Definability problems for graph query languages [*]

Timos Antonopoulos
Hasselt University and
Transnational University of
Limburg
timos.antonopoulos@uhasselt.be

Frank Neven
Hasselt University and
Transnational University of
Limburg
frank.neven@uhasselt.be

Frédéric Servais
Hasselt University and
Transnational University of
Limburg
frederic.servais@uhasselt.be

ABSTRACT

Given a graph, a relation on its nodes, and a query language $\mathcal{Q}$ of interest, we study the $\mathcal{Q}$-definability problem which amounts to deciding whether there exists a query in $\mathcal{Q}$ defining precisely the given relation over the given graph. Previous research has identified the complexity of FO- and CQ-definability. In this paper, we consider the definability problem for regular paths and conjunctive regular path queries (CRPQs) over labelled graphs.

1. INTRODUCTION

Banchilhon [2] and Paredaens [27], showed that a relation S whose active domain is contained in the active domain of a database D, is definable over D in the relational algebra (or, equivalently in first-order logic (FO)) if and only if every automorphism of D is also an automorphism of S. This seminal result is often referred to by the name of BP-completeness as Codd termed a query language complete when it has the same expressiveness as the relation algebra (or, equivalently, first-order logic). The latter language-independent characterization of relational completeness, provides a means to decide whether a given relation S is definable over a given database D: simply guess an automorphism on D that does not hold in S. For a query language $\mathcal{Q}$ of interest, we define $\mathcal{Q}$-definability as the problem to decide whether for a database D and a relation S over its active domain, there exists a query in $\mathcal{Q}$ defining precisely S over D. By the argument given above, FO-definability is in CONP. Moreover, Fletcher at al. show that the extension of the definability problem which requires to find an FO-formula consistent with a given *set* of pairs $\{(D_1, S_1), \ldots, (D_n, S_n)\}$ is co-graph-isomorphism hard [16]. Definability as defined above has been studied for the conjunctive queries (CQs) as well and has been shown to be CONEXPTIME-complete by Willard [31].[1] Motivated by the recent renewed interest in path query languages for graphs (e.g., [25, 5, 24, 4, 3, 15]), we study the complexity of the definability problem for graph query languages.More precisely, we study regular paths [26, 25] and conjunctive regular path queries [9, 17, 14].

The paper is structured as follows. In Section 2, we introduce the necessary definitions and formally define the definability problem. In Section 3, we consider binary queries defining regular paths in graphs. We argue that in the general case, regular path definability reduces to definability by a finite language and show that the corresponding decision problem is PSPACE-complete (even when restricting to deterministic graphs). Then, we turn to simple path semantics and classes which do not contain all finite languages. In particular, we consider single-occurrence regular [7, 6, 10] and abbreviated regular expressions [26]. We show that all corresponding decision problems become NP-complete. In Section 4, we consider the more expressive conjunctive regular path queries (CRPQs). We obtain that definability for chain and linear CRPQs is PSPACE-complete, for acyclic CRPQs is PSPACE-hard and in EXPTIME, and definability is in EXPSPACE for general CRPQs. Finally, we show that definability for unions of CRPQs is CONP-complete. In Section 5, we discuss the relationship between CQ-definability and CRPQ-definability. We discuss related work in Section 6 and conclude in Section 7.

2. DEFINITIONS

We introduce the necessary definitions concerning regular expressions, automata, graphs, and queries, and we define the problem central to this paper.

In the following, Σ always denotes a finite alphabet. A word over Σ is a sequence $a_1 \cdots a_n$ of Σ-symbols. The concatenation of two words $w_1 = a_1 \cdots a_n$ and $w_2 = b_1 \cdots b_m$, denoted $w_1 \cdot w_2$, is defined as the word $a_1 \cdots a_n \cdot b_1 \cdots b_n$. As usual, the empty word is denoted by ε. A language over Σ is a set of words. By Σ^*, we denote the language of all finite words over Σ.

A *regular expression* is defined by the following grammar:

$$r := \varepsilon \mid a \in \Sigma \mid r_1 + r_2 \mid r_1 \cdot r_2 \mid r^*.$$

The *language* $\mathcal{L}(r)$ defined by r is defined inductively as

[*] We acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Com- mission, under the FET-Open grant agreement FOX, number FP7-ICT-233599.

[1] We refer to the section on related work for a more elaborate discussion.

EDBT/ICDT '13, March 18 - 22 2013, Genoa, Italy
Copyright 2013 ACM 978-1-4503-1598-2/13/03 ...$15.00.

follows:

$$\begin{aligned}
\mathcal{L}(\varepsilon) &= \{\varepsilon\},\\
\mathcal{L}(a) &= \{a\}, \text{ where } a \in \Sigma,\\
\mathcal{L}(r_1 + r_2) &= \mathcal{L}(r_1) \cup \mathcal{L}(r_2),\\
\mathcal{L}(r_1 \cdot r_2) &= \{w_1 \cdot w_2 \mid w_1 \in \mathcal{L}(r_1), w_2 \in \mathcal{L}(r_2)\},\\
\mathcal{L}(r^*) &= \{w_1 \cdots w_n \mid n \in \mathbb{N}, w_i \in \mathcal{L}(r) \text{ for } i \leq n\}.
\end{aligned}$$

We also use the abbreviations $r?$ and r^+ to denote the expressions $r + \varepsilon$ and rr^*. Sometimes, we abuse notation and write w to denote the word w as well as the regular expression for which $\mathcal{L}(w) = \{w\}$. The *length* of a regular expression is the size of its word representation including operators and parenthesis.

A *nondeterministic finite word automaton (NFA)* A is a tuple $(Q, \Sigma, \delta, I, F)$, where Q is a finite set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states and $\delta : Q \times \Sigma \to 2^Q$ is the transition function. A *run* of A on a word $w = a_1 \cdots a_n$ is a sequence of states $q_0 q_1 \cdots q_n$ such that $q_i \in \delta(q_{i-1}, a_i)$ for each $i \in [1, n]$ and $q_0 \in I$. A run on w is *accepting* if $q_n \in F$, and a word w is *accepted* by A, if there exists an accepting run of A on w. The *size* $|A|$ of A is its total number of states and transitions. By $\mathcal{L}(A)$ we denote the set of words accepted by A. A *deterministic finite word automaton (DFA)* A is an NFA $(Q, \Sigma, \delta, I, F)$ where I is a singleton, and $\delta(q, \sigma)$ is a singleton or the empty set for all $q \in Q$ and all $\sigma \in \Sigma$.

As usual, a word language L is *regular* if there is an NFA A with $\mathcal{L}(A) = L$. We denote by $\mathsf{REG}(\Sigma)$ the set of regular languages over Σ. Sometimes we leave Σ implicit and just write REG.

The *inclusion problem for NFAs* is to decide whether $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ for two given NFAs A and B. Given DFAs $A_1, \ldots, A_n$, the *emptiness of intersection problem for DFAs* is to decide whether $\bigcap_{i=1}^{n} \mathcal{L}(A_i) = \emptyset$. Similarly, the *universality of union problem for DFAs* is to decide whether $\bigcup_{i=1}^{n} \mathcal{L}(A_i) = \Sigma^*$.

We make use of the following results. Note that (3) follows directly from (2).

Theorem 2.1. *1. The inclusion problem for NFAs is* PSPACE*-complete. [28]*

2. The emptiness of intersection problem for DFAs is PSPACE*-complete. [23]*

3. The universality of union problem for DFAs is PSPACE*-complete.*

We consider finite labelled directed *graphs* $G = (V_G, E_G)$, where V_G is the set of nodes and $E_G \subseteq V_G \times \Sigma \times V_G$ is the set of labelled edges. We also write $u \xrightarrow{\sigma}_G v$ to denote $(u, \sigma, v) \in E_G$. We abuse notation and also write $v \xrightarrow{\varepsilon}_G u$. The latter only holds when $u = v$. For $w \in \Sigma^*$, we denote by $u \xrightarrow{w}_G v$ the existence of a path from u to v labelled with w. That is, there is a sequence of nodes $v_1, \ldots, v_n \in V_G$ such that $v_1 = u$, $v_n = v$, for all $i < n$, $v_i \xrightarrow{\sigma_i}_G v_{i+1}$, and $w = \sigma_1 \ldots \sigma_{n-1}$. By $G \uplus G'$, we denote the disjoint union of the graphs G and G'.

Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two graphs. A *homomorphism* from G to H is a mapping $h : V_G \to V_H$ such that if $u \xrightarrow{\sigma}_G v$ then $h(u) \xrightarrow{\sigma}_H h(v)$. An *endomorphism* is an homomorphism from a graph onto itself.

Given a graph G, two nodes u and v, and a regular expression r, the *evaluation problem* is the problem to decide whether there is a $w \in \mathcal{L}(r)$, such that $u \xrightarrow{w}_G v$. The following theorem seems to belong to folklore and is for instance mentioned in [26]:

Theorem 2.2. *The evaluation problem for regular expressions over graphs is in* PTIME *in the size of the graph and the regular expression.*

In the following, we leave n as an implicit natural number. An *n-ary query* q is a function mapping graphs to n-ary relations over their nodes. We only consider queries which are closed under isomorphism.

For a class of n-ary queries $\mathcal{Q}$, a graph $G = (V_G, E_G)$, and a set $S \subseteq V_G^n$, we say that (G, S) is *$\mathcal{Q}$-definable* if there exists a query $q \in \mathcal{Q}$ such that $q(G) = S$. We are now ready to define the problem central to this paper.

Definition 2.3. *Let $\mathcal{Q}$ be a class of n-ary queries and $\mathcal{G}$ be a class of graphs. Then* $\mathrm{DEF}(\mathcal{Q}, \mathcal{G})$ *is the problem to decide whether (G, S) is $\mathcal{Q}$-definable for a graph $G \in \mathcal{G}$ and $S \subseteq V_G^n$.*

When $\mathcal{G}$ is the class of all graphs, then we denote $\mathrm{DEF}(\mathcal{Q}, \mathcal{G})$ simply by $\mathrm{DEF}(\mathcal{Q})$.

