3. CXRPQ-*Compute can be solved in nondeterministic logarithmic space in data complexity.*

Proof. Let $q = \bar{z} \leftarrow G_q \in \text{CXRPQ}_{\Sigma, \mathcal{X}_s}$ with $|\bar{z}| = \ell$ and let $\mathcal{D}$ be some graph database. For each tuple $\bar{t} \in (V_{\mathcal{D}})^{\ell}$, we check whether or not $\bar{t} \in q(\mathcal{D})$, and, if this is the case, we produce $\bar{t}$ as output. This obviously produces an enumeration of $q(\mathcal{D})$, and therefore solves CXRPQ-Compute for input q and $\mathcal{D}$.

For a fixed tuple $\bar{t}$, checking whether $\bar{t} \in q(\mathcal{D})$ can be done in nondeterministic logarithmic space in data complexity due to Theorem 9.2. Consequently, the whole procedure can be carried out in nondeterministic logarithmic space in data complexity. ◻

10 EXPRESSIVE POWER

In this section, we compare the expressive power of CXRPQs and their fragments with other established classes of graph queries that are suitable for comparison.

First, we recall the definition of CRPQ, which has been presented in Section 2.2. Moreover, we define so-called *extended conjunctive regular path queries* (ECRPQs), which have been introduced in [7]. These ECRPQs are an extension of CRPQs: while CRPQs can only use regular expressions to impose unary constraints on the paths that are matched by the edges of the graph pattern, ECRPQs can use regular relations of arbitrary arity[7] to describe constraints on tuples of paths, e. g., that certain paths are all equal, or have equal length or that one path is a prefix of the other etc.

More formally, ECRPQs have the form $q = \bar{z} \leftarrow G_q, \bigwedge_{j \in [t]} R_j(\bar{\omega}_j)$, where $\bar{z} \leftarrow G_q$ is a CRPQ and, for every $j \in [t]$, $\bar{\omega}_j$ is an s_j-tuple over E_q and R_j is a regular expression that describes a regular relation over Σ^* of arity s_j. The semantics of ECRPQ can be derived from the semantics of CRPQ as follows. We interpret q as a CRPQ, but we add to the concept of a matching morphism the requirement that there must be a tuple $(w_{e_1}, w_{e_2}, \ldots, w_{e_m})$ of matching words such that, for each $\bar{\omega}_j = (e_{p_1}, e_{p_2}, \ldots, e_{p_{s_j}})$, we have $(w_{e_{p_1}}, w_{e_{p_2}}, \ldots, w_{e_{p_{s_j}}}) \in \mathcal{L}(R_j)$.

For our comparison with CXRPQ, we are mainly interested in ECRPQ^{er}, i. e., the fragment of ECRPQ obtained by only allowing unary relations or equality relations (i. e., relations requiring certain paths to be equal).

[7]Note that the definition of regularity for binary relations over Σ^* from the beginning of Section 8.1 extends to relations of larger arity in a natural way.

$$\begin{tikzcd}
\llbracket \mathsf{ECRPQ} \rrbracket \arrow[rrrrr, dashed] & & & & & \llbracket \cup\text{-}\mathsf{ECRPQ} \rrbracket \\
& & \llbracket \mathsf{CXRPQ}^{\mathsf{vs}} \rrbracket & & & \\
\llbracket \mathsf{ECRPQ}^{\mathsf{er}} \rrbracket \arrow[uu] \arrow[urr, bend left] \arrow[r, dashed] & \left\llbracket \mathsf{CXRPQ}^{\mathsf{d} \leq 0} \right\rrbracket \arrow[r, dashed] & \left\llbracket \mathsf{CXRPQ}^{\mathsf{d} \leq 1} \right\rrbracket \arrow[r, dashed] & \cdots \arrow[r, dashed] & \llbracket \mathsf{CXRPQ} \rrbracket \arrow[ull, dashed, bend right] \arrow[r, dashed] & \llbracket \cup\text{-}\mathsf{ECRPQ}^{\mathsf{er}} \rrbracket \arrow[uu] \\
\llbracket \mathsf{CRPQ} \rrbracket \arrow[u] \arrow[r] & \left\llbracket \mathsf{CXRPQ}^{\mathsf{vs,bi} \leq 1} \right\rrbracket \arrow[r, dashed] & \left\llbracket \mathsf{CXRPQ}^{\mathsf{vs,bi} \leq 2} \right\rrbracket \arrow[r, dashed] & \cdots \arrow[ur, dashed, bend right] \arrow[rr, dashed] & & \llbracket \cup\text{-}\mathsf{CRPQ} \rrbracket \arrow[u]
\end{tikzcd}$$

Fig. 6. Illustration of the relations between the considered classes of conjunctive path queries. A dashed or solid arrow from A to B denotes $A \subseteq B$ or $A \subsetneq B$, respectively. Note that $\mathsf{CXRPQ}^{\mathsf{depth} \leq i}$ is abbreviated as $\mathsf{CXRPQ}^{\mathsf{d} \leq i}$.

For a query class Q, a *union of Qs* (or $\cup$-Q, for short) is a query of the form $q = q_1 \vee q_2 \vee \ldots \vee q_k$, where, for every $i \in [k]$, $q_i \in Q$. For a graph-database $\mathcal{D}$, we define $q(\mathcal{D}) = \bigcup_{i \in [k]} q_i(\mathcal{D})$.

Recall that by $\mathsf{CXRPQ}^{\mathsf{vs}}$ we denote the class of queries obtained by dropping the requirement that the conjunctive xregex must be variable-star free (see also the last paragraph of Section 4.2).

Theorem 10.1. *The inclusions illustrated by Figure 6 are correct.*

We shall discuss the result of Theorem 10.1 on an intuitive level, and provide proofs in full detail in the appendix.

The diagram of Figure 6 shows on the left and on the right side two vertical layers of query classes of increasing expressive power. More precisely, on the left side we have the *classical* conjunctive path query classes CRPQ, $\mathsf{ECRPQ}^{\mathsf{er}}$ and ECRPQ, and on the right side, we have the union-closures of these classes. These two layers are naturally ordered by the subset relation, and all these inclusions follow directly by definition. However, the strictness of the inclusions

$$\llbracket \mathsf{CRPQ} \rrbracket \subsetneq \llbracket \mathsf{ECRPQ}^{\mathsf{er}} \rrbracket \subsetneq \llbracket \mathsf{ECRPQ} \rrbracket$$

need individual proofs that, if slightly adapted, also show the strictness of the inclusions

$$\llbracket \cup\text{-}\mathsf{CRPQ} \rrbracket \subsetneq \llbracket \cup\text{-}\mathsf{ECRPQ}^{\mathsf{er}} \rrbracket \subsetneq \llbracket \cup\text{-}\mathsf{ECRPQ} \rrbracket .$$

The strictness of $\llbracket \mathsf{CRPQ} \rrbracket \subsetneq \llbracket \mathsf{ECRPQ}^{\mathsf{er}} \rrbracket$ and $\llbracket \mathsf{ECRPQ}^{\mathsf{er}} \rrbracket \subsetneq \llbracket \mathsf{ECRPQ} \rrbracket$ are not mentioned in the literature, but the strictness of $\llbracket \mathsf{CRPQ} \rrbracket \subsetneq \llbracket \mathsf{ECRPQ} \rrbracket$ is shown in [7].

All the CXRPQ fragments introduced in this paper (except the class $\mathsf{CXRPQ}^{\mathsf{vs}}$) lie in between these two layers.

That $\llbracket \mathsf{ECRPQ}^{\mathsf{er}} \rrbracket$ is included in $\llbracket \mathsf{CXRPQ} \rrbracket$ is expected, but $\llbracket \mathsf{ECRPQ}^{\mathsf{er}} \rrbracket \subseteq \llbracket \mathsf{CXRPQ}^{\mathsf{depth} \leq 0} \rrbracket$ points out that quite restricted classes of CXRPQ already cover CRPQ extended by equality relations. That $\llbracket \mathsf{CRPQ} \rrbracket$ is a subset of $\llbracket \mathsf{CXRPQ}^{\mathsf{vs,bi} \leq 1} \rrbracket$ is not surprising, but we can also show that this inclusion is strict, which is less obvious. After all, $\mathsf{CXRPQ}^{\mathsf{vs,bi} \leq 1}$ can use variable references only for referencing single symbols; thus, it seems reasonable that we can replace the xregex by more complicated classical regular expressions. In fact, this is true even for arbitrary large image size bounds if we are talking about string matching, but it is not the case if xregex are used as graph queries. Intuitively speaking, even $\mathsf{CXRPQ}^{\mathsf{vs,bi} \leq 1}$ can use the variables to define inter-path dependencies that cannot be described by any CRPQ.

The inclusion $\llbracket \mathsf{CXRPQ} \rrbracket \subseteq \llbracket \cup\text{-}\mathsf{ECRPQ}^{\mathsf{er}} \rrbracket$ is rather non-trivial to show, and the proof requires the transformation techniques from the proof of Lemma 5.4 described in Section 5.2. We stress here the fact that the conversion

CRPQ	ECRPQ^{er}	CXRPQ	$\text{CXRPQ}^{\text{depth} \leq k}$	$\text{CXRPQ}^{\text{vs}, \text{bi} \leq k}$
NP	PSpace	ExpSpace	PSpace	NP
NL	NL	NL	NL	NL

Table 1. A summary of the evaluation complexity upper bounds in combined complexity (upper row) and data complexity (lower row).

from CXRPQ to a union of ECRPQ^{er} may produce exponentially many ECRPQ^{er}s, each of which may be of exponential size. This also demonstrates that even though we can express any CXRPQ as a union of ECRPQ this is, at least by the transformations presented here, expected to be very complicated and unwieldy. Analogously, the fact that we can express $\text{CXRPQ}^{\text{vs}, \text{bi} \leq k}$ as unions of CRPQ follow from the upper bounds presented in Section 7.

From an intuitive point of view, it is to be expected that the inclusion of ECRPQ^{er} in CXRPQ is strict. However, we can only show that CXRPQ^{vs} are strictly more powerful than ECRPQ^{er} (and even this result has a non-trivial proof). We expect a proof for the strictness of $[\![\text{ECRPQ}^{\text{er}}]\!] \subseteq [\![\text{CXRPQ}]\!]$ to be rather challenging.