We observe the following lemma which says that lower bounds (resp. upper bounds) carry over when the class of graphs under consideration grows (resp. shrinks):

Lemma 2.4. *Let $\mathcal{C}$ be a complexity class and let $\mathcal{G}$ and $\mathcal{G}'$ be two classes of graphs such that $\mathcal{G} \subseteq \mathcal{G}'$. Then for any class of queries $\mathcal{Q}$, $\mathcal{C}$-hardness of* $\mathrm{DEF}(\mathcal{Q}, \mathcal{G})$ *implies $\mathcal{C}$-hardness of* $\mathrm{DEF}(\mathcal{Q}, \mathcal{G}')$ *and membership of* $\mathrm{DEF}(\mathcal{Q}, \mathcal{G}')$ *in $\mathcal{C}$ implies membership of* $\mathrm{DEF}(\mathcal{Q}, \mathcal{G})$ *in $\mathcal{C}$.*

Note that the above lemma does not extend to classes of queries.

3. PATHS

In this section, we consider binary queries defining paths in graphs. Any language $R \subseteq \Sigma^*$ can define a binary query as follows. For a graph G, define $q_R(G)$ as the set $\{(u, v) \mid \exists w \in R, u \xrightarrow{w}_G v\}$. Any class of languages $\mathcal{L}$ therefore corresponds to a class of binary queries. We abuse notation and use $\mathcal{L}$ to refer to the class of languages as well as to the corresponding class of binary queries.

Let $\mathcal{F}$ be the class of all finite languages. We argue next that $\mathcal{L}$-definability reduces to $\mathcal{F}$-definability for any class of languages $\mathcal{L}$ containing all finite languages. Intuitively, the proposition holds because definability is a property defined over a single finite graph.

Proposition 3.1. *Let $\mathcal{L}$ be a class of languages containing $\mathcal{F}$. Then for every graph G and set $S \subseteq V_G \times V_G$, (G, S) is $\mathcal{L}$-definable iff (G, S) is $\mathcal{F}$-definable.*

Proof. The *if* direction is immediate. For the converse direction, assume there is an $R \in \mathcal{L}$ with $q_R(G) = S$. For every $(u, v) \in S$ choose a $w_{u,v} \in R$ with $u \xrightarrow{w_{u,v}}_G v$. Then, define F as the set $\{w_{u,v} \mid (u, v) \in S\}$. Clearly, $q_F(G) = S$. As F is finite, (G, S) is $\mathcal{F}$-definable. □

3.1 $\mathcal{F}$-definability

Denote by $L_G(u, v)$ the set of paths from u to v. That is, define $L_G(u, v) = \{w \mid u \xrightarrow{w}_G v\}$. The next lemma shows that (G, S) is $\mathcal{F}$-definable if for every pair $(u, v) \in S$ there is at least one word in $L_G(u, v)$ which selects no pair outside S.

LEMMA 3.2. *For a graph $G = (V_G, E_G)$ and $S \subseteq V_G \times V_G$, (G, S) is $\mathcal{F}$-definable iff*

$$L_G(u, v) \not\subseteq \bigcup_{(x,y)\notin S} L_G(x, y),$$

for all pairs $(u, v) \in S$.

PROOF. For the *if* direction, suppose that for all pairs $(u, v) \in S$, there is a word

$$w_{(u,v)} \in L_G(u, v) \setminus (\bigcup_{(x,y)\notin S} L_G(x, y)).$$

Then, let $R = \{w_{(u,v)} \mid (u, v) \in S\}$. Now, it follows that $q_R(G) = S$, and (G, S) is $\mathcal{F}$-definable. For the *only if* direction, suppose that (G, S) is $\mathcal{F}$-definable. Then there is a finite language R in $\mathcal{F}$ such that $q_R(G) = S$. Therefore, for each pair $(u, v) \in S$, there is at least one word $w \in R$, such that $w \in L_G(u, v)$ but $w \notin L_G(x, y)$ for any pair $(x, y) \notin S$. □

As every $L_G(u, v)$ can be represented by an NFA, the previous lemma shows that $\mathcal{F}$-definability reduces to containment testing of NFAs which is known to be in PSPACE. In the following Theorem, we obtain PSPACE-hardness already when graphs are required to be 'deterministic' (as, for instance, considered in [8]). A graph G is *deterministic* when for every node there is at most one outgoing edge for every label. That is, $u \xrightarrow{\sigma}_G v$ and $u \xrightarrow{\sigma}_G v'$ imply $v = v'$. Denote by $\mathcal{D}$ the class of deterministic graphs.

THEOREM 3.3. *The problems* $\mathrm{DEF}(\mathcal{F})$ *and* $\mathrm{DEF}(\mathcal{F}, \mathcal{D})$ *are* PSPACE-*complete.*

PROOF. We decompose the proof in two steps. First we show that $\mathrm{DEF}(\mathcal{F})$ is in PSPACE; then, we show that the problem $\mathrm{DEF}(\mathcal{F}, \mathcal{D})$ is PSPACE-hard. The result then follows from Lemma 2.4.

(1) We witness membership in PSPACE by a reduction to the inclusion problem for NFAs.

Given a graph $G = (V, E)$ and $S \subseteq V \times V$, we construct for each pair $(u, v) \in V \times V$, the automata $A_{u,v} = (V, \Sigma, \delta_E, \{u\}, \{v\})$ where $(x, \sigma, y) \in \delta_E$ iff $(x, \sigma, y) \in E$. Then, $\mathcal{L}(A_{u,v}) = L_G(u, v)$. By Lemma 3.2, (G, S) is $\mathcal{F}$-definable iff for every $(u, v) \in S$,

$$\mathcal{L}(A_{u,v}) \not\subseteq \bigcup_{(x,y)\notin S} \mathcal{L}(A_{x,y}).$$

As $\bigcup_{(x,y)\notin S} \mathcal{L}(A_{x,y})$ can be represented by an NFA of size linear in the sum of the size of the $A_{x,y}$'s, the result follows by Theorem 2.1.

(2) For hardness in the case of a deterministic graph, we reduce from the universality of union problem for DFAs, which is hard for PSPACE by Theorem 2.1. Let s, f be two new symbols not in Σ. Given n DFAs $D_1, \ldots, D_n$, we construct G as the disjoint union of $G_1, \ldots, G_n$, and G_*. Here, for each i, G_i is the graph obtained from D_i by adding a new edge, labelled with s, from some new node u_i to the initial state, and by adding a new edge, labelled with f, from each of the final states to some new node v_i. We refer to u_i and v_i as the initial and final node in G_i. Then, $L_{G_i}(u_i, v_i) = \{swf \mid w \in \mathcal{L}(D_i)\}$. Furthermore, let G_* be the graph 'defining' all words of the form swf for $w \in \Sigma^*$. That is, define $G_* = (V, E)$, where $V = \{v_1, v_2, v_3\}$, $E = \{(v_1, s, v_2), (v_2, f, v_3)\} \cup \{(v_2, \sigma, v_2)\}_{\sigma\in\Sigma}$. Then, $L_G(v_1, v_3) = \{swf \mid w \in \Sigma^*\}$. Finally, let $S = \{(v_1, v_3)\}$.

By Lemma 3.2, the instance (G, S) is $\mathcal{F}$-definable if and only if $L_G(v_1, v_3) \not\subseteq \bigcup_{(x,y)\notin S} L_G(x, y)$. Since each word in $L_G(v_1, v_3)$ starts with s and ends with f, the latter condition is equivalent to $L_G(v_1, v_3) \not\subseteq \bigcup_{i=1}^{n} L_G(u_i, v_i)$ where u_i and v_i are the initial and final node of G_i. But, this last condition is true if and only if $\Sigma^* \not\subseteq \bigcup_{i=1}^{n} \mathcal{L}(D_i)$, i.e., if and only if the union of the DFA is not universal. □

Combining Theorem 3.3 with Proposition 3.1, we obtain that $\mathrm{DEF}(\mathsf{REG}, \mathcal{D})$ is PSPACE-complete.

COROLLARY 3.4. *The problem* $\mathrm{DEF}(\mathsf{REG})$ *and the problem* $\mathrm{DEF}(\mathsf{REG}, \mathcal{D})$ *are* PSPACE-*complete.*

3.2 Single-occurrence regular expressions

The PSPACE-hardness result obtained in Theorem 3.3 already holds for every subclass of the regular languages containing $\mathcal{F}$. In an effort to lower the complexity, we now consider a class of regular expressions not containing $\mathcal{F}$. Single-occurrence regular expressions (SOREs) are regular expressions where every Σ-symbol can occur at most once. For instance, $a((b + c)^+)d)^*?$ is a SORE, while $a(a + b)^*$ is not. SOREs have been considered in several papers (e.g., [7, 6, 10]). Denote by SORE($\cdot$) the fragment of SOREs that only use concatenation. Each expression in this fragment defines a single word.

THEOREM 3.5. DEF(*SORE*) *and* DEF(*SORE*($\cdot$)) *are* NP-*complete.*

PROOF. (1) Let us first show that DEF(SORE) is NP-complete. In the proof of Theorem 3.1 in [6], it is shown that every SORE over Σ is equivalent to one of size at most $10|\Sigma|$. Therefore, to show membership in NP, one simply guesses a SORE r of size at most $10|\Sigma|$ and verifies that $q_{\mathcal{L}(r)} = S$. By Theorem 2.2, the latter verification is in PTIME.

We next show NP-hardness through a reduction from 3-SAT. Let X be a set of variables and let φ be an instance of 3-SAT with variables in X. That is, $\varphi = (\alpha_{11} \vee \alpha_{12} \vee \alpha_{13}) \wedge \cdots \wedge (\alpha_{n1} \vee \alpha_{n2} \vee \alpha_{n3})$, where α_{ij} is a literal of the form x or $\neg x$ for some variable $x \in X$.