11 CONCLUSIONS AND OPEN PROBLEMS

Table 1 summarises the upper complexity bounds for the fragments presented in this work (note that we have also shown matching lower bounds for all these upper bounds, except for the ExpSpace upper bound in combined complexity for CXRPQ, for which we only know PSpace-hardness). For comparison, we have added columns for CRPQ and ECRPQ^{er} (note that in Table 1, ECRPQ^{er} could also be replaced by ECRPQ, since the upper bounds are the same).

In data complexity, the picture looks rather good, since all our fragments of CXRPQ achieve an evaluation complexity of NL. This gives some hope that in scenarios were queries can be expected to be small, these fragments are also practically relevant. In combined complexity, the only upper bound that seems problematic is the one for the unrestricted class CXRPQ. However, for real-world queries, it seems very likely that the depth is not very large; in fact, conjunctive xregex with a depth of, say 5 or 8, are already quite complex and difficult to parse for a human user. Therefore, it is a safe assumption that in practical scenarios CXRPQ would most likely fall in the fragment $\text{CXRPQ}^{\text{depth} \leq k}$ for even low k. We might even go a step further and assume that in most cases the total number of variables – which also bounds the depth – is not very high. At least this seems to be the case if regular expressions with capture variables are used as string searching tools, which also explains why they find such wide practical application even though they are theoretically intractable to match. It is also worth noting that all the practically motivated examples from Section 1.4 belong to the fragments with good evaluation complexity.

Nevertheless, the question whether the ExpSpace upper bound for CXRPQ can be lowered to PSpace is definitely of theoretical interest.

Several of our results implicitly pose conciseness questions. Each $\text{CXRPQ}^{\text{vs}, \text{bi} \leq k}$ can be represented as the union of $O(|\Sigma| + 1)^{|\mathcal{X}_s|k})$ many CRPQs, and each CXRPQ can be represented as the union of exponentially many ECRPQ^{er}s of exponential size. Are these exponential blow-ups necessary?

The classical use of regular expressions with capture variables is that of a string searching tool, while using them for graph querying is a rather new approach. Nevertheless, it is very interesting to compare these two scenarios directly with each other. In the string case, the NP-complete matching problem for xregex trivially becomes polynomial-time solvable, if the number of variables is bounded by a constant (see [48]). For xregex path queries that are not variable-star free, bounding the number of variables has now such effect, as pointed out by Lemma 3.7. However, if we consider CXRPQ, which have variable-star free conjunctive xregex, then bounding the number of variables bounds the depth, and therefore improves our upper bound to PSpace. Moreover, the restriction of variable star-freeness has a huge impact for xregex path queries: it lets the data-complexity drop from PSpace-hardness to NL (see Lemma 3.7 and Theorem 5.1). Likewise, restricting the image size of the variables by a constant lets the combined complexity drop from PSpace-hardness to NP. However, the matching problem for xregex in the string case remains NP-hard, even if we require variable-star freeness *and* that variables can only range over words of length at most 1 (see [25, Theorem 3]).

REFERENCES

[1] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. 2017. Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.* 50, 5 (2017), 68:1–68:40.

[2] Renzo Angles, Juan L. Reutter, and Hannes Voigt. 2019. Graph Query Languages. In *Encyclopedia of Big Data Technologies.*

[3] Guillaume Bagan, Angela Bonifati, and Benoît Groz. 2013. A trichotomy for regular simple path queries on graphs. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013.* 261–272.

[4] Jean-François Baget, Meghyn Bienvenu, Marie-Laure Mugnier, and Michaël Thomazo. 2017. Answering Conjunctive Regular Path Queries over Guarded Existential Rules. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017.* 793–799. https://doi.org/10.24963/ijcai.2017/110

[5] Pablo Barceló. 2013. Querying graph databases. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013.* 175–188.

[6] Pablo Barceló, Diego Figueira, and Miguel Romero. 2019. Boundedness of Conjunctive Regular Path Queries. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece.* 104:1–104:15.

[7] Pablo Barceló, Leonid Libkin, Anthony Widjaja Lin, and Peter T. Wood. 2012. Expressive Languages for Path Queries over Graph-Structured Data. *ACM Transactions on Database Systems (TODS)* 37, 4 (2012), 31:1–31:46.

[8] Pablo Barceló, Leonid Libkin, and Juan L. Reutter. 2014. Querying Regular Graph Patterns. *J. ACM* 61, 1 (2014), 8:1–8:54.

[9] Pablo Barceló and Pablo Muñoz. 2017. Graph Logics with Rational Relations: The Role of Word Combinatorics. *ACM Transactions on Computational Logic (TOCL)* 18, 2 (2017), 10:1–10:41.

[10] Pablo Barceló, Miguel Romero, and Moshe Y. Vardi. 2016. Semantic Acyclicity on Graph Databases. *SIAM J. Comput.* 45, 4 (2016), 1339–1376.

[11] Meghyn Bienvenu, Magdalena Ortiz, and Mantas Simkus. 2015. Regular Path Queries in Lightweight Description Logics: Complexity and Algorithms. *J. Artif. Intell. Res.* 53 (2015), 315–374. https://doi.org/10.1613/jair.4577

[12] Meghyn Bienvenu and Michaël Thomazo. 2016. On the Complexity of Evaluating Regular Path Queries over Linear Existential Rules. In *Web Reasoning and Rule Systems - 10th International Conference, RR 2016, Aberdeen, UK, September 9-11, 2016, Proceedings.* 1–17. https://doi.org/10.1007/978-3-319-45276-0_1

[13] Angela Bonifati, Wim Martens, and Thomas Timm. 2017. An Analytical Study of Large SPARQL Query Logs. *PVLDB* 11, 2 (2017), 149–161.

[14] Angela Bonifati, Wim Martens, and Thomas Timm. 2020. An analytical study of large SPARQL query logs. *VLDB J.* 29, 2-3 (2020), 655–679. https://doi.org/10.1007/s00778-019-00558-9

[15] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. 2000. Containment of Conjunctive Regular Path Queries with Inverse. In *KR 2000, Principles of Knowledge Representation and Reasoning Proceedings of the Seventh International Conference, Breckenridge, Colorado, USA, April 11-15, 2000.* 176–185.

[16] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. 2003. Reasoning on regular path queries. *SIGMOD Record* 32, 4 (2003), 83–92.

[17] Cezar Câmpeanu, Kai Salomaa, and Sheng Yu. 2003. A Formal Study Of Practical Regular Expressions. *Int. J. Found. Comput. Sci.* 14, 6 (2003), 1007–1018. https://doi.org/10.1142/S012905410300214X

[18] Benjamin Carle and Paliath Narendran. 2009. On Extended Regular Expressions. In *Language and Automata Theory and Applications, Third International Conference, LATA 2009, Tarragona, Spain, April 2-8, 2009. Proceedings.* 279–289. https://doi.org/10.1007/978-3-642-00982-2_24

[19] Katrin Casel and Markus L. Schmid. 2021. Fine-Grained Complexity of Regular Path Queries. In *24th International Conference on Database Theory, ICDT 2021, March 23-26, 2021, Nicosia, Cyprus.* 19:1–19:20. https://doi.org/10.4230/LIPIcs.ICDT.2021.19

[20] Mariano P. Consens and Alberto O. Mendelzon. 1990. GraphLog: a Visual Formalism for Real Life Recursion. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA.* 404–416.

[21] Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. 1987. A Graphical Query Language Supporting Recursion. In *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, California, May 27-29, 1987.* 323–330.

[22] Wojciech Czerwinski, Wim Martens, Matthias Niewerth, and Pawel Parys. 2018. Minimization of Tree Patterns. *J. ACM* 65, 4 (2018), 26:1–26:46.

[23] Alin Deutsch and Val Tannen. 2001. Optimization Properties for Classes of Conjunctive Regular Path Queries. In *Database Programming Languages, 8th International Workshop, DBPL 2001, Frascati, Italy, September 8-10, 2001, Revised Papers*. 21–39.

[24] Johannes Doleschal, Benny Kimelfeld, Wim Martens, Yoav Nahshon, and Frank Neven. 2019. Split-Correctness in Information Extraction. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. 149–163.

[25] Henning Fernau and Markus L. Schmid. 2015. Pattern matching with variables: A multivariate complexity analysis. *Information and Computation (I&C)* 242 (2015), 287–305.

[26] Diego Figueira. 2020. Containment of UC2RPQ: The Hard and Easy Cases. In *23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark*. 9:1–9:18. https://doi.org/10.4230/LIPIcs.ICDT.2020.9

[27] Diego Figueira, Adwait Godbole, Shankara Narayanan Krishna, Wim Martens, Matthias Niewerth, and Tina Trautner. 2020. Containment of Simple Regular Path Queries. *CoRR* abs/2003.04411 (2020). arXiv:2003.04411 https://arxiv.org/abs/2003.04411

[28] Daniela Florescu, Alon Y. Levy, and Dan Suciu. 1998. Query Containment for Conjunctive Queries with Regular Expressions. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA*. 139–148.

[29] Dominik D. Freydenberger. 2013. Extended Regular Expressions: Succinctness and Decidability. *Theory of Computing Systems (ToCS)* 53, 2 (2013), 159–193.

[30] Dominik D. Freydenberger. 2019. A Logic for Document Spanners. *Theory Comput. Syst.* 63, 7 (2019), 1679–1754.

[31] Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. 2018. Joining Extractions of Regular Expressions. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*. 137–149.

[32] Dominik D. Freydenberger and Markus L. Schmid. 2019. Deterministic regular expressions with back-references. *J. Comput. Syst. Sci.* 105 (2019), 1–39. https://doi.org/10.1016/j.jcss.2019.04.001

[33] Dominik D. Freydenberger and Nicole Schweikardt. 2013. Expressiveness and static analysis of extended conjunctive regular path queries. *J. Comput. System Sci.* 79, 6 (2013), 892–909.

[34] Jeffrey E. F. Friedl. 2006. *Mastering regular expressions - understand your data and be more productive: for Perl, PHP, Java, .NET, Ruby, and more (3. ed.)*. O'Reilly.

[35] Grzegorz Gluch, Jerzy Marcinkowski, and Piotr Ostropolski-Nalewaja. 2019. The First Order Truth Behind Undecidability of Regular Path Queries Determinacy. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*. 15:1–15:18.

[36] Steve Harris and Andy Seaborne. 2013. *SPARQL 1.1 Query Language. W3C Recommendation*. Technical Report https://www.w3.org/TR/sparql11-query/. W3C.

[37] S.C. Kleene. 1956. Representation of events in nerve nets and finite automata. In *Automata Studies*, C.E. Shannon and J. McCarthy (Eds.). Annals of Mathematics Studies, Vol. 34. Princeton University Press, 3–41.

[38] Egor V. Kostylev, Juan L. Reutter, and Domagoj Vrgoc. 2018. Containment of queries for graphs with data. *J. Comput. Syst. Sci.* 92 (2018), 65–91.

[39] Leonid Libkin, Wim Martens, and Domagoj Vrgoc. 2016. Querying Graphs with Data. *Journal of the ACM* 63, 2 (2016), 14:1–14:53.

[40] Katja Losemann and Wim Martens. 2013. The complexity of regular expressions and property paths in SPARQL. *ACM Transactions on Database Systems (TODS)* 38, 4 (2013), 24:1–24:39.

[41] Wim Martens, Matthias Niewerth, and Tina Trautner. 2020. A Trichotomy for Regular Trail Queries. In *37th Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, Germany*. Accepted for publication.

[42] Wim Martens and Tina Trautner. 2018. Evaluation and Enumeration Problems for Regular Path Queries. In *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*. 19:1–19:21.

[43] Wim Martens and Tina Trautner. 2019. Bridging Theory and Practice with Query Log Analysis. *SIGMOD Record* 48, 1 (2019), 6–13.

[44] Wim Martens and Tina Trautner. 2019. Dichotomies for Evaluating Simple Regular Path Queries. *ACM Trans. Database Syst.* 44, 4 (2019), 16:1–16:46. https://doi.org/10.1145/3331446

[45] Alberto O. Mendelzon and Peter T. Wood. 1995. Finding Regular Simple Paths in Graph Databases. *SIAM Journal on Computing (SICOMP)* 24, 6 (1995), 1235–1258.

[46] Juan L. Reutter, Miguel Romero, and Moshe Y. Vardi. 2017. Regular Queries on Graph Databases. *Theory of Computing Systems (ToCS)* 61, 1 (2017), 31–83.

[47] Miguel Romero, Pablo Barceló, and Moshe Y. Vardi. 2017. The homomorphism problem for regular graph patterns. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. 1–12. https://doi.org/10.1109/LICS.2017.8005106

[48] Markus L. Schmid. 2016. Characterising REGEX languages by regular languages equipped with factor-referencing. *Information and Computation (I&C)* 249 (2016), 1–17.

[49] Markus L. Schmid. 2020. Conjunctive Regular Path Queries with String Variables. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*. 361–374. https://doi.org/10.1145/3375395.3387663

[50] Markus L. Schmid and Nicole Schweikardt. 2021. A Purely Regular Approach to Non-Regular Core Spanners. In *24th International Conference on Database Theory, ICDT 2021, March 23-26, 2021, Nicosia, Cyprus*. 4:1–4:19. https://doi.org/10.4230/LIPIcs.ICDT.2021.4
[51] Markus L. Schmid and Nicole Schweikardt. 2021. Spanner Evaluation over SLP-Compressed Documents. In *PODS'21: Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Virtual Event, China, June 20-25, 2021*. 153–165. https://doi.org/10.1145/3452021.3458325
[52] The Neo4j Team. 2016. The Neo4j Cypher Manual v3.5. retrieved from https://neo4j.com/docs/developer-manual/current/cypher/.
[53] K. Thompson. 1968. Programming Techniques: Regular expression search algorithm. *Commun. ACM* 11 (1968).
[54] Apache TinkerPop^{TM}. 2013. Gremlin Language. retrieved from https://tinkerpop.apache.org/gremlin.html.
[55] Peter T. Wood. 2012. Query languages for graph databases. *SIGMOD Record* 41, 1 (2012), 50–60.

A PROOF OF PROPOSITION 3.4

Proof. If we reach Step 2 without any definition for a variable, then there can also be no reference of a variable, since we removed all variable references that have no definition in Step 1, and for every variable definition removed in Step 2b, we also replaced all corresponding references by words over Σ. Consequently, if we reach Step 2 without variable definitions, then we have obtained a word over Σ and terminate. If, on the other hand, there is at least one definition for some variable x when we reach Step 2, then, due to the acyclicity of ref-words, there must be at least one definition that does not contain another definition. Hence, there must be at least one definition $^{\mathsf{x}}\triangleright\, v_{\mathsf{x}} \triangleleft^{\mathsf{x}}$ such that $v_{\mathsf{x}} \in \Sigma^*$, as required by Step 2a. In Step 2b this definition of x and all its references are then replaced by a word over Σ. □

B PROOF OF LEMMA 4.4

Proof. We first observe that $\mathcal{L}_{\text{ref}}(\alpha_i) = \mathcal{L}_{\text{ref}}(\beta_i)$, for every $i \in [m]$, implies $\mathcal{L}_{\text{ref}}(\beta_1\beta_2\ldots\beta_m) = \mathcal{L}_{\text{ref}}(\alpha_1\alpha_2\ldots\alpha_m)$. If $\bar{\beta}$ is not a conjunctive xregex, then $\beta_1\beta_2\ldots\beta_m$ is not a valid xregex. However, if $\beta_1\beta_2\ldots\beta_m$ is not valid, then $\mathcal{L}_{\text{ref}}(\beta_1\beta_2\ldots\beta_m)$ must contain a word that is not a ref-word. This directly implies, that $\mathcal{L}_{\text{ref}}(\alpha_1\alpha_2\ldots\alpha_m)$ contains a word that is not a ref-word, which contradicts the fact that $\alpha_1\alpha_2\ldots\alpha_m$ is a valid xregex.

It remains to show that $\mathcal{L}(\bar{\alpha}) = \mathcal{L}(\bar{\beta})$:

$$\begin{aligned}\mathcal{L}(\bar{\alpha}) &= \{(w_1, w_2, \ldots, w_m) \mid w_1\#w_2\#\ldots\#w_m \in \mathcal{L}(\alpha_1\#\alpha_2\#\ldots\#\alpha_m)\}\\ &= \{(w_1, w_2, \ldots, w_m) \mid w_1\#w_2\#\ldots\#w_m \in \mathsf{deref}(\mathcal{L}_{\text{ref}}(\alpha_1\#)\,\mathcal{L}_{\text{ref}}(\alpha_2\#)\ldots\mathcal{L}_{\text{ref}}(\#\alpha_m))\}\\ &= \{(w_1, w_2, \ldots, w_m) \mid w_1\#w_2\#\ldots\#w_m \in \mathsf{deref}(\mathcal{L}_{\text{ref}}(\beta_1\#)\,\mathcal{L}_{\text{ref}}(\beta_2\#)\ldots\mathcal{L}_{\text{ref}}(\#\beta_m))\}\\ &= \{(w_1, w_2, \ldots, w_m) \mid w_1\#w_2\#\ldots\#w_m \in \mathcal{L}(\beta_1\#\beta_2\#\ldots\#\beta_m)\} = \mathcal{L}(\bar{\beta})\,.\end{aligned}$$ □

C PROOF OF THEOREM 8.3

We first recall the main definitions. For an alphabet Σ, let the *palindrome* relation be defined by $u \sqsubset_P v$ if $u = v = w(w)^R$ for some $w \in \Sigma^*$; for any $\Delta \subseteq \Sigma$, let the Δ-*restriction relation* $\sqsubset_\Delta$ be defined by $u \sqsubset_\Sigma v$ if u restricted to alphabet Σ equals v.

The *post correspondence problem* (PCP) is defined as follows. Given two lists $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$ of words over some alphabet Δ, decide whether there are indices $i_1, i_2, \ldots, i_k \in [n]$ with $a_{i_1}a_{i_2}\ldots a_{i_k} = b_{i_1}b_{i_2}\ldots b_{i_k}$. It is a well-known fact that PCP is an undecidable problem.

Given a PCP-instance $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$ over some alphabet Δ, we define the following regular languages and relations over $\Sigma = \Delta \cup [n]$. Let $r_{a,n} = (a_1 1 \vee a_2 2 \vee \ldots \vee a_n n)$ and $r_{b,n} = ((b_1)^R 1 \vee (b_2)^R 2 \vee \ldots \vee (b_n)^R n)$.

Lemma C.1. *The PCP instance is positive if and only if there are $w_a \in \mathcal{L}(r_{a,n})$, $w_b \in \mathcal{L}(r_{b,n})$, $u_a, u_b \in [n]^*$, and $v_a, v_b \in \Delta^*$ such that*

$$\begin{array}{lll} w_a \sqsubset_{[n]} u_a & w_a \sqsubset_\Delta v_a & u_a u_b \sqsubset_P u_a u_b\,,\\ w_b \sqsubset_{[n]} u_b & w_b \sqsubset_\Delta v_b & v_a v_b \sqsubset_P v_a v_b\,.\end{array}$$

Proof. We start with the *if* direction. Let $w_a \in \mathcal{L}(r_{a,n})$, $w_b \in \mathcal{L}(r_{b,n})$, $u_a, u_b \in [n]^*$, and $v_a, v_b \in \Delta^*$ satisfy the properties of the lemma. This means that there are $i_1, i_2, \ldots, i_{k_a} \in [n]$ and $j_1, j_2, \ldots, j_{k_b} \in [n]$ such that

$$\begin{aligned}
w_a &= a_{i_1} i_1 a_{i_2} i_2 \ldots a_{i_{k_a}} i_{k_a}\,,\\
w_b &= (b_{j_{k_b}})^R j_{k_b} (a_{j_{k_b-1}})^R j_{k_b-1} \ldots (a_{j_1})^R j_1\,,\\
u_a &= i_1 i_2 \ldots i_{k_a}\,,\\
u_b &= j_{k_b} j_{k_b-1} \ldots j_1\,,\\
v_a &= a_{i_1} a_{i_2} \ldots a_{i_{k_a}}\,,\\
v_b &= (b_{j_{k_b}})^R (b_{j_{k_b-1}})^R \ldots (b_{j_1})^R\,.
\end{aligned}$$

From $u_a u_b \sqsubset_P u_a u_b$, we conclude that $u_a u_b = u(u)^R$ for some u, which implies that $u_a = (u_b)^R$. Thus, we have $i_1 i_2 \ldots i_{k_a} = j_1 j_2 \ldots j_{k_b}$, which implies that $k_a = k_b$, and $i_\ell = j_\ell$ for every $\ell \in [k_a]$. Analogously, from $v_a v_b \sqsubset_P v_a v_b$, we conclude that $v_a v_b = u(u)^R$ for some u, which implies that $v_a = (v_b)^R$. Consequently, $a_{i_1} a_{i_2} \ldots a_{i_{k_a}} = ((b_{j_{k_a}})^R (b_{j_{k_a-1}})^R \ldots (b_{j_1})^R)^R$. Since

$$((b_{j_{k_a}})^R (b_{j_{k_a-1}})^R \ldots (b_{j_1})^R)^R = ((b_{j_1} b_{j_2} \ldots b_{j_{k_a}})^R)^R = b_{j_1} b_{j_2} \ldots b_{j_{k_a}}$$

this implies that $a_{i_1} a_{i_2} \ldots a_{i_{k_a}} = b_{j_1} b_{j_2} \ldots b_{j_{k_a}}$. Thus, $i_1, i_2, \ldots, i_{k_a} \in [n]$ is a solution for the PCP-instance.