We construct a graph G and a relation $S \subseteq V_G \times V_G$ such that there is a SORE r with $q_{\mathcal{L}(r)}(G) = S$ if and only if φ is satisfiable. Define $\Sigma = \{a_{ij} \mid 0 < i \leq n \wedge 0 < j \leq 3\} \uplus \{s, f\}$. Then, G is defined as the disjoint union of G_φ and G_x for every $x \in X$. Here, G_φ represents all possible satisfying truth assignments for the formula φ, while each G_x ensures consistency. In particular, G_x ensures that in such a truth assignment the variable x can not be assigned both the value true and false.

The graph G_φ is illustrated in Figure 1. Intuitively, every path from s to f assigns the value true to one literal a_{i1}, a_{i2}, or a_{i3} by traversing the corresponding edge, for every $i \leq n$. Note that every such path starts and ends with the symbol s and f, respectively.

Figure 2 illustrates G_x. Here, an edge labelled with a set denotes multiple edges where each edge is labelled with a distinct element of the set. Figure 2 uses the sets P_x and N_x which contain the positive and negative occurrences of variable x in φ. That is, define $P_x = \{a_{ij} \mid \alpha_{ij} = x\}$ and $N_x = \{a_{ij} \mid \alpha_{ij} = \neg x\}$. Intuitively, when a path from s to f

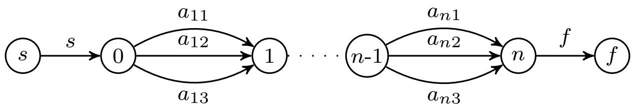


Figure 1: The graph G_φ representing all possible truth assignments for the formula φ.

in G_φ contains a symbol $a \in P_x$ and another $a' \in N_x$, then that path is conflicting and defines an inconsistent valuation. The purpose of G_x is to define these conflicting paths. The upper half of the graph defines paths where an occurrence of N_x precedes an occurrence of P_x, while the lower half of the graph defines paths where an occurrence of P_x precedes an occurrence of N_x.

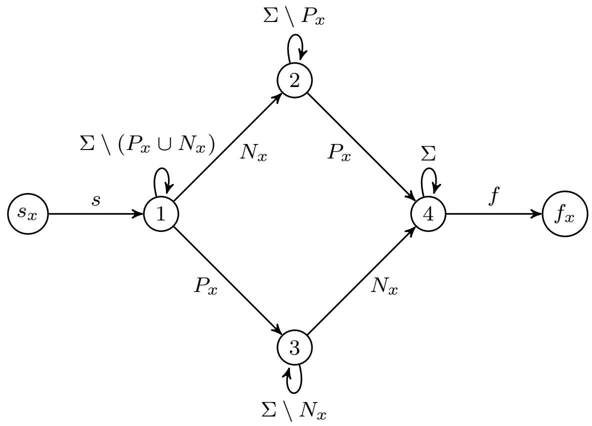


Figure 2: The graph G_x ensures that the valuations are consistent with regards to the variable x.

The set $S = \{(s, f)\}$ now contains a single pair. Assume (G, S) is SORE-definable. Then, there is a SORE r with $q_{\mathcal{L}(r)}(G) = S$. In particular, there is a word $w \in \mathcal{L}(r)$ such that $s \xrightarrow{w}_G f$ and $s_x \not\xrightarrow{w}_G f_x$ for every $x \in X$. This means that w encodes a satisfying truth assignment for φ. Conversely, assume φ is satisfiable. For every clause i, pick one literal α_{ij_i} which is assigned true. Then, let $w = sa_{1j_1} \cdots a_{nj_n} f$. Now, $q_{\mathcal{L}(w)} = \{(s, f)\}$ and (G, S) is SORE-definable.

(2) The NP-algorithm for DEF(SORE(·)) is the same as the one in (1). For the lower bound it suffices to remark that for the graph G and the relation S constructed in (1), (G, S) is SORE(·)-definable iff (G, S) is SORE-definable. □

3.3 Simple path semantics

Simple path semantics enjoys renewed attention due to its inclusion at the core of the semantics of SPARQL, see for example [25]. Recall that a path is called simple when it contains no repetition of nodes. We consider definability in the context of simple paths and show that the complexity drops from PSPACE to NP. This is in contrast to the evaluation problem which is harder for the simple path semantics than for the standard semantics. Indeed, for a regular expression r, using Theorem 2.2, computing $q_r(G)$ can be done in time polynomial in the size of G and r. However, Mendelzon and Wood [26] showed that the evaluation problem for regular expressions over graphs under the simple path semantics becomes intractable (NP-complete to be precise).

For a language L, we define $q_L^{\text{simple}}(G)$ as the set of pairs (u, v) for which there is a sequence of nodes $u = v_1, \ldots, v_n = v$ in V_G and a sequence of labels $\sigma_1, \ldots, \sigma_n$ such that $v_i \neq v_j$ for $i \neq j$ and $v_i \xrightarrow{\sigma_i}_G v_{i+1}$ for all $i < n$. By $\mathcal{L}_{\text{simple}}$ we denote $\mathcal{L}$ under the simple path semantics. In analogy to Proposition 3.1, DEF($\mathcal{L}_{\text{simple}}$) is equivalent to DEF($\mathcal{F}_{\text{simple}}$) for every class of languages $\mathcal{L}$ containing $\mathcal{F}$.

THEOREM 3.6. *DEF($\mathcal{F}_{simple}$) is* NP*-complete.*

PROOF. We start by membership in NP. Let $G = (V, E)$ be a graph with n nodes and let $S \subseteq V \times V$. Then, as every simple path can contain at most n nodes, every pair $(u, v) \in S$ is witnessed by a word of length at most n. Therefore, non-deterministically guess for every such pair a word $w_{u,v}$ of at most length n, and test whether $q_R(G) = S$ for $R = \{w_{u,v} \mid (u, v) \in S\}$.

For the NP-hardness part of the argument, we reduce validity of 3DNF to the complement of DEF($\mathcal{F}_{\text{simple}}$), where 3DNF denotes the boolean propositional formulas in disjunctive normal form where each disjunct contains precisely three literals. The validity problem for 3DNF is NP-complete. Therefore, let φ be a formula over the variables $X = \{x_1, \ldots, x_k\}$ which is of the form $\bigvee_{i=1}^{n} \alpha_i$ where each $\alpha_i = \alpha_{i1} \wedge \alpha_{i2} \wedge \alpha_{i3}$ is a conjunction of literals. Let $\Sigma = \{0, 1, s, f\}$. When interpreting s as the initial node and f as the final node, the graph G_X, illustrated for $X = \{x_1, x_2, \ldots x_6\}$ in Figure 3, defines all words of the form $s\sigma_1 \cdots \sigma_k f$ where each $\sigma_i \in \{0, 1\}$ assigns a truth value to variable x_i. Then, similarly, G_{α_i}, illustrated for $X = \{x_1, x_2, \ldots x_6\}$ and $\alpha_i = \neg x_2 \wedge x_4 \wedge x_5$ in Figure 4, defines all words of the form swf where w encodes a truth assignment which makes α_i true.

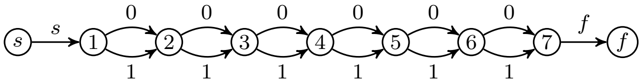


Figure 3: Graph G_X for $X = \{x_1, x_2, \ldots x_6\}$.

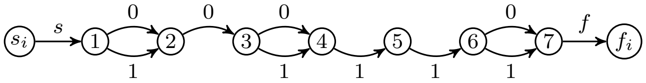


Figure 4: Graph G_{α_i} for $X = \{x_1, x_2, \ldots x_6\}$ and $\alpha_i = \neg x_2 \wedge x_4 \wedge x_5$.

Now, define G as the disjoint union of $G_{\alpha_1}, \ldots, G_{\alpha_n}$, and G_X, and set $S = \{(s, f)\}$. Then φ is valid iff (G, S) is not $\mathcal{F}$-definable. Indeed, when φ is valid, for every word w for which $s \xrightarrow{w}_G f$ there is a disjunct α_i for which $s_i \xrightarrow{w}_G f_i$. Conversely, when φ is not valid there is a truth assignment which makes φ false. Let w be the word encoding of that truth assignment. Then, $q_{\{w\}}(G) = S$. □

The class of *restricted regular expressions* was introduced by Mendelzon and Wood [26] in connection with the evaluation problem of regular path queries under the simple path semantics. In particular, they showed that the evaluation problem for restricted regular expressions under the simple path semantics is in PTIME. A regular expression is restricted if it is equivalent to the regular expression obtained from it by replacing all symbols a by $a?$. For example, 0^*10^* is not restricted but $0^*10^* + 0^*$ is restricted as it is equivalent to $(0?)^*1?(0?)^* + (0?)^*$. Restricted regular expressions define the abbreviated regular languages. These are the regular languages which are closed under the subword operation. A word v is a *subword* of w, when v is obtained from w by deleting some of its, not necessarily consecutive, letters. Now, a regular language L is *abbreviated* if for any word in L all its subwords also belong to L. We denote the class of abbreviated regular languages by $\mathcal{A}$ and we show (proof omitted) that the definability problem remains NP-complete even though the evaluation problem under the simple path semantics becomes tractable.

THEOREM 3.7. DEF($\mathcal{A}$) *is* NP*-complete.*

4. CONJUNCTIVE REGULAR PATH QUERIES

In this section, we consider more expressive query languages. In particular, we investigate the definability problem for various classes of conjunctive regular path queries (CRPQs) [9, 17, 14] such as chain (CCRPQs), linear (LCRPQs) and acyclic CRPQs (ACRPQs) or the class formed by queries that are unions of CRPQs (UCRPQs). Interestingly, we show that while the problem is PSPACE-hard for LCRPQs, it drops to NP for the most expressive class, the UCRPQs.