Next, we prove the *only if* direction. Let $i_1, i_2, \ldots, i_k \in [n]$ be a solution for the PCP instance. Recall that this means that $a_{i_1} a_{i_2} \ldots a_{i_k} = b_{i_1} b_{i_2} \ldots b_{i_k}$. We define

$$\begin{aligned}
w_a &= a_{i_1} i_1 a_{i_2} i_2 \ldots a_{i_k} i_k\,,\\
w_b &= (b_{i_k})^R i_k (b_{i_{k-1}})^R i_{k-1} \ldots (b_{i_1})^R i_1\,,\\
u_a &= i_1 i_2 \ldots i_k\,,\\
u_b &= i_k i_{k-1} \ldots i_1\,,\\
v_a &= a_{i_1} a_{i_2} \ldots a_{i_k}\,,\\
v_b &= (b_{i_k})^R (b_{i_{k-1}})^R \ldots (b_{i_1})^R\,.
\end{aligned}$$

Obviously, $w_a \in \mathcal{L}(r_{a,n})$ and $w_b \in \mathcal{L}(r_{b,n})$. Furthermore, $w_a \sqsubset_{[n]} u_a$, $w_b \sqsubset_{[n]} u_b$, $w_a \sqsubset_\Delta v_a$ and $w_b \sqsubset_\Delta v_b$. Finally, we observe that $u_a = i_1 i_2 \ldots i_k = (i_k i_{k-1} \ldots i_1)^R = (u_b)^R$ and

$$\begin{aligned}
v_a &= a_{i_1} a_{i_2} \ldots a_{i_k}\\
&= b_{i_1} b_{i_2} \ldots b_{i_k}\\
&= ((b_{i_1} b_{i_2} \ldots b_{i_k})^R)^R\\
&= ((b_{i_k})^R (b_{i_{k-1}})^R \ldots (b_{i_1})^R)^R = (v_b)^R\,.
\end{aligned}$$

Hence, $u_a u_b \sqsubset_P u_a u_b$ and $v_a v_b \sqsubset_P v_a v_b$, which concludes the proof. □

Let $\mathsf{x}_{a,[n]}$, $\mathsf{x}_{b,[n]}$, $\mathsf{x}_{a,\Delta}$, $\mathsf{x}_{b,\Delta}$, $\mathsf{y}_{[n]}$ and y_Δ be variables with the following relations:

$$\begin{aligned}
\sqsubset_{\mathsf{x}_{a,[n]}} &= \sqsubset_{\mathsf{x}_{b,[n]}} = \sqsubset_{[n]}\,,\\
\sqsubset_{\mathsf{x}_{a,\Delta}} &= \sqsubset_{\mathsf{x}_{b,[n]}} = \sqsubset_{\Delta}\,,\\
\sqsubset_{\mathsf{y}_{[n]}} &= \sqsubset_{\mathsf{y}_\Delta} = \sqsubset_P\,.
\end{aligned}$$

We define the conjunctive xregex $\bar{\alpha} = (\alpha_a, \alpha_b, \beta_{P,[n]}, \beta_{P,\Delta}, \gamma_{[n]}, \gamma_\Delta)$, where

$$\begin{aligned}\alpha_a &= \mathsf{x}_{a,[n]}\{\mathsf{x}_{a,\Delta}\{r_{a,n}\}\}\,,\\ \alpha_b &= \mathsf{x}_{b,[n]}\{\mathsf{x}_{b,\Delta}\{r_{b,n}\}\}\,,\\ \beta_{P,[n]} &= \mathsf{y}_{[n]}\{\mathsf{x}_{a,[n]}\mathsf{x}_{b,[n]}\}\,,\\ \beta_{P,\Delta} &= \mathsf{y}_\Delta\{\mathsf{x}_{a,\Delta}\mathsf{x}_{b,\Delta}\}\,,\\ \gamma_{[n]} &= \mathsf{y}_{[n]}\,,\\ \gamma_{[n]} &= \mathsf{y}_\Delta\end{aligned}$$

Lemma C.2. *There exists a conjunctive match for $\bar{\alpha}$ if and only if there are $w_a \in \mathcal{L}(r_{a,n})$, $w_b \in \mathcal{L}(r_{b,n})$, $u_a, u_b \in [n]^*$, and $v_a, v_b \in \Delta^*$ such that*

$$\begin{array}{lll} w_a \sqsubset_{[n]} u_a & w_a \sqsubset_\Delta v_a & u_a u_b \sqsubset_P u_a u_b\,,\\ w_b \sqsubset_{[n]} u_b & w_b \sqsubset_\Delta v_b & v_a v_b \sqsubset_P v_a v_b\,. \end{array}$$

Proof. We start with the *if* direction. Let $w_a \in \mathcal{L}(r_{a,n})$, $w_b \in \mathcal{L}(r_{b,n})$, $u_a, u_b \in [n]^*$, and $v_a, v_b \in \Delta^*$ satisfy the properties of the lemma. Then, by definition of conjunctive xregex, $(w_a, w_b, u_a u_b, v_a v_b, u_a u_b, v_a v_b)$ is a conjunctive match for $\bar{\alpha}$.

In order to prove the *only if* direction, we assume that $(w_1, w_2, \ldots, w_6)$ is a conjunctive match for $\bar{\alpha}$. This means that $w_1 \in \mathcal{L}(r_{a,n})$ and $w_2 \in \mathcal{L}(r_{b,n})$. Furthermore, there are words u_a and u_b (namely the ones corresponding to the references of $\mathsf{x}_{a,[n]}$ and $\mathsf{x}_{b,[n]}$) with $w_1 \sqsubset_{[n]} u_a$ and $w_2 \sqsubset_{[n]} u_b$, and there are words v_a and v_b (namely the ones corresponding to the references of $\mathsf{x}_{a,\Delta}$ and $\mathsf{x}_{b,\Delta}$) with $w_1 \sqsubset_\Delta v_a$ and $w_2 \sqsubset_\Delta v_b$. This also means that $w_3 = u_a u_b$ and $w_4 = v_a v_b$. Finally, w_5 is a word with $w_3 = u_a u_b \sqsubset_P w_5$ and w_6 is a word with $w_4 = v_a v_b \sqsubset_P w_6$. This means that the words $w_a := w_1$, $w_b := w_2$, u_a, u_b, v_a and v_b satisfy the conditions of the lemma. □

We are now ready to give a proof of Theorem 8.3:

Proof. Given a PCP instance, we first construct the conjunctive xregex $\bar{\alpha} = (\alpha_a, \alpha_b, \beta_{P,[n]}, \beta_{P,\Delta}, \gamma_{[n]}, \gamma_\Delta)$, then we construct a CXRPQ that has just two nodes with 6 arcs each of which pointing from one node to the other, and the arcs are labelled with the components of $\bar{\alpha}$. The graph database $\mathcal{D}$ is just a single node v with a loop for every symbol in Σ. Note that for every $w \in \Sigma^*$ there is a path from v to v labelled with w.

If $\mathcal{D} \models q$, then there is a conjunctive match for $\bar{\alpha}$ and, according to Lemmas C.1 and C.2, this means that the PCP instance is positive. On the other hand, if the PCP instance is positive, then there is a conjunctive match for $\bar{\alpha}$, which, by construction of $\mathcal{D}$, means that $\mathcal{D} \models q$. □

D FORMAL PROOFS FOR THE INCLUSIONS OF FIGURE 6

For convenience, we repeat Figure 6 here again as Figure 7.

We recall that by ECRPQ with *equality relations* (ECRPQ$^{\text{er}}$ for short), we denote the class of ECRPQ for which each R_j is the *equality relation* (for some arity s_j), i. e., the relation $\{(u_1, u_2, \ldots, u_{s_j}) \in (\Sigma^*)^{s_j} \mid u_1 = u_2 = \ldots = u_{s_j}\}$. In order to ease our notations, we sometimes represent the equality relations as a partition $\{E_{q,1}, E_{q,2}, \ldots, E_{q,t}\}$ of E_q (i. e., for every $j \in [t]$, the edges of $E_{q,j}$ are subject to an equality relation), or, for simple queries, we also state which edges are required to be equal without formally stating the equality relations.

First, by the following Theorems D.1 and D.2, we will show the vertical inclusion chains

$$\begin{aligned}&[\![\text{CRPQ}]\!] \subsetneq [\![\text{ECRPQ}^{\text{er}}]\!] \subsetneq [\![\text{ECRPQ}]\!] \text{ and}\\ &[\![\cup\text{-CRPQ}]\!] \subsetneq [\![\cup\text{-ECRPQ}^{\text{er}}]\!] \subsetneq [\![\cup\text{-ECRPQ}]\!]\,,\end{aligned}$$

respectively, of the diagram of Figure 6.

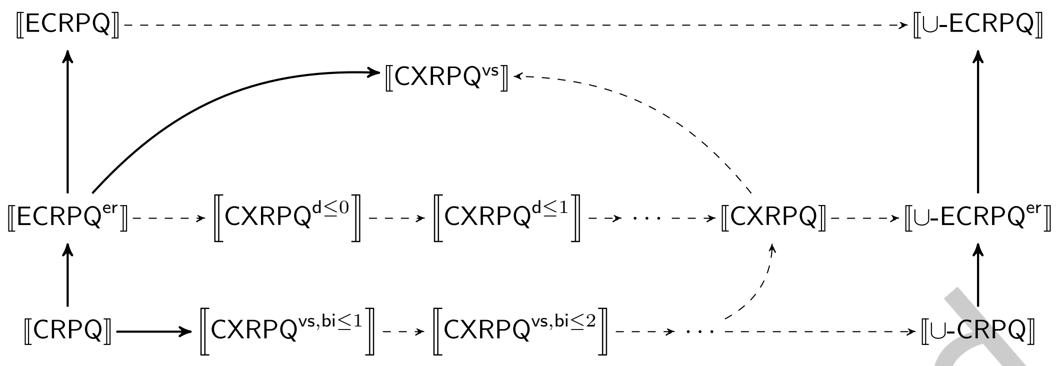


Fig. 7. Illustration of the relations between the considered classes of conjunctive path queries. A dashed or solid arrow from A to B denotes $A \subseteq B$ or $A \subsetneq B$, respectively. Note that $\mathsf{CXRPQ}^{\mathsf{depth}\leq i}$ is abbreviated as $\mathsf{CXRPQ}^{\mathsf{d}\leq i}$.

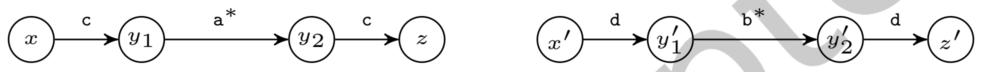


Fig. 8. Illustration of the graph-pattern for $q_{\mathsf{a}^n\mathsf{b}^n}$.

Theorem D.1. $[\![\mathsf{CRPQ}]\!] \subsetneq [\![\mathsf{ECRPQ}^{\mathsf{er}}]\!] \subsetneq [\![\mathsf{ECRPQ}]\!]$.

Proof. The inclusions follow immediately, since CRPQ can be interpreted as $\mathsf{ECRPQ}^{\mathsf{er}}$ without any equality relations, and $\mathsf{ECRPQ}^{\mathsf{er}} \subseteq \mathsf{ECRPQ}$. We shall next prove that they are proper.

Let $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$ and let $q_{\mathsf{a}^n\mathsf{b}^n}$ be the Boolean ECRPQ over Σ defined by the graph-pattern $G_{q_{\mathsf{a}^n\mathsf{b}^n}} = (V_{q_{\mathsf{a}^n\mathsf{b}^n}}, E_{q_{\mathsf{a}^n\mathsf{b}^n}})$ with

$$\begin{aligned} V_{q_{\mathsf{a}^n\mathsf{b}^n}} &= \{x, y_1, y_2, z, x', y'_1, y'_2, z'\}\,, \\ E_{q_{\mathsf{a}^n\mathsf{b}^n}} &= \{(x, \mathsf{c}, y_1), (y_1, \mathsf{a}^*, y_2), (y_2, \mathsf{c}, z), \\ &\qquad (x', \mathsf{d}, y'_1), (y'_1, \mathsf{b}^*, y'_2), (y'_2, \mathsf{d}, z')\}\,, \end{aligned}$$

and with only one equal-length relation (i. e., the relation $\{(u_1, u_2) \in (\Sigma^*)^2 \mid |u_1| = |u_2|\}$) that applies to the edges (y_1, a^*, y_2) and $(y'_1, \mathsf{b}^*, y'_2)$. See Figure 8 for an illustration. We note that $[\![q_{\mathsf{a}^n\mathsf{b}^n}]\!]$ is the set of graph-databases $\mathcal{D}$ that contain (not necessarily distinct) vertices u, v, u', v' and a path from u to v labelled with $\mathsf{c}\mathsf{a}^n\mathsf{c}$ and a path from u' to v' labelled with $\mathsf{d}\mathsf{b}^n\mathsf{d}$, respectively, for some $n \geq 0$.

Claim 1: $[\![q_{\mathsf{a}^n\mathsf{b}^n}]\!] \notin [\![\mathsf{ECRPQ}^{\mathsf{er}}]\!]$.

Proof of Claim 1: For the sake of convenience, we relabel $q_{\mathsf{a}^n\mathsf{b}^n}$ to q in the proof of the claim. We assume that there is a Boolean $q' \in \mathsf{ECRPQ}^{\mathsf{er}}$, such that $[\![q']\!] = [\![q]\!]$. Moreover, let q' be defined by a graph pattern $G_{q'} = (V_{q'}, E_{q'})$ with $E_{q'} = \{(\widehat{x}_i, \alpha_i, \widehat{y}_i) \mid i \in [m]\}$ and some equality relations.

For every $n \in \mathbb{N}$, let $\mathcal{D}_{n,n}$ be the graph-database given by two node-disjoint paths $(r_0, r_1, \ldots, r_{n+2})$ and $(s_0, s_1, \ldots, s_{n+2})$ labelled with $\mathsf{c}\mathsf{a}^n\mathsf{c}$ and $\mathsf{d}\mathsf{b}^n\mathsf{d}$, respectively. Obviously, for every $n \in \mathbb{N}$, $\mathcal{D}_{n,n} \in [\![q]\!] = [\![q']\!]$, which means that there is at least one matching morphism h for q' and $\mathcal{D}_{n,n}$. In the following, for every $n \in \mathbb{N}$, let h_n be some fixed matching morphism for q' and $\mathcal{D}_{n,n}$. By the structure of $\mathcal{D}_{n,n}$, we also know that, for every $i \in [m]$, the arc $(\widehat{x}_i, \alpha_i, \widehat{y}_i)$ is matched to some sub-path of either $(r_0, r_1, \ldots, r_{n+2})$ or $(s_0, s_1, \ldots, s_{n+2})$; more precisely, there are $\ell_i, \ell'_i \in [n+2] \cup \{0\}$ with $0 \leq \ell_i \leq \ell'_i \leq n+2$ such that either $h_n(\widehat{x}_i) = r_{\ell_i}$ and $h_n(\widehat{y}_i) = r_{\ell'_i}$, or $h_n(\widehat{x}_i) = s_{\ell_i}$

and $h_n(\widehat{y_i}) = s_{\ell'_i}$. Now let $\{C_n, D_n\}$ be a partition of $[m]$ such that $i \in C_n$ if $(\widehat{x_i}, \alpha_i, \widehat{y_i})$ is matched to some sub-path of $(r_0, r_1, \ldots, r_{n+2})$ and $i \in D_n$ if $(\widehat{x_i}, \alpha_i, \widehat{y_i})$ is matched to some sub-path of $(s_0, s_1, \ldots, s_{n+2})$. In particular, we note that $(r_0, r_1, \ldots, r_{n+2})$ only contains labels a and c, while $(s_0, s_1, \ldots, s_{n+2})$ only contains labels b and d. This means that for each single equality relation of q' with arity p that applies to a set $\{e_{j_1}, e_{j_2}, \ldots, e_{j_p}\}$ of arcs, there are three possibilities: (1) $\{j_1, j_2, \ldots, j_p\} \subseteq C_n$, (2) $\{j_1, j_2, \ldots, j_p\} \subseteq D_n$, or (3), for every $i \in [p]$, $h_n(\widehat{x_{j_i}}) = h_n(\widehat{y_{j_i}})$ (i. e., they cover paths of length 0 that are labelled with ε).

Since there is only a finite number of partitions of $[m]$ into two sets, there must be some $n_1, n_2 \in \mathbb{N}$ with $n_1 \neq n_2$ such that $C_{n_1} = C_{n_2}$ and $D_{n_1} = D_{n_2}$. We now define a morphism $h : V_{q'} \to \{r_0, r_1, \ldots, r_{n_1+2}\} \cup \{s_0, s_1, \ldots, s_{n_2+2}\}$ by setting, for every $i \in C_{n_1} = C_{n_2}$, $h(\widehat{x_i}) = h_{n_1}(\widehat{x_i})$ and $h(\widehat{y_i}) = h_{n_1}(\widehat{y_i})$, and, for every $i \in D_{n_1} = D_{n_2}$, $h(\widehat{x_i}) = h_{n_2}(\widehat{x_i})$ and $h(\widehat{y_i}) = h_{n_2}(\widehat{y_i})$. We note that h is a matching morphism for q' and $\mathcal{D}_{n_1,n_2}$. In particular, due to our observation from above, each equality relation is satisfied. Since $n_1 \neq n_2$, we have that $\mathcal{D}_{n_1,n_2} \notin [\![q']\!]$, which is a contradiction.

□ (*Claim* 1)

Let $q_{\mathsf{a}^n\mathsf{a}^n}$ be the $\mathsf{ECRPQ}^{\mathsf{er}}$ over Σ that is defined in almost the same way as $q_{\mathsf{a}^n\mathsf{b}^n}$, with the only differences that we label the arc from y'_1 to y'_2 of the graph-pattern by a^* instead of b^* and that the binary equal-length relation on (y_1, a^*, y_2) and $(y'_1, \mathsf{b}^*, y'_2)$ becomes a binary equality relation on (y_1, a^*, y_2) and $(y'_1, \mathsf{a}^*, y'_2)$.

More formally, let $q_{\mathsf{a}^n\mathsf{a}^n} \in \mathsf{ECRPQ}^{\mathsf{er}}$ over $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$ be defined by the graph-pattern $G_{q_{\mathsf{a}^n\mathsf{a}^n}} = (V_{q_{\mathsf{a}^n\mathsf{a}^n}}, E_{q_{\mathsf{a}^n\mathsf{a}^n}})$ with

$$\begin{aligned} V_{q_{\mathsf{a}^n\mathsf{a}^n}} &= \{x, y_1, y_2, z, x', y'_1, y'_2, z'\}, \\ E_{q_{\mathsf{a}^n\mathsf{a}^n}} &= \{(x, \mathsf{c}, y_1), (y_1, \mathsf{a}^*, y_2), (y_2, \mathsf{c}, z), \\ &\qquad (x', \mathsf{d}, y'_1), (y'_1, \mathsf{a}^*, y'_2), (y'_2, \mathsf{d}, z')\}, \end{aligned}$$

and only one binary equality relation that applies to the edges (y_1, a^*, y_2) and $(y'_1, \mathsf{a}^*, y'_2)$. We note that $[\![q_{\mathsf{a}^n\mathsf{a}^n}]\!]$ is the set of graph-databases $\mathcal{D}$ that contain (not necessarily distinct) vertices u, v, u', v' and a path from u to v labelled with $\mathsf{c}\mathsf{a}^n\mathsf{c}$ and a path from u' to v' labelled with $\mathsf{d}\mathsf{a}^n\mathsf{d}$, respectively, for some $n \geq 0$.