Let $\overline{x} = (x_1, \dots x_n)$ and $\overline{y} = (y_1, \dots y_m)$ be tuples of variables. Let $X = \{x_1, \dots x_n\}$ and $Y = \{y_1, \dots y_m\}$. A *Conjunctive Regular Path Query* (CRPQs) is a formula $\varphi(\overline{x})$ of the form:

$$\exists\ \overline{y}\ (r_1(z_1, z_1') \wedge \dots \wedge r_k(z_k, z_k'))$$

where each r_i is a regular expression and $\{z_1, \dots z_k, z_1', \dots, z_k'\}$ is equal to $X \cup Y$. We abuse notation and denote the class of CRPQs by CRPQ as well and similarly for the subclasses we consider below.

Let $G = (V, E)$ be a graph, we say that a tuple of nodes $(v_1, \dots, v_n) \in V^n$ *satisfies* the query $\varphi(\overline{x})$ if there is a mapping h from $X \cup Y$ to V with $h(x_i) = v_i$ and such that $(h(z_i), h(z_i')) \in q_{\mathcal{L}(r_i)}(G)$ for all $i \leq k$. We call such a mapping a *valid assignment*. We define the evaluation $q_\varphi(G)$ of $\varphi(\overline{x})$ on G as the set of tuples $(v_1, \dots, v_n) \in V^n$ that satisfy φ.

We restrict attention to unary and binary queries.

4.1 Chain CRPQs

We start by defining CRPQs of a restricted form: chain and linear CRPQs. We say that a CRPQ is a *chain* CRPQ (CCRPQ) if it is of the form

$$\varphi(x, y) = \exists \bar{x}\ (r_1(x, x_1) \wedge r_2(x_1, x_2) \wedge \dots \wedge r_n(x_{n-1}, y)).$$

The unary variant then is of the form

$$\varphi(x) = \exists \bar{x}\ (r_1(x, x_1) \wedge r_2(x_1, x_2) \wedge \dots \wedge r_n(x_{n-1}, x_n)).$$

We say that a CRPQ is *linear* (LCRPQ) if it is of the form $\varphi(x, y) =$

$$\exists \bar{x}_1, \dots, \bar{x}_n\ (\psi_0(x) \wedge r_1(x, x_1) \wedge \psi_1(x_1) \wedge r_2(x_1, x_2) \wedge \dots$$
$$\dots \wedge \psi_{n-1}(x_{n-1}) \wedge r_n(x_{n-1}, y) \wedge \psi_n(y))$$

where each ψ_i is a unary CCRPQ. So, an LCRPQ is a CCRPQ with tentacles. Note that as regular expressions can be ε, LCRPQ is a superset of CCRPQ.

As every CCRPQ $\varphi(x, y)$ of the form

$$\exists \bar{x}\ (r_1(x, x_1) \wedge r_2(x_1, x_2) \wedge \dots \wedge r_n(x_{n-1}, y)),$$

is equivalent to the path query (in the sense of the previous section) defined by the regular language expressed by the regular expression $r_1 \cdots r_n$, it readily follows from Corollary 3.4, that DEF(CCRPQ) is PSPACE-complete. Nevertheless, to prepare for the more elaborate construction in Theorem 4.3, we provide a more direct proof which shows that CCRPQs and LCRPQs defining a given relation can be constructed in PSPACE by guessing the associated regular expressions.

We recall some basic operations on binary relations. Let R_1 and R_2 be two binary relations. The composition of R_1 and R_2, denoted by $R_1 \circ R_2$, is defined as $\{(x, y) \mid \exists z R_1(x, z) \text{ and } R_2(z, y)\}$. We denote by $R_1 \ltimes R_2 = \{(x, y) \in R_1 \mid \exists z R_2(y, z)\}$ the semi-join of R_1 and R_2. Finally, by $\pi_1(R_1)$ we denote $\{x \mid \exists y (x, y) \in R_1\}$.

By $\mathcal{Q}^{\text{unary}}$, we denote $\mathcal{Q}$ restricted to unary queries only.

LEMMA 4.1. *1.* DEF(*CCRPQ*^{unary}) *is in* PSPACE.

2. DEF(*CCRPQ*) *is in* PSPACE.

3. DEF(*LCRPQ*) *is in* PSPACE.

PROOF. (1) Let $G = (V_G, E_G)$ be a graph and $S \subseteq V_G$. Note that (G, S) is unary CCRPQ-definable iff there is a formula $\varphi(x) = \exists y\, r(x, y) \wedge \varphi_y(y)$, where $\varphi_y(y)$ is a unary CCRPQ, with $q_\varphi(G) = S$.[2] The latter requirement reduces to the existence of a relation $R_{x,y} \subseteq V_G \times V_G$ and a set $S_y \subseteq V_G$ such that

- $(G, R_{x,y})$ is REG-definable;
- (G, S_y) is unary CCRPQ-definable; and,
- $S = \pi_1(R_{x,y} \circ S_y \times S_y)$.

On input (G, S), the following algorithm checks for the existence of such $R_{x,y}$ and S_y:

1. Guess a REG-definable binary relation $R_{x,y} \subseteq V_G \times V_G$ and a set of nodes $S_y \subseteq V_G$ such that $S = \pi_1(R_{x,y} \circ S_y \times S_y)$.
2. If $S_y = V_G$ accept. Otherwise, set $S := S_y$ and go to step (1).

The algorithm is in PSPACE. Indeed, the size of the guessed relations is at most quadratic in the input and testing for REG-definability is in PSPACE by Corollary 3.4.

It remains to argue that the algorithm accepts iff (G, S) is CCRPQ-definable. Assume (G, S) is CCRPQ-distinguishable. Then, there is a formula $\varphi(x) = \exists \bar{x}\ (r_1(x, x_1) \wedge r_2(x_1, x_2) \wedge \dots \wedge r_n(x_{n-1}, x_n))$ with $q_\varphi(G) = S$. The algorithm accepts by guessing at each stage $i < n$ the relation $q_{\mathcal{L}(r_i)}(G)$ and the set $q_{\varphi_i}(G)$ where

$$\varphi_i = \exists x_{i+1}, \dots, x_n\ (r_{i+1}(x_i, x_{i+1}) \wedge \dots \wedge r_n(x_{n-1}, x_n)].$$

At stage n, the algorithm guesses the relation $q_{\mathcal{L}(r_n)}(G)$ and the set V_G. Conversely, assume the algorithm accepts after

[2] Note that φ is syntactically not a CCRPQ but is equivalent to one.

n iterations and let $(R_i)_{i \leq n}$ be the REG-definable relations definable by the regular expressions $(r_i)_{i \leq n}$. Then, $q_\varphi(G) = S$ for $\varphi(x) = \exists \bar{x}\ (r_1(x, x_1) \wedge r_2(x_1, x_2) \wedge \ldots \wedge r_n(x_{n-1}, x_n))$.

(2) Let $G = (V_G, E_G)$ be a graph and $S \subseteq V_G \times V_G$. Note that (G, S) is binary CCRPQ-definable iff there is a formula $\varphi(x, y) = \exists z\, (r(x, z) \wedge \varphi_{z,y}(z, y))$, where $\varphi_{z,y}(z, y)$ is a binary CCRPQ, with $q_\varphi(G) = S$. The latter requirement reduces to the existence of a relation $R_{x,z} \subseteq V_G \times V_G$ and a relation $R_{z,y} \subseteq V_G \times V_G$ such that

- $(G, R_{x,z})$ is REG-definable;
- $(G, R_{z,y})$ is binary CCRPQ-definable; and,
- $S = R_{x,z} \circ R_{z,y}$.

The following algorithm checks for the existence of such relations $R_{x,z}$ and $R_{z,y}$:

1. Guess a REG-definable binary relation $R_{x,z} \subseteq V_G \times V_G$ and a binary relation $R_{z,y} \subseteq V_G \times V_G$ such that $S = R_{x,z} \circ R_{z,y}$.
2. If $R_{z,y} = V_G \times V_G$ accept. Otherwise, set $S := R_{z,y}$ and go to step (1).

Correctness and membership in PSPACE of the above algorithm is similar to the corresponding argument for unary CCRPQs and is therefore omitted.

(3) Let $G = (V_G, E_G)$ be a graph and $S \subseteq V_G \times V_G$. Note that (G, S) is LCRPQ-definable iff there is a formula $\varphi(x, y) = \exists z\, r(x, z) \wedge \varphi_z(z) \wedge \varphi_{z,y}(z, y)$, where $\varphi_z(z)$ is a unary CCRPQ and $\varphi_{z,y}(z, y)$ is a binary CCRPQ, with $q_\varphi(G) = S$. The latter requirement reduces to the existence of a relation $R_{x,z} \subseteq V_G \times V_G$, a set $S_z \subseteq V_G$, and a relation $R_{z,y} \subseteq V_G \times V_G$ such that

- $(G, R_{x,z})$ is REG-definable;
- (G, S_z) is unary CCRPQ-definable;
- $(G, R_{z,y})$ is binary CCRPQ-definable; and,
- $S = R_{x,z} \circ (S_z \times S_z) \circ R_{z,y}$.

Consider the following algorithm:

1. Guess a REG-definable relation $R_{x,z}$, a unary CRPQ-definable set S_z, and a relation $R_{z,y}$ such that $S = R_{x,z} \circ (S_z \times S_z) \circ R_{z,y}$.
2. If $R_{z,y} = V_G \times V_G$ accept. Otherwise, set $S := R_{z,y}$ and go to step (1).

Correctness and membership in PSPACE of the above algorithm is similar to the corresponding argument for unary CCRPQs and is therefore omitted. □

The following Theorem follows from the proof of Theorem 4.10.

THEOREM 4.2. DEF(*LCRPQ*) *is* PSPACE-*complete.*

4.2 Acyclic CRPQs

We first recall the definition of acyclic CRPQs as, for instance, introduced in [5]. A query $\varphi(\overline{x}) = \exists\ \overline{y}\ (r_1(z_1, z_1') \wedge \cdots \wedge r_k(z_k, z_k'))$ naturally defines a graph $G_\varphi = (V_\varphi, E_\varphi)$ whose nodes V_φ are the variables in φ and whose edges are labelled with regular expressions as follows: $(z, r, z') \in E_\varphi$ if and only if there is some $i \leq k$ with $z = z_i$, $z' = z_i'$ and $r = r_i$. A CRPQ is *acyclic* (ACRPQ) if its graph (considered undirected) is acyclic.

THEOREM 4.3. *1.* DEF($ACRPQ^{unary}$) *is in* EXPTIME.

2. DEF(*ACRPQ*) *is in* EXPTIME.

PROOF. The proof follows closely that of Lemma 4.1. Whereas in the latter proof the recursion is linear, the structure of ACPRQs requires to use non-linear recursion. To this end, we employ alternation exploiting that EXPTIME equals alternating PSPACE.

(1) Let $G = (V_G, E_G)$ be a graph and $S \subseteq V_G$. Note that (G, S) is unary ACRPQ-definable iff there is a formula $\varphi(x) = \exists \bar{y} \bigwedge_{i=1}^m r_i(x, y_i) \wedge \varphi_i(y_i)$, where $\varphi_i(y)$ is a unary ACRPQ with $q_\varphi(G) = S$.

The latter reduces to the existence of relations $R_{x,y_i} \subseteq V_G \times V_G$ and sets $S_{y_i} \subseteq V_G$ such that

- each (G, R_{x,y_i}) is REG-definable;
- each (G, S_{y_i}) is unary tree definable; and,
- $S = \bigcap_{i=1}^m (R_{x,y_i} \circ S_{y_i} \times S_{y_i})$.

Consider the following alternating PSPACE algorithm:

```
1. Set $S_{\text{temp}} = V_G \times V_G$.
2. Guess a REG-definable binary relation $R_{x,y}$ and a set of nodes $S_y$.
3. Create two branches and accept if both accept:
    (a) Set $S_{\text{temp}} := S_{\text{temp}} \cap (R_{x,y} \circ S_y \times S_y)$. If $S = S_{\text{temp}}$ accept. Otherwise go to step (2).
    (b) If $S_y = V_G$ then accept, else set $S := S_y$ and go to (1).
```

(2) Let G be a graph and $S \subseteq V_G \times V_G$. Note that (G, S) is binary ACRPQ-definable iff there is a formula

$$\varphi(x, y) = \varphi_i(x) \wedge \exists \bar{u} \bigwedge_{i=1}^{n} s_i(x, u_i) \wedge \psi_i(u_i, y),$$

where $\varphi_i(x)$ is a unary ACRPQ and $\psi_i(u_i, y)$ is a binary ACRPQ, with $q_\varphi(G) = S$.

The latter reduces to the existence of a set $S_x \subseteq V_G$ and relations $R_{x,u_i}, R_{u_i,y} \subseteq V_G \times V_G$ such that

- (G, S_x) is unary ACRPQ-definable;
- each (G, R_{x,u_i}) is REG-definable;
- each $(G, R_{u_i,y})$ is binary ACRPQ-definable; and,
- $S = (S_x \times V_G) \cap \bigcap_{i=1}^n R_{x,u_i} \circ R_{u_i,y}$.

The alternating PSPACE algorithm is now similar to (1). □

4.2.1 Hardness

We show that DEF(ACRPQ) is PSPACE-hard. It will follow from the proof that DEF(LCRPQ) is PSPACE-hard, as well. We reduce from the emptiness of intersection problem for DFAs as defined in Section 2. The input consists of a sequence of DFAs $A_1, \ldots, A_n$. We construct a graph G and a set S such that (G, S) is ACRPQ-definable iff $\bigcap_{i=1}^{n} \mathcal{L}(A_i)$ is non-empty. We start by summarizing the main steps of the proof. For each DFA A_i, we construct a graph G_i with a designated start node s_i and a designated end node f_i such that a word w is accepted by A_i if and only if there is a path in G_i labelled with w (or more precisely by an encoding $\gamma(w)$ of w) from s_i to f_i. We then define the graph G as the disjoint union of the G_i and a constraint graph G_c (to be discussed next) and define the set S to contain all (s_i, f_i) but no elements from the constraint set S_c. The constraint graph G_c together with the constraint set S_c ensures that if there is an ACRPQ $\varphi(x, y)$ with $S_c \cap q_\varphi(G) = \emptyset$, then $\varphi(x, y)$ is equivalent to a linear ACRPQ whose regular expressions consist of a single alphabet symbol. Therefore, $\varphi(x, y)$ defines a single word and as every $(s_i, f_i) \in q_\varphi(G)$, that word is accepted by every A_i. The main difficulty of the proof lies in the construction of G_c and S_c.

Let us first note that queries are monotone with regards to the subgraph relation. Specifically, we write $G_1 \subseteq G_2$ for two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ when $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$.

LEMMA 4.4. *Let G_1, G_2 be two graphs with $G_1 \subseteq G_2$. Then $q_\varphi(G_1) \subseteq q_\varphi(G_2)$, for any CRPQ φ.*

We use the above monotonicity property to construct G as the disjoint union of several graphs. Some properties are easier to prove on the subgraphs, they then carry over, by monotonicity, to the whole graph.

The first part of this section is devoted to the constraint graph G_c. It is constructed as the disjoint union of the subgraphs $G_\rightarrow$, $G_\perp$, and $G_{\alpha\beta}$. For each of them we prove the relevant properties that also hold, by monotonicity, to G.

Denote by $G_\rightarrow = (V_\rightarrow, E_\rightarrow)$ the graph depicted in Fig. 5. The first lemma states that if the pair $(s_\rightarrow, f_\rightarrow)$ is *not* selected by a ACRPQ $\varphi(x, y)$, then there is a directed path in the graph of φ from x to y.

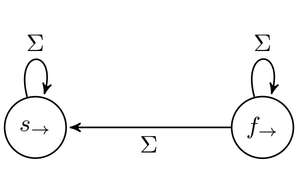


Figure 5: Component $G_\rightarrow$.

LEMMA 4.5. *Let $\varphi(x, y)$ be a ACRPQ. If $(s_\rightarrow, f_\rightarrow) \notin q_\varphi(G_\rightarrow)$ then φ is equivalent to a formula of the form $\exists \bar{z}$ $(\varphi_1(x, y) \wedge \psi(x, y))$ where $\varphi_1 = r_1(x, z_1) \wedge r_2(z_1, z_2) \wedge \cdots \wedge r_n(z_{n-1}, y)$.*

PROOF. We assume that φ does not contain conjuncts of the form $\varepsilon(z, z')$ as we can remove these by replacing every occurrence of the variable z' by z. Let G_φ be the graph associated with the query $\varphi(x, y)$. Let S_1 be the set of variables reachable from x in G_φ and let S_2 be the set comprising the remaining variables. Then, let h be a mapping that maps all variables in S_1 to $s_\rightarrow$ and all variables in S_2 to $f_\rightarrow$. Assume φ is not of the claimed form. But, then $y \in S_2$ as y is not reachable from x. Let us show that h is a valid assignment witnessing that $(s_\rightarrow, f_\rightarrow) \in q_\varphi(G_\rightarrow)$, which contradicts our assumption. Indeed, first we have $h(x) = s_\rightarrow$ and $h(y) = f_\rightarrow$. Second, for any disjunct $r(z_1, z_2)$ in φ we check that $(h(z), h(z')) \in q_r(G)$. We consider three cases: (*i*) $h(z) = h(z') = s_\rightarrow$ or $h(z) = h(z') = f_\rightarrow$, then $(h(z), h(z')) \in q_r(G)$ for any r, (*ii*) $h(z) = s_\rightarrow$, $h(z') = f_\rightarrow$, this is impossible by definition of h. (*iii*) $h(z) = f_\rightarrow$, $h(z') = s_\rightarrow$, then $(h(z), h(z')) \in q_r(G)$ for any r (as there is a Σ labelled edge from the first node to the second and there are Σ self-loops on both nodes). □

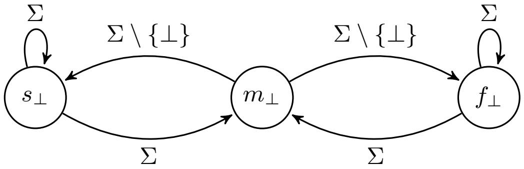


Figure 6: Component $G_\perp$.

Denote by $G_\perp = (V_\perp, E_\perp)$ the graph depicted in Fig. 6. The following result states that if an acyclic query $\varphi(x, y)$, whose graph contains a directed path from x to y, does not select $(s_\perp, f_\perp)$ in $G_\perp$, then each regular expression, appearing on an edge of that path, is a disjunction of words of length at most 1.

LEMMA 4.6. *Let $\perp \in \Sigma$ and $\varphi(x, y)$ be an ACRPQ query of the form $\exists \bar{z}(\varphi_1(x, y) \wedge \psi(x, y))$ where $\varphi_1 = r_1(x, z_1) \wedge r_2(z_1, z_2) \wedge \cdots \wedge r_n(z_{n-1}, y)$ and no r_i contains the symbol $\perp$.*

If $(s_\perp, f_\perp) \notin q_\varphi(G_\perp)$, then, for each $i \leq n$, $r_i = \alpha_{i,1} + \cdots + \alpha_{i,k_i}$, with $\alpha_{i,j} \in \Sigma \cup \{\varepsilon\}$.