Claim 2: $[\![\widehat{q}_{\mathsf{a}^n\mathsf{a}^n}]\!] \notin [\![\mathsf{CRPQ}]\!]$.

Proof of Claim 2: We assume that there is a $q' \in \mathsf{CRPQ}$, such that $[\![q']\!] = [\![\widehat{q}_{\mathsf{a}^n\mathsf{a}^n}]\!]$. Moreover, let q' be defined by a graph pattern $G_{q'} = (V_{q'}, E_{q'})$ with $E_{q'} = \{(\widehat{x_i}, \alpha_i, \widehat{y_i}) \mid i \in [m]\}$.

We can now obtain a contradiction analogously as in to the proof of Claim 1. In fact, the argument is almost the same, but we argue with graph-databases $\mathcal{D}_{n,n}$ given by two node-disjoint paths $(r_0, r_1, \ldots, r_{n+2})$ and $(s_0, s_1, \ldots, s_{n+2})$ labelled with $\mathsf{c}\mathsf{a}^n\mathsf{c}$ and $\mathsf{d}\mathsf{a}^n\mathsf{d}$, respectively. In general, the argument is simpler, because we do not have to take care of possible equality relations.

□ (*Claim* 2)

This concludes the proof. □

Theorem D.2. $[\![\cup\text{-}\mathsf{CRPQ}]\!] \subsetneq [\![\cup\text{-}\mathsf{ECRPQ}^{\mathsf{er}}]\!] \subsetneq [\![\cup\text{-}\mathsf{ECRPQ}]\!]$.

Proof. The inclusions follow by definition. We next show that the inclusion $[\![\cup\text{-}\mathsf{ECRPQ}^{\mathsf{er}}]\!] \subseteq [\![\cup\text{-}\mathsf{ECRPQ}]\!]$ is proper. To this end, we first recall the proof of Claim 1 in the proof of Theorem D.1, which showed that for the query $q = q_{\mathsf{a}^n\mathsf{b}^n} \in \mathsf{ECRPQ} \subseteq \cup\text{-}\mathsf{ECRPQ}$ (see also Figure 8), we have that $[\![q]\!] \notin [\![\mathsf{ECRPQ}^{\mathsf{er}}]\!]$. We have demonstrated that if there is a query $q' \in \mathsf{ECRPQ}^{\mathsf{er}}$ with $\mathcal{D}_{n,n} \models q'$ for every $n \in \mathbb{N}$, then there are $n_1, n_2 \in \mathbb{N}$ with $n_1 \neq n_2$, such that q can be matched to both $\mathcal{D}_{n_1,n_1}$ and $\mathcal{D}_{n_2,n_2}$ in such a way that the partition of the edges of q' according to whether they are matched to the $\mathsf{c}\mathsf{a}^n\mathsf{c}$ path or the $\mathsf{d}\mathsf{b}^n\mathsf{d}$ path are exactly the same. This has lead to the contradiction that $\mathcal{D}_{n_1,n_2} \models q'$.

If we now instead consider a $q' \in \cup\text{-}\mathsf{ECRPQ}^{\mathsf{er}}$, then q' contains only a finite number of graph patterns $G_q^{(1)}, G_q^{(2)}, \ldots, G_q^{(\ell)}$, and, for every $n \in \mathbb{N}$, there is at least one $j \in [\ell]$, such that $G_q^{(j)}$ can be matched to $\mathcal{D}_{n,n}$. This implies again, that there are $n_1, n_2 \in \mathbb{N}$ with $n_1 \neq n_2$, such that, for some $j \in [\ell]$, $G_q^{(j)}$ can be matched to both $\mathcal{D}_{n_1,n_1}$ and $\mathcal{D}_{n_2,n_2}$ in such a way that the partition of the edges of $G_q^{(j)}$ according to whether they are matched to the $\mathsf{c}\mathsf{a}^n\mathsf{c}$ path or the $\mathsf{d}\mathsf{b}^n\mathsf{d}$ are exactly the same. Again, this leads to the contradiction that $G_q^{(j)}$ can be matched to $\mathcal{D}_{n_1,n_2}$ and therefore $\mathcal{D}_{n_1,n_2} \models q'$.

In order to show that the inclusion $[\![\cup\text{-ECRPQ}^{\text{er}}]\!] \subseteq [\![\cup\text{-ECRPQ}]\!]$ is proper, we argue analogously, but with $q = q_{\mathsf{a}^n\mathsf{a}^n} \in \text{ECRPQ}^{\text{er}} \subseteq \cup\text{-ECRPQ}^{\text{er}}$ (i. e., we extend the argument of the proof of Claim 2 of the proof of Theorem D.1 to the case of unions of queries just as it is done above with respect to Claim 1 of the proof of Theorem D.1). □

All inclusions of

$$[\![\text{ECRPQ}^{\text{er}}]\!] \subseteq [\![\text{CXRPQ}^{\text{depth}\leq 0}]\!] \subseteq [\![\text{CXRPQ}^{\text{depth}\leq 1}]\!] \subseteq \ldots \subseteq [\![\text{CXRPQ}]\!] \subseteq [\![\text{CXRPQ}^{\text{vs}}]\!]$$

follow by definition, except the first one, which is due to the following Lemma D.3. Note that this inclusion chain also implies the inclusion $[\![\text{ECRPQ}^{\text{er}}]\!] \subseteq [\![\text{CXRPQ}^{\text{vs}}]\!]$ depicted in Figure 6; its strictness follows from Lemma D.7 below.

Lemma D.3. $[\![\text{ECRPQ}^{\text{er}}]\!] \subseteq [\![\text{CXRPQ}^{\text{depth}\leq 0}]\!]$.

Proof. Let $q \in \text{ECRPQ}^{\text{er}}$ be of the form $q = \bar{z} \leftarrow G_q$ with $E_q = \{e_i = (x_i, \alpha_i, y_i) \mid i \in [m]\}$ and let $\{E_{q,1}, E_{q,2}, \ldots, E_{q,t}\}$ with $E_{q,j} = \{e_{p_1}, e_{p_2}, \ldots, e_{p_{s_j}}\}$ be the partition of E_q that represents the equality constraints. For every $j \in [t]$, we successively modify q as follows. We replace $(x_{p_1}, \alpha_{p_1}, y_{p_1})$ by $(x_{p_1}, \beta, y_{p_1})$, where β is a regular expression for $\bigcap_{i=1}^{s_j} \mathcal{L}(\alpha_{p_i})$, and, for every i with $2 \leq i \leq p_{s_j}$, we replace $(x_{p_i}, \alpha_{p_i}, y_{p_i})$ by $(x_{p_i}, \Sigma^*, y_{p_i})$. We denote the ECRPQ^{er} constructed in this way by q' and we note that q' is equivalent to q. Moreover, q' is represented by a graph $(V_{q'}, E_{q'})$ and a partition $\{E_{q,1}, E_{q,2}, \ldots, E_{q,t}\}$ such that, for every $j \in [t]$, $E_{q,j} = \{(x_{p_1}, \beta_{p_1}, y_{p_1}), (x_{p_2}, \Sigma^*, y_{p_2}), \ldots, (x_{p_{s_j}}, \Sigma^*, y_{p_{s_j}})\}$ for some regular expression β_{j_1}. We can now translate q' into a $q'' \in \text{CXRPQ}^{\text{depth}\leq 0}$ by replacing, for every $j \in [t]$, edge $(x_{j_1}, \beta_{j_1}, y_{j_1})$ by $(x_{j_1}, \mathsf{z}_j\{\beta_{j_1}\}, y_{j_1})$ and every edge $(x_{j_\ell}, \Sigma^*, y_{j_\ell})$, $2 \leq \ell \leq s_j$, by $(x_{j_\ell}, \mathsf{z}_j, y_{j_\ell})$. It can be easily verified that q'' is equivalent to q. □

The inclusion $[\![\text{ECRPQ}]\!] \subseteq [\![\cup\text{-ECRPQ}]\!]$ follows trivially by definition, whereas $[\![\text{CXRPQ}]\!] \subseteq [\![\cup\text{-ECRPQ}^{\text{er}}]\!]$ is shown by the following Lemma D.4.

Lemma D.4. $[\![\text{CXRPQ}]\!] \subseteq [\![\cup\text{-ECRPQ}^{\text{er}}]\!]$.

Proof. Let $q = \bar{z} \leftarrow G_q$ with conjunctive xregex $\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m)$. We first assume that every $\bar{\alpha}$ is variable simple and has only basic variable definitions (the general case is considered later on).

We can transform q into an equivalent $q'' \in \text{ECRPQ}^{\text{er}}$ as follows. By assumption, for every $i \in [m]$, $\alpha_i = \pi_1\pi_2\ldots\pi_t$, where each π_ℓ is a classical regular expression γ, a variable definition $\mathsf{x}\{\gamma\}$, where γ is a classical regular expression, or a variable reference x. Thus, for every $i \in [m]$, we can break up the edge labelled with α_i into a path of size t with the edge labels π_ℓ. If we do this for all α_i, then we have turned q into a $q' \in \text{CXRPQ}$, such that every edge is labelled by a classical regular expression, a variable definition over a classical regular expression, or a variable reference. For every variable x, we now do the following. We replace the edge label $\mathsf{x}\{\gamma\}$ by γ (since q' is variable simple, there can be at most one such edge label) and all edge labels x by Σ^* and add an equality constraint that applies to exactly the edges modified by this step. It can be easily seen that the thus obtained $q'' \in \text{ECRPQ}^{\text{er}}$ is equivalent to q (note that the tuple of output nodes $\bar{z}$ remains unchanged).

For the general case, where $\bar{\alpha}$ is not variable simple or has not only basic variable definitions, we proceed similarly as in the proof of Lemma 5.4. More precisely, we first carry out the nondeterministic modification described in the paragraph before Proposition 5.8, but we do this in a deterministic way, i. e., instead of nondeterministically replacing $(\gamma_1 \vee \gamma_2)$ by either γ_1 or γ_2, we create two copies, one that contains γ_1 instead of $(\gamma_1 \vee \gamma_2)$, and one that contains γ_2 instead of $(\gamma_1 \vee \gamma_2)$ (note that this corresponds to "multiplying-out" alternations). Let us describe this in more detail.