PROOF. Let $\varphi(x, y)$ be as stated in the lemma and assume that $(s_\perp, f_\perp) \notin q_\varphi(G_\perp)$. Towards a contradiction suppose that for some r_i, there is a word w of length larger than 1 with $w \in \mathcal{L}(r_i)$. We define the following valid assignment h which witnesses that $(s_\perp, f_\perp) \in q_\varphi(G_\perp)$. Let G_φ be the graph associated with the query φ, and let G_φ^- be the graph obtained by G_φ by removing the edge from z_{i-1} to z_i. Since φ is acyclic we have the following three disjoint sets over the variables $\{x, y\} \cup \bar{z}$. Let S_1 be all the variables on the same connected component of G_φ^- as x, let S_2 be all the variables on the same connected component of G_φ^- as y and let S_3 be the rest of the variables. Then define h so that all variables of S_1 and S_3 are mapped to $s_\perp$ and all variables of S_2 are mapped to $f_\perp$. Notice that, since no r_i contains the symbol $\perp$, any word of length larger than 1 is satisfied on the path from $s_\perp$ to $f_\perp$ via $m_\perp$. Therefore, $r_i(z_{i-1}, z_i)$ is satisfied. One can easily check that the other conjuncts of φ are also satisfied. Indeed, if $r(z, z')$ is any other conjunct of φ then, by construction of h, $h(z) = h(z') = s_\perp$ or $h(z) = h(z') = f_\perp$, and $(s_\perp, s_\perp), (f_\perp, f_\perp) \in q_r(G_\perp)$ for any regular expression r. Therefore, h witnesses $(s_\perp, f_\perp) \in q_\varphi(G_\perp)$ which leads to the desired contradiction. □

Let $\alpha, \beta \in \Sigma$, then $G_{\alpha\beta} = (V_{\alpha\beta}, E_{\alpha\beta})$ is the graph described in Fig. 7. The following lemma states that, if an

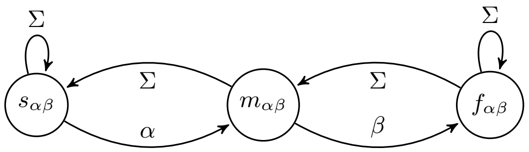


Figure 7: Component $G_{\alpha\beta}$ for $\alpha, \beta \in \Sigma$.

acyclic query $\varphi(x, y)$ whose graph contains a directed path from x to y does not select $(s_{\alpha\beta}, f_{\alpha\beta})$ in $G_{\alpha\beta}$, then the word $\alpha\beta$ does not appear on that path. This property is later used to prevent disjunctions in the regular expressions. The proof is similar to the proof of Lemma 4.6.

LEMMA 4.7. *Let $\varphi(x, y)$ be a ACRPQ of the form $\exists\overline{z}$ $(\varphi_1(x, y) \wedge \psi(x, y))$ where $\varphi_1 = r_1(x, z_1) \wedge r_2(z_1, z_2) \wedge \cdots \wedge r_n(z_{n-1}, y)$.*
Then, $(s_{\alpha\beta}, f_{\alpha\beta}) \in q_\varphi(G_{\alpha\beta})$, if and only if there exist $\alpha', \beta' \in \Sigma^*$, such that $\alpha'\alpha\beta\beta' \in \mathcal{L}(r_1 \cdot r_2 \cdots r_n)$.

Consider a formula φ of the form depicted in Fig. 8. That is, (*i*) φ has a directed path from x to y, (*ii*) each node on the unique path from x to y in φ has an outgoing edge labelled $\perp$, and (*iii*) moreover, each edge on that path is labelled with a regular expression formed by a single letter.

We show in the next lemma that the pair $(s_\perp, f_\perp)$ and $(s_\rightarrow, f_\rightarrow)$, are not selected by φ in any graph G that contains as disjoint subgraphs the graphs $G_\perp$ and $G_\rightarrow$. Moreover $(s_{\alpha\beta}, f_{\alpha\beta})$ is not selected in any graph that contains $G_{\alpha\beta}$ as a disjoint subgraph, whenever $\alpha\beta$ is not a factor of $\sigma_1 \ldots \sigma_n$.

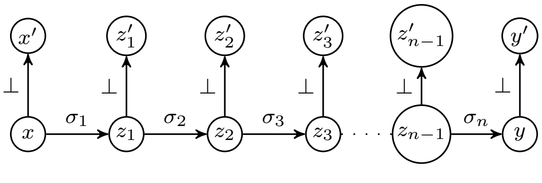


Figure 8: The graph of φ such that $(s_\perp, f_\perp) \notin q_\varphi(G_\perp)$.

LEMMA 4.8. *Let $\perp \in \Sigma$ and $\varphi(x, y)$ be an ACRPQ of the form (depicted in Fig. 8):*

$$\begin{aligned}\varphi(x,y) \quad = \quad & \exists\overline{z}(\sigma_1(x, z_1) \wedge \perp(x, x') \wedge \\ & \bigwedge_{i=1}^{n-2} (\sigma_{i+1}(z_i, z_{i+1}) \wedge \perp(z_i, z'_i)) \wedge \\ & \sigma_n(z_{n-1}, y) \wedge \perp(z_{n-1}, z'_{n-1}) \wedge \perp(y, y')),\end{aligned}$$

where, for each $i \leq n$, $\sigma_i \in \Sigma$.
Then

- *$(s_\perp, f_\perp) \notin q_\varphi(G)$ for any G with $G = G_\perp \cup G'$ where $V_\perp \cap V' = \emptyset$ (we assume $G' = (V', E')$);*
- *$(s_{\alpha\beta}, f_{\alpha\beta}) \notin q_\varphi(G)$ for any G with $G = G_{\alpha\beta} \cup G'$, $V_{\alpha\beta} \cap V' = \emptyset$, and $\alpha\beta$ is not a factor of $\sigma_1 \ldots \sigma_n$; and,*
- *$(s_\rightarrow, f_\rightarrow) \notin q_\varphi(G)$ for any G with $G = G_\rightarrow \cup G'$ and $V_\rightarrow \cap V' = \emptyset$.*

PROOF. We prove the first item, the others are proved similarly. Suppose for contradiction that $(s_\perp, f_\perp) \in q_\varphi(G)$. Let h be the valid assignment witnessing this. Since $h(x) = s_\perp$ and $h(y) = f_\perp$, and since for each $i \leq n$, $\sigma_i \in \Sigma$, it cannot be the case that for some $j < n$, $h(z_j) = s_\perp$ and $h(z_{j+1}) = f_\perp$. Therefore, for some $j < n$, it is the case that $h(z_j) = m_\perp$ and $h(z_{j+1}) = f_\perp$. But φ expresses that each of the variables z_j has an outgoing edge labelled with $\perp$. But this is a contradiction since $m_\perp$ in $G_\perp$ has no such outgoing edge, and therefore $m_\perp$ in $G = G_\perp \cup G'$ has no outgoing $\perp$ edge either because $G_\perp$ and G' are disjoint. □

The main proof relies on a transformation that maps a DFA A on the binary alphabet $\Delta = \{a, b\}$ to a graph G whose edges are labelled with elements of $\Sigma = \{a_1, a_2, b_1, b_2, s, f, \perp\}$, and with two distinguished nodes s and f. This transformation is such that a word w is accepted by A if and only if there is a path from s to f in G labelled with $s\mu(w)f$ where μ is the morphism that maps a on a_1a_2 and b on b_1b_2. All words labelling a path in G corresponding to accepted words in A belong to $L = \mathcal{L}(s((a_1a_2) \vee (b_1b_2))^*f)$. The purpose of the next lemma is to show that we can enforce some queries to only define words in L.

Every factor of length 2 of a word w in L (two consecutive letters of w) belongs to the following set:

$$\begin{aligned}\mathit{factors} = \quad & \{sa_1, sb_1, sf, a_1a_2, a_2a_1, a_2b_1, a_2f, \\ & b_1b_2, b_2a_1, b_2b_1, b_2f\}.\end{aligned}$$

We define G_{factors} to be the graph $\bigcup_{\alpha\beta \notin \mathit{factors}} G_{\alpha\beta}$, where we assume that $V_{\alpha\beta} \cap V_{\alpha'\beta'} = \emptyset$ whenever $\alpha\beta \neq \alpha'\beta'$. We will show that, under some hypothesis, a query that does not select $(s_{\alpha\beta}, f_{\alpha\beta})$ (for $\alpha\beta \notin \mathit{factors}$) in G_{factors} is equivalent to a query where every regular expression is a single alphabet symbol.

LEMMA 4.9. *Let $\varphi(x, y)$ be a ACRPQ of the form $\exists\overline{z}$ $(\varphi_1(x, y) \wedge \psi(x, y))$ where $\varphi_1 = r_1(x, z_1) \wedge r_2(z_1, z_2) \wedge \cdots \wedge r_n(z_{n-1}, y)$, $r_i = \alpha_{i,1} + \cdots + \alpha_{i,k_i}$, with $\alpha_{i,j} \in \Sigma \cup \{\varepsilon\} \setminus \{\perp\}$, and $r_1 = s$, $r_n = f$. Let*

$$\alpha_{1,1} \cdot \alpha_{2,1} \cdots \alpha_{n,1} \in \mathcal{L}(s((a_1a_2) \vee (b_1b_2))^*f). \qquad (1)$$

If $(s_{\alpha\beta}, f_{\alpha\beta}) \notin q_\varphi(G_{\mathit{factors}})$ for all $\alpha\beta \notin \mathit{factors}$, then $r_i \equiv \alpha_i$ and $\alpha_i \in \Sigma \cup \{\varepsilon\} \setminus \{\perp\}$ for all $i \leq n$.

PROOF. We show that each r_i is equivalent to $\alpha_{i,1}$. The argument is based on the following observations. Because $(s_{\alpha\beta}, f_{\alpha\beta}) \notin q_\varphi(G_{\alpha\beta})$ for all $\alpha\beta \notin \mathit{factors}$, and equation (1) holds, it follows that (*i*) s (resp. f) is the first (resp. last) symbol; (*ii*) a_1 (resp. b_1) can only be followed by a_2 (resp. b_2); (*iii*) a_2 (resp. b_2) can only be preceded by a_1 (resp. b_1); and, (*iv*) s, a_2 and b_2 can only be followed by f, a_1 or b_1.