For every $i \in [m]$, if α_i is not already an alternation of valt-simple xregex, then it has the form $\alpha_i = \pi_1 \vee \pi_2 \vee \ldots \vee \pi_q$ (note that $q = 1$ is possible), where at least one π_j is not valt-free (since α_i is vstar-free, each π_j is vstar-free as well). This means that π_j has some subexpression $\gamma = (\gamma_1 \vee \gamma_2)$ and γ contains a variable definition or a variable reference. Let $\pi_{j,1}$ be obtained from π_j by replacing γ by γ_1 and let $\pi_{j,2}$ be obtained from π_j by replacing γ by γ_2. Finally, let $\alpha'_i = \pi_1 \vee \ldots \vee \pi_{j,1} \vee \pi_{j,2} \vee \ldots \vee \pi_q$ and let $\bar{\alpha}'$ be obtained from $\bar{\alpha}$ by replacing α_i by α'_i. We observe that

$$\mathcal{L}_{\mathrm{ref}}(\pi_1 \vee \pi_2 \vee \ldots \vee \pi_q) = \mathcal{L}_{\mathrm{ref}}(\pi_1 \vee \ldots \vee \pi_{j,1} \vee \pi_{j,2} \vee \ldots \vee \pi_q)\,.$$

Consequently, we have $\mathcal{L}_{\mathrm{ref}}(\alpha_i) = \mathcal{L}_{\mathrm{ref}}(\alpha'_i)$, which, by Lemma 4.4, means that $\mathcal{L}(\bar{\alpha}) = \mathcal{L}(\bar{\alpha}')$.

This means that by repeating this modification, we can change $\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ into $\bar{\alpha}' = (\alpha'_1, \alpha'_2, \ldots, \alpha'_m)$ such that, for every $i \in [m]$, $\alpha'_i = \alpha'_{i,1} \vee \alpha'_{i,2} \vee \ldots \vee \alpha'_{i,t_i}$ and every $\alpha'_{i,j}$ for $j \in [t_i]$ is variable simple. Let q' be obtained from q by replacing each α_i by α'_i. As observed above, q and q' are equivalent.

We can now transform q' into $q'_1, q'_2, \ldots, q'_r \in$ CXRPQ, such that, for every graph-database $\mathcal{D}$, $q'(\mathcal{D}) = \bigcup_{j=1}^{r} q'_j(\mathcal{D})$, and, for every $j \in [r]$, q'_j is variable simple and that satisfies that every variable has at most one definition (the latter is a consequence from the fact that q'_j is variable simple and therefore each variable definition is necessarily instantiated by every tuple of ref-words). More precisely, the q_j with $j \in [r]$ are obtained by considering, for every $i \in [m]$, all possibilities of replacing α'_i by exactly one of the $\alpha'_{i,1}, \alpha'_{i,2}, \ldots, \alpha'_{i,t_i}$, which obviously results in CXRPQ that are variable simple (since each $\alpha'_{i,\ell}$ is variable simple), and that satisfy $q(\mathcal{D}) = q'(\mathcal{D}) = \bigcup_{j=1}^{r} q'_j(\mathcal{D})$ for every graph database $\mathcal{D}$.

Since each q'_j with $j \in [r]$ is variable simple and satisfies that every variable has at most one definition, we can transform it into an equivalent q''_j that is variable simple and has only basic variable definitions, in the same way as done in the second part of the modification used in the proof of Lemma 5.4, i. e., by using the *main modification step* as explained after Proposition 5.8. In particular, we observe that we now have that $q(\mathcal{D}) = \bigcup_{j=1}^{r} q'_j(\mathcal{D}) = \bigcup_{j=1}^{r} q''_j(\mathcal{D})$ for every database $\mathcal{D}$.

Finally, since each q''_j with $j \in [r]$ is variable simple and has only basic variable definitions, each q''_j can be transformed into an equivalent $\widehat{q}_j \in \mathsf{ECRPQ}^{\mathsf{er}}$ as explained at the beginning of the proof. Consequently, $q(\mathcal{D}) = \bigcup_{j=1}^{r} \widehat{q}_j(\mathcal{D})$ for every database and for $\widehat{q}_1, \widehat{q}_2, \ldots, \widehat{q}_r \in \mathsf{ECRPQ}^{\mathsf{er}}$. Thus, $[\![q]\!] \in [\![\cup\text{-}\mathsf{ECRPQ}^{\mathsf{er}}]\!]$. □

The inclusions $[\![\mathsf{CXRPQ}^{\mathsf{vs,bi}\le k}]\!] \subseteq [\![\cup\text{-}\mathsf{CRPQ}]\!]$, for every $k \geq 1$, are due to the following Lemma D.5. Moreover, the inclusion $[\![\mathsf{CRPQ}]\!] \subseteq [\![\mathsf{CXRPQ}^{\mathsf{vs,bi}\le 1}]\!]$ follow trivially by definition and the strictness of this inclusion follows from Lemma D.6 below.

Lemma D.5. *For every $k \geq 1$, $[\![\mathsf{CXRPQ}^{\mathsf{vs,bi}\le k}]\!] \subseteq [\![\cup\text{-}\mathsf{CRPQ}]\!]$.*

Proof. Let $k \geq 1$ and let $q \in \mathsf{CXRPQ}^{\mathsf{vs,bi}\le k}$ be defined by a graph pattern $G_q = (V_q, E_q)$ with $E_q = \{(x_i, \alpha_i, y_i) \mid i \in [m]\}$, where $\bar{\alpha} \in \mathsf{CXRE}_{\Sigma,\mathcal{X}_s}$ with $\mathcal{X}_s = \{\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n\}$. Now, for every $\bar{v} \in (\Sigma^{\le k})^n$, let $q[\bar{v}]$ be a CRPQ with the property that, for every graph database $\mathcal{D}$, $q[\bar{v}](\mathcal{D}) = q^{\bar{v}}(\mathcal{D})$. Such $q[\bar{v}]$ exist due to Lemma 7.3. This directly implies that, for every graph database $\mathcal{D}$, $q(\mathcal{D}) = \bigcup_{\bar{v} \in (\Sigma^{\le k})^n} q[\bar{v}](\mathcal{D})$. Moreover, we can conclude that $[\![q]\!] = [\![q']\!]$, where $q' \in \cup$-CRPQ is defined by $q' = \bigvee_{\bar{v} \in (\Sigma^{\le k})^n} q[\bar{v}]$. □

It now only remains to prove the strictness of the inclusion $[\![\mathsf{CRPQ}]\!] \subseteq [\![\mathsf{CXRPQ}^{\mathsf{vs,bi}\le 1}]\!]$, and $[\![\mathsf{ECRPQ}^{\mathsf{er}}]\!] \subseteq [\![\mathsf{CXRPQ}]\!]$; the witnesses used for proving these claims are illustrated in Figure 9.

Lemma D.6. *There is a Boolean $q \in \mathsf{CXRPQ}^{\mathsf{vs,bi}\le 1}$ with $[\![q]\!] \notin [\![\mathsf{CRPQ}]\!]$.*

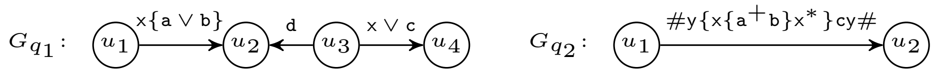


Fig. 9. The queries q_1 and q_2 for the proofs of Lemmas D.6 and D.7, respectively.

Proof. Let the Boolean $q_1 \in$ CXRPQ over $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$ be defined by the graph pattern $G_{q_1} = (V_{q_1}, E_{q_1})$ with $V_{q_1} = \{u_1, u_2, u_3, u_4\}$ and

$$E_{q_1} = \{(u_1, \alpha_1, u_2), (u_3, \alpha_2, u_2), (u_3, \alpha_3, u_4)\},$$

where $\alpha_1 = \mathsf{x}\{\mathsf{a} \vee \mathsf{b}\}$, $\alpha_2 = \mathsf{d}$ and $\alpha_3 = \mathsf{x}$ (see Figure 9). We note that since the only variable definition of q has the form $\mathsf{x}\{\mathsf{a} \vee \mathsf{b}\}$, the image size is necessarily bounded by 1, which means that we can interpret q as a normal CXRPQ without bounded image size. Thus, in order to prove the statement of the lemma, it is sufficient to show that $[\![q_1]\!] \notin [\![\text{CRPQ}]\!]$, where q_1 is interpreted as a CXRPQ without any restrictions.

For every $\sigma_1, \sigma_2 \in \Sigma$, let $\mathcal{D}_{\sigma_1,\sigma_2} = (V_{\sigma_1,\sigma_2}, E_{\sigma_1,\sigma_2})$ be a graph-database with $V_{\sigma_1,\sigma_2} = \{v_1, v_2, v_3, v_4\}$ and

$$E_{\sigma_1,\sigma_2} = \{(v_1, \sigma_1, v_2), (v_3, \mathsf{d}, v_2), (v_3, \sigma_2, v_4)\}.$$

We note that $\mathcal{D}_{\sigma_1,\sigma_2} \models q_1$ for every $\sigma_1, \sigma_2 \in \{\mathsf{a}, \mathsf{b}\}$ with $\sigma_1 = \sigma_2$. We assume that there is a $q' \in$ CRPQ with $q' \equiv q_1$, defined by the graph pattern $G_{q'} = (V_{q'}, E_{q'})$ with $E_{q'} = \{(x_i, \beta_i, y_i) \mid i \in [m]\}$.