We consider several cases:

- If $\alpha_{i,1} = a_1$ then, by observation (*ii*) above, $\alpha_{i+1,1} = a_2$. But then, by observation (*iii*), all other disjuncts $\alpha_{i,j} = a_1$ (they can not be ε either, as shown in the last item below).
- If $\alpha_{i,1} = b_1$ then, similarly as in the case for a_1, each $\alpha_{i,j}$ can only be b_1.
- If $\alpha_{i,1} = a_2$ then, by (*iii*), $\alpha_{i-1,1} = a_1$. But then, by (*ii*), all other disjuncts $\alpha_{i,j}$ can only be a_2.
- If $\alpha_{i,1} = b_2$ then, similarly as in the case for a_2, all other disjuncts $\alpha_{i,j}$ can only be b_2.

- If $\alpha_{i,1} = s$ (resp. $\alpha_{i,1} = f$) then it can only be the first (resp. last) symbol (observation (i)), i.e. $i = 1$ (resp. $i = n$). And by equation (1) r_1 is equivalent to s (resp., r_n is equivalent to f).

- If $\alpha_{i,1} = \varepsilon$ and $\alpha_{i,2} \neq \varepsilon$, then $\alpha_{1,1} \dots \alpha_{i,1} \dots \alpha_{n,1}$ or $\alpha_{1,1} \dots \alpha_{i-1,1}\alpha_{i,2}\alpha_{i+1,1} \dots \alpha_{n,1}$ is of the form $w_1\alpha\beta w_2$ with $\alpha\beta \notin$ *factors*, which implies that $(s_{\alpha\beta}, f_{\alpha\beta}) \in q_\varphi(G_{factors})$.

This concludes the proof. □

We are now ready for the main result of this section:

THEOREM 4.10. DEF(*ACRPQ*) *is* PSPACE-*hard.*

PROOF. We reduce the emptiness problem for the intersection of n DFAs to DEF(ACRPQ). Let $A_1, \dots A_n$ be n DFAs over the alphabet $\Delta = \{a, b\}$. For each $i \leq n$, we define the associated graph $G_i = (V_i, E_i)$, on the alphabet $\Sigma = \{s, f, a_1, a_2, b_1, b_2, \bot\}$, obtained from the graph representation of A_i by

- adding an initial node s_i connected to the initial node by an edge labelled s;

- adding a final node f_i and connecting each final state of A_i to f_i by an edge labelled f;

- for each edge $e = (v_1, a, v_2)$ (resp. $e = (v_1, b, v_2)$), adding a new node v_1' and replacing e with two new edges (v_1, a_1, v_1') and (v_1', a_2, v_2) (resp (v_1, b_1, v_1') and (v_1', b_2, v_2)); and,

- adding a new node, say $v_\bot$, and for each node v, adding a new edge $(v, \bot, v_\bot)$.

Consider the morphism μ from Δ^* to Σ^* defined by $\mu(a) = a_1a_2$ and $\mu(b) = b_1b_2$ and let γ be the function from Δ^* to Σ^* defined as $\gamma(w) = s\mu(w)f$. Clearly, A_i accepts the word w if and only if there is a path from s_i to f_i in G_i labelled with $\gamma(w)$.

Let $G = \bigcup_{i \leq n} G_i \uplus G_c$, where $G_c = G_\to \uplus G_\bot \uplus G_{factors}$. Denote the set of vertices and edges of G by V and E, respectively. Furthermore, let $\mathcal{S}$ be the set containing all sets S for which

1. $(s_\bot, f_\bot) \notin S$;
2. $(s_{\alpha\beta}, f_{\alpha\beta}) \notin S$ for any $\alpha\beta \notin$ *factors*;
3. $(s_\to, f_\to) \notin S$;
4. $(s_1, f_1), \dots (s_n, f_n) \in S$; and,
5. $(V_i \times V_i) \setminus \{(s_i, f_i)\} \cap S = \emptyset$, all $i \leq n$.

Notice that by the last condition, S does not contain any state of a DFA A_i apart from s_i and f_i.

It remains to show that there is a set $S \in \mathcal{S}$ for which (G, S) is ACRPQ-definable if and only if $\bigcap_{i \leq n} \mathcal{L}(A_i)$ is not empty.

First assume there exists a word $w \in \bigcap_{i \leq n} \mathcal{L}(A_i)$ with $\mu(w) = \sigma_1 \dots \sigma_m$. Then one can easily check that the query:

$$\begin{aligned}\varphi(x, y) \quad = \quad & \exists \bar{z}(s(x, z_1) \wedge \bot(x, x') \wedge \\ & \bigwedge_{i=1}^{m-1} (\sigma_i(z_i, z_{i+1}) \wedge \bot(z_i, z_i')) \wedge \\ & f(z_m, y) \wedge \bot(z_m, z_m') \wedge \bot(y, y'))\end{aligned}$$

satisfies the properties (1) to (5) of sets in $\mathcal{S}$. Indeed, the first three properties follow directly from Lemma 4.8. The fourth property follows as w belongs to the intersection of the A_i's, by construction of G_i. Finally, the last property follows as only the path from (s_i, f_i) in G_i is labelled with a word of the form swf.

For the *only if* direction, assume there exists an acyclic query $\varphi(x, y)$ with $q_\varphi(G) \in \mathcal{S}$. We proceed in several steps to show that there exists a formula of the form

$$\exists \bar{z}\ (s(x, z_1) \wedge \sigma_2(z_1, z_2) \wedge \dots \wedge f(z_m, y) \wedge \psi(x, y))$$

with $\sigma_i \in \Sigma$, such that $q_\varphi(G) \in \mathcal{S}$.

By Lemma 4.5 and Lemma 4.4, the query is of the form $\exists \bar{z}(\varphi_1(x, y) \wedge \psi(x, y))$ where $\varphi_1 = r_1(x, z_1) \wedge r_2(z_1, z_2) \wedge \dots \wedge r_m(z_m, y)$.

The following observation (†) is key to the sequel of the proof. Consider the language L_{valid} defined by the regular expression $s((a_1a_2) \vee (b_1b_2))^*f$. Then, let φ' be a formula obtained from φ by replacing every r_i by a r_i' such that

$$\mathcal{L}(r_1 \cdot r_2 \dots r_m) \cap L_{valid} \subseteq \mathcal{L}(r_1' \cdot r_2' \dots r_m')$$

and

$$\mathcal{L}(r_1' \cdot r_2' \dots r_m') \subseteq \mathcal{L}(r_1 \cdot r_2 \dots r_m).$$

Then $q_{\varphi'}(G) \in \mathcal{S}$. Indeed, because $q_{\varphi'}(G) \subseteq q_\varphi(G)$ properties (1),(2), (3) and (5) hold. And (4) holds because all paths from (s_i, f_i) are labelled with words in L_{valid}, and no such words have been removed. Therefore any valid assignment witnessing $(s_i, f_i) \in q_\varphi(G)$ is also a valid assignment witnessing $(s_i, f_i) \in q_{\varphi'}(G)$.

Next, we show that we can assume that no r_i in φ contains $\bot$. To this end, define $L_i' = \{w \in \mathcal{L}(r_i) \mid w \in (\Sigma \setminus \{\bot\})^*\}$ as the restriction of r_i to $\Sigma\Sigma \setminus \{\bot\}$. As L_i' is regular, there exists an equivalent regular expression r_i'. Moreover, it is easy to see that

$$\mathcal{L}(r_1 \cdot r_2 \dots r_m) \cap L_{valid} \subseteq \mathcal{L}(r_1' \cdot r_2' \dots r_m') \subseteq \mathcal{L}(r_1 \cdot r_2 \dots r_m)$$

Therefore by (†), $q_{\varphi'}(G) \in \mathcal{S}$, where φ' is obtained by replacing each r_i by r_i'. Therefore, we can assume that r_i does not contain $\bot$.

By Lemma 4.6 and Lemma 4.4, it follows that each r_i in φ is of the form $r_i = \alpha_{i,1} + \dots + \alpha_{i,k_i}$, with $\alpha_{i,j} \in \Sigma \cup \{\varepsilon\}$.

Next, utilizing observation (†), we show that we can assume that $r_1 = s$ in φ. We consider three cases (recall that $r_1 = \alpha_{1,1} + \dots + \alpha_{1,k_1}$):

- $\alpha_{1,j} \neq s$ for all $j \leq k_1$: Then $\alpha_{i,j} = \varepsilon$ for some $j \leq k_1$, because otherwise $\mathcal{L}(r_1 \cdot r_2 \dots r_m) \cap L_{valid} = \emptyset$ and $(s_i, f_i) \notin q_\varphi(G)$, which contradicts the hypothesis. By taking $r_1' = \varepsilon$ and using (†), we have $q_{\varphi'}(G) \in \mathcal{S}$. Then φ'' obtained from φ' by removing r_1 and renaming z_1 with x is equivalent to φ'. And we can reiterate the procedure on φ''.

- $\alpha_{1,1} = s$ and $\alpha_{1,j} \neq \varepsilon$ for all $j > 1$: clearly if one takes $r_1' = s$, then, by (†), $q_{\varphi'}(G) \in \mathcal{S}$ because $\mathcal{L}(r_1 \dots r_m) \cap L_{valid} \subseteq \mathcal{L}(r_1'r_2 \dots r_m) \subseteq \mathcal{L}(r_1 \dots r_m)$.

- $\alpha_{1,1} = s$ and $\alpha_{1,j} = \varepsilon$ for some $j > 1$: if $sw' \in \mathcal{L}(r_2 \dots r_m)$ for some word w then $ssw \in \mathcal{L}(r_1r_2 \dots r_m)$, but this is not possible because ss can not be a factor of words in $\mathcal{L}(r_1r_2 \dots r_m)$ (by Lemma 4.7, as $G_{ss} \in G$). Otherwise, if $sw \notin \mathcal{L}(r_2 \dots r_m)$ for any $w \in \Sigma^*$, then take $r_1' = s$, we have $\mathcal{L}(r_1 \dots r_m) \cap L_{valid} \subseteq$

$\mathcal{L}(r_1'r_2 \dots r_m) \subseteq \mathcal{L}(r_1 \dots r_m)$, and, using †, $q_{\varphi'}(G) \in \mathcal{S}$.