If, for some $i \in [m]$, there is no $w_i \in \mathcal{L}(\beta_i)$ with $|w_i|_{\mathsf{a}} = 0$, then $\mathcal{D}_{\mathsf{b},\mathsf{b}} \not\models q'$, which, since $\mathcal{D}_{\mathsf{b},\mathsf{b}} \models q_1$, is a contradiction. Therefore, we can assume that, for every $i \in [m]$, there is some $w_i \in \mathcal{L}(\beta_i)$ with $|w_i|_{\mathsf{a}} = 0$ (note that $w_i = \varepsilon$ is possible). Next, we consider a matching morphism h for q' and $\mathcal{D}_{\mathsf{a},\mathsf{a}}$, which, since $\mathcal{D}_{\mathsf{a},\mathsf{a}} \models q_1$, must exist. Let $A \subseteq [m]$ be exactly the set of $i \in [m]$ with $h(x_i) = v_1$ and $h(y_i) = v_2$. If $A = \emptyset$, then we can conclude that the edge (v_1, a, v_2) is not part of any of the paths between some $h(x_i)$ and $h(y_i)$ that are necessary for h being a matching morphism (this is due to the fact that in $\mathcal{D}_{\mathsf{a},\mathsf{a}}$ there are no paths of length strictly greater than 2). Consequently, we can remove (v_1, a, v_2) from $\mathcal{D}_{\mathsf{a},\mathsf{a}}$ in order to obtain a graph database $\mathcal{D}'$, such that h would still be a matching morphism for q' and $\mathcal{D}'$. This, however, is a contradiction, since $\mathcal{D}' \not\models q_1$. Thus, $A \neq \emptyset$. Now let $\mathcal{D}'_{\mathsf{a},\mathsf{a}}$ be the graph database obtained from $\mathcal{D}_{\mathsf{a},\mathsf{a}}$, by deleting the edge (v_1, a, v_2) and, for every $i \in A$ with $w_i \neq \varepsilon$, adding a path labelled with w_i from v_1 to v_2. We now slightly change the matching morphism h for $\mathcal{D}_{\mathsf{a},\mathsf{a}}$ into a matching morphism h' for $\mathcal{D}'_{\mathsf{a},\mathsf{a}}$: for every $i \notin A$ and for every $i \in A$ with $w_i \neq \varepsilon$, we set $h'(x_i) = h(x_i)$ and $h'(y_i) = h(y_i)$; for every $i \in A$ with $w_i = \emptyset$, we set $h'(x_i) = h'(y_i) = h(x_i)$. We note that h' is a matching morphism for q' and $\mathcal{D}'_{\mathsf{a},\mathsf{a}}$, since, for every $i \in A$ with $w_i \neq \varepsilon$, there is a path from v_1 to v_2 labelled with $w_i \in \mathcal{L}(\beta_i)$, and, for every $i \in A$ with $w_i = \varepsilon$, there is a path from v_1 to v_1 labelled with $w_i = \varepsilon$. Furthermore, as already observed above, the deleted edge (v_1, a, v_2) was exclusively covered by edges (x_i, β_i, y_i) with $i \in A$. However, since $|w_i|_{\mathsf{a}} = 0$ for every $i \in A$, we have that $\mathcal{D}'_{\mathsf{a},\mathsf{a}} \not\models q_1$, which is a contradiction. □

Lemma D.7. *There is a Boolean $q \in \text{CXRPQ}^{\text{vs}}$ such that $[\![q]\!] \notin [\![\text{ECRPQ}^{\text{er}}]\!]$.*

Proof. Let q_2 be defined by a graph pattern with just a single edge (u_1, β, u_2) with $\beta = \#\mathsf{y}\{\mathsf{x}\{\mathsf{a}^+\mathsf{b}\}\mathsf{x}^*\}\mathsf{c}\mathsf{y}\#$ (see Figure 9). We note that $\mathcal{D} \models q_2$ if and only if $\mathcal{D}$ contains a path labelled with $\#(\mathsf{a}^{n_1}\mathsf{b})^{n_2}\mathsf{c}(\mathsf{a}^{n_1}\mathsf{b})^{n_2}\#$ for some $n_1, n_2 \geq 1$. Let us assume that there is some $q' \in \text{ECRPQ}^{\text{er}}$ with $q_2 \equiv q'$ and q' is defined by a graph pattern $G_{q'} = (V_{q'}, E_{q'})$ with $E_{q'} = \{(x_i, \alpha_i, y_i) \mid i \in [m]\}$ and some equality relations. Moreover, for every $i \in [m]$, let p_i be the pumping lemma constant of $\mathcal{L}(\alpha_i)$ and let $p = \max\{p_i \mid i \in [m]\}$. We consider the graph database $\mathcal{D} = (V_{\mathcal{D}}, E_{\mathcal{D}})$ with $V_{\mathcal{D}} = \{v_0, v_1, \ldots, v_t\}$, where $t = 2(p^2m + pm) + 3$ and $(v_0, v_1, \ldots, v_t)$ is a path labelled with $\#(\mathsf{a}^p\mathsf{b})^{pm}\mathsf{c}(\mathsf{a}^p\mathsf{b})^{pm}\#$.

Since $\mathcal{D} \models q'$, there is a matching morphism h for q' and $\mathcal{D}$ with some matching words $(w_1, w_2, \ldots, w_m)$, such that, for every $i \in [m]$, $h(x_i) = v_{j_i}$ and $h(y_i) = v_{j'_i}$ with $0 \leq j_i \leq j'_i \leq t$. We now partition $[m]$ into

$S = \{i \mid j'_i - j_i < 2p + 1\}$ and $L = [m] \setminus S$, i. e., (x_i, α_i, y_i) is matched to a *long* sub-path of $(v_0, v_1, \ldots, v_t)$ of length at least $2p + 1$ if $i \in L$ and to a *short* sub-path of $(v_0, v_1, \ldots, v_t)$ of length strictly less than $2p + 1$ otherwise.

If there is an arc (v_r, σ, v_{r+1}) of the path $(v_0, v_1, \ldots, v_t)$ that is not covered by some (x_i, α_i, y_i) (i. e., for every $i \in [m]$, it is not the case that $j_i \leq r < j'_i$), then we can contract nodes v_r and v_{r+1} and h is still a matching morphism for q' and the thus modified graph database $\mathcal{D}'$, which is not in $[\![q_2]\!]$ anymore. This can be seen by observing that removing a single symbol from a word $w \in \mathcal{L}(\beta)$ yields a word that is not in $\mathcal{L}(\beta)$ anymore. Therefore, we can assume that every arc (v_r, σ, v_{r+1}) of $\mathcal{D}$ is covered by some (x_i, α_i, y_i), i. e., $j_i \leq r < j'_i$. Since, for every $i \in S$, at most $2p$ edges can be covered by (x_i, α_i, y_i), we also know that $L \neq \emptyset$ (since otherwise not all arcs are covered).

We now modify $\mathcal{D}$ as follows. For every $i \in [m]$, we add a *shortcut* from v_{j_i} to $v_{j'_i}$, which is a new path of the same length and with the same label as the path $(v_{j_i}, v_{j_i+1}, \ldots, v_{j'_i})$. In particular, we note that for every $i \in [m]$, the labels of the shortcuts are identical to the matching words of h.

We now pump some of the shortcuts depending on the equality relation of q' as follows. Assume that $A \subseteq [m]$ represents an equality relation of q', i. e., exactly the edges $\{(x_i, \alpha_i, y_i) \mid i \in A\}$ are subject to the equality relation. This also means that either all (x_i, α_i, y_i) with $i \in A$ cover a short path, i. e., $A \subseteq S$, or all (x_i, α_i, y_i) with $i \in A$ cover a long path, i. e., $A \subseteq L$. Moreover, as observed above, $L \neq \emptyset$, so there is at least one such equality relation A (note that we assume that the equality relations are represented by a partition of the edge-set, i. e., every edge is subject to exactly one equality relation, possibly a unary one).

Recall that $(w_1, w_2, \ldots, w_m)$ are the matching words for h. Thus, for every $i \in A$, w_i is the label of the sub-path $(v_{j_i}, v_{j_i+1}, \ldots, v_{j'_i})$. Since h is a matching morphism and since we have the equality relation represented by A, we know that, for some u, we have $u = w_i$ for every $i \in A$; moreover, the corresponding shortcuts for edges (x_i, α_i, y_i) with $i \in A$ are also all labelled with u. Since $A \subseteq L$, we have $|u| \geq 2p + 1$, which means that $u = u'\mathtt{a}^p u''$. Hence, for every $i \in A$, there is a d_i such that $u'\mathtt{a}^{p+\delta d_i}u'' \in \mathcal{L}(\alpha_i)$ for every $\delta \geq 0$. This means that, for every $i \in A$, $w' = u'\mathtt{a}^{p+d}u'' \in \mathcal{L}(\alpha_i)$, where $d = \Pi_{i \in A} d_i$. Consequently, we can pump all shortcuts for the edges $(x_i, \alpha_, y_i)$ with $i \in A$ by the factor $\mathtt{a}^d$, i. e., we replace them by paths of length $|u| + d$ labelled by w'. We repeat this pumping-step with respect to all equality relations that refer to edges that cover long paths. After this modification, we have the property that in the tuple $(w_1, w_2, \ldots, w_m)$ of matching words for h, we can arbitrarily replace some w_i by the label of the corresponding shortcut for (x_i, α_i, y_i) (regardless of whether it has been pumped or not) and still h is a matching morphism with respect to this modified tuple of matching words.

We now choose an arbitrary edge $(v_\ell, \sigma, v_{\ell+1})$ of the original path $(v_0, v_1, \ldots, v_t)$, which is only covered by edges (x_i, α_i, y_i) with $i \in L$, which means that their shortcuts have been pumped. Such an edge must exist, since otherwise all edges are covered by edges from S, which is not possible. Then, we delete this edge and we denote the obtained graph database by $\mathcal{D}'$. After this modification, due to the shortcuts, the matching morphism h must still be a valid matching morphism for q' and $\mathcal{D}$, i. e., $\mathcal{D} \models q'$. We now conclude the proof by showing that $\mathcal{D}' \not\models q_2$, which clearly is a contradiction.

For $\mathcal{D}' \models q_2$, there must be a path in $\mathcal{D}'$ that is labelled by a word $\#(\mathtt{a}^{n_1}\mathtt{b})^{n_2}\mathtt{c}(\mathtt{a}^{n_1}\mathtt{b})^{n_2}\#$ for some $n_1, n_2 \geq 1$. We note that this is only possible for paths from v_0 to v_t. Now let us consider an arbitrary path from v_0 and v_t in $\mathcal{D}'$. Since we deleted the edge $(v_\ell, \sigma, v_{\ell+1})$, this path must use at least one shortcut that corresponds to an edge (x_i, α_i, y_i) with $j_i \leq \ell < j'_i$. However, by our choice of $(v_\ell, \sigma, v_{\ell+1})$, all such shortcuts have been pumped, which means that the path is labelled with a word $\widehat{w}$ that can be obtained from $\#(\mathtt{a}^p\mathtt{b})^{pm}\mathtt{c}(\mathtt{a}^p\mathtt{b})^{pm}\#$ by pumping some unary factors over $\mathtt{a}$. Furthermore, it is not possible that all maximal unary factors over $\mathtt{a}$, i. e., factors of the form $\#\mathtt{a}^p\mathtt{b}$, $\mathtt{b}\mathtt{a}^p\mathtt{b}$ or $\mathtt{c}\mathtt{a}^p\mathtt{b}$, have been pumped, since the considered path can take at most m shortcuts. □