Similarly, we can suppose that $r_m = f$.

We can now apply Lemma 4.9 which shows that $r_i = \alpha_i$ with $\alpha_i \in \Sigma \cup \{\varepsilon\} \setminus \{\bot\}$. Therefore, there exists a formula of the form

$$\exists \overline{z}\ (s(x, z_1) \wedge \sigma_2(z_1, z_2) \wedge \dots \wedge f(z_m, y) \wedge \psi(x, y))$$

with $\sigma_i \in \Sigma$, such that $q_\varphi(G) \in \mathcal{S}$.

Now, as φ is acyclic there is only one path from x to y. This path is labelled by a single word $w = \alpha_1 \dots \alpha_m$. Therefore, for each $i \leq m$, there is a path from s_i to f_i labelled with w, and hence A_i accepts $w' = \gamma^{-1}(w)$, i.e. $w' \in \bigcap_{i \leq n} \mathcal{L}(A_i)$.

To conclude the proof, note that the size of the set $\mathcal{S}$ is fixed (it does not depend on the A_i nor on n). Therefore, the reduction goes as follows. Given n DFA we build the graph G (whose size is polynomial in the automata). Then, for each set $S \in \mathcal{S}$ we check whether S is ACRPQ-definable. If one of them is, then the intersection is not empty, otherwise it is. The latter is known as a truth table reduction.

Finally note that this reduction is also valid for the class of LCRPQs. □

4.3 General Case

In this section, we shed some light on the complexity of DEF(CRPQ). We present an EXPSPACE algorithm for deciding definability for the general class of binary CRPQs. We show that if there exists a query defining the given relation, then there exists one of size at most exponential in the size of the graph. Essentially, the algorithm guesses the query and then verifies that it indeed defines the given set. For proving the bound on the size of the query, we show that (i) the size of the regular expressions appearing in the query, (ii) the number of variables, and (iii) the number of occurrences of each variable, are all bounded by an exponential in the size of the graph.

We say that a CRPQ is in *word normal form* if every regular expression occurring in it is a disjunction of words.

LEMMA 4.11. *Let $G = (V, E)$ be a graph, $S \subseteq V \times V$, and let φ be a CRPQ. There exists a CRPQ ψ with $q_\varphi(G) = q_\psi(G)$ such that ψ is in word normal form and the size of ψ is at most exponential in the size of G.*

PROOF. We construct ψ by replacing every regular expression in φ by a disjunction of at most $|S|$ words of length exponential in the size of G. Let $r(x, y)$ be a conjunct in φ. Denote $q_{\mathcal{L}(r)}(G)$ by S_r. Then (G, S_r) is REG-definable and, by Proposition 3.1, (G, S_r) is $\mathcal{F}$-definable. By Lemma 3.2, it follows that for every $(u, v) \in S_r$,

$$L_G(u, v) \not\subseteq \bigcup_{(x,y) \notin S_r} L_G(x, y).$$

So, for each $(u, v) \in S_r$, there is a word $w_{(u,v)}$ with

$$w_{(u,v)} \in L_G(u, v) \setminus \Big(\bigcup_{(x,y) \notin S_r} L_G(x, y) \Big).$$

Then $q_R(G) = S_r$, for $R = \{w_{(u,v)} \mid (u, v) \in S_r\}$. As $\bigcup_{(x,y) \notin S_r} L_G(x, y)$ can be represented by an NFA of size linear in the sum of the sizes of the languages $L_G(x, y)$ (see Lemma 3.2), the witness of non-inclusion can be chosen with a length bounded by an exponential in the size of G. □

The next lemma shows that exponentially many variables are enough to define a given set if this set is CRPQ-definable.

LEMMA 4.12. *Let $G = (V, E)$ be a graph, $S \subseteq V \times V$, and φ a CRPQ with $q_\varphi(G) = S$. Then, there is a CRPQ ψ with $q_\psi(G) = S$ that has at most $|V|^{|S|}$ variables.*

PROOF. Assume φ is a CRPQ defining S with the smallest number of variables. Towards a contradiction, assume that number is larger than $|V|^\ell$, for $\ell = |S|$. Let $\{x, y\} \cup Y$ be the variables occurring in $\varphi(x, y)$. As $q_\varphi(G) = S$, for each pair $(s_1, s_2) \in S$ there is a valid assignment h_{s_1,s_2} from $\{x, y\} \cup Y$ to V that maps x on s_1 and y on s_2. Then, there are two variables $z_1, z_2 \in Y$, such that $h_{s_1,s_2}(z_1) = h_{s_1,s_2}(z_2)$ for all $(s_1, s_2) \in S$. Indeed, let $S = \{p_1, \dots, p_\ell\}$ and consider the mapping $v(z) = (h_{p_1}(z), \dots, h_{p_\ell}(z)) \in V^\ell$. Since there are $|V|^\ell$ elements in V^ℓ and more than $|V|^\ell$ variables in φ, there exist $z_1, z_2 \in Y$ with $v(z_1) = v(z_2)$. That is, $h_{s_1,s_2}(z_1) = h_{s_1,s_2}(z_2)$ for all $(s_1, s_2) \in S$. □

THEOREM 4.13. DEF(*CRPQ*) *is in* EXPSPACE.

PROOF. Given a graph $G = (V, E)$ and $S \subseteq V \times V$, we argue that if there exists a CRPQ φ with $q_\varphi(G) = S$, then there exists one of size at most exponential in the size of G. The algorithm then reduces to guessing such a CRPQ φ of at most exponential size and verifying that it defines S on G. The verification step is achieved by testing for every pair $(u, v) \in S$ that $(u, v) \in q_\varphi(G)$ and for every pair $(u, v) \notin S$ that $(u, v) \notin q_\varphi(G)$. The latter is done by cycling through all variable assignments (which are of size exponential in G) and using Theorem 2.2.

By Lemma 4.11, the regular expressions in any CRPQ are equivalent to one of size at most exponential in the size of G. By Lemma 4.12, the number of variables is at most exponential in the size of G. We next argue that the number of occurrences of each variable is also at most exponential in the size of G. We say that two conjuncts $r(z_1, z_2)$ and $r'(z_1, z_2)$ are G-equivalent whenever $q_{\mathcal{L}(r)}(G) = q_{\mathcal{L}(r')}(G)$. As $q_{\mathcal{L}(r)}(G)$ is a subset of $S \times S$, there are at most $2^{|S \times S|}$ non G-equivalent regular expressions. Therefore two variables, z_1 and z_2, occur in no more than $2^{|S \times S|}$ non-equivalent conjuncts of the form $r(z_1, z_2)$. □

4.4 Union of CRPQs

In this section, we consider unions of conjunctive regular path queries. We say that a pair of nodes (v_1, v_2) is CRPQ-distinguishable from a pair (v_1', v_2') in a graph G iff there is a CRPQ φ such that $(v_1, v_2) \in q_\varphi(G)$ and $(v_1', v_2') \notin q_\varphi(G)$. We first show that the complexity of checking whether a single pair of nodes is distinguishable in CRPQ from another pair of nodes, is in CONP. Then to decide whether a relation is definable using a union of CRPQs, amounts to checking if each of the pairs in the relation is definable by some query, and then constructing the union of all those queries. This is in contrast to what has been observed above for the CRPQ-definability problem, as the queries that can distinguish each individual pair in the set of pairs given as input, cannot necessarily be combined into a single CRPQ.

A formula $\psi(\overline{x})$ is a *union of CRPQs* (UCRPQs) if it is of the following form:

$$\varphi_1(\overline{x}) \vee \dots \vee \varphi_k(\overline{x})$$

where each φ_i is a CRPQ. Let $G = (V, E)$ be a graph, a tuple of nodes $(v_1, \dots, v_n) \in V^n$ satisfies the query $\psi(\overline{x}) =$

$\varphi_1(\overline{x}) \vee \cdots \vee \varphi_k(\overline{x})$ if it satisfies one of the φ_i. Then $q_\psi(G)$ is defined as $\bigcup_{i=1}^k q_{\varphi_i}(G)$.

The following lemma essentially shows that homomorphisms preserve paths.

Lemma 4.14. *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two graphs and let h be a homomorphism from G to H. Then $L_G(v_1, v_2) \subseteq L_H(h(v_1), h(v_2))$, for any two nodes v_1, $v_2 \in V_G$.*

Next, we observe that two pairs of nodes can not be distinguished by a CRPQ iff there is an endomorphism mapping one pair to the other.

Lemma 4.15. *Let $G = (V, E)$ be a graph and let (v_1, v_2) and (v_1', v_2') be two pairs of nodes of G. There exists an endomorphism e on G, with $e(v_1) = v_1'$ and $e(v_2) = v_2'$ if and only if the pair (v_1, v_2) is not CRPQ-distinguishable from (v_1', v_2') over G.*

The following lemma will be used in Theorem 4.17.

Lemma 4.16. *Given a graph $G = (V, E)$ and two pairs of nodes $(v_1, v_2), (v_1', v_2') \in V \times V$, checking whether there is an endomorphism h on G with $h(v_1) = v_1'$ and $h(v_2) = v_2'$, is* NP*-complete.*

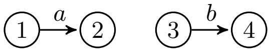
As a consequence of the previous lemmas we show (proof omitted) that UCRPQ-definability is CONP-complete.

Theorem 4.17. DEF(*UCRPQ*) *is* CONP*-complete.*

5. CQ-DEFINABILITY

In this section, we discuss CQ-definability where CQ stands for the class of traditional conjunctive queries [1]. To this end, a graph G is represented in the usual way over the vocabulary $(E_\sigma)_{\sigma \in \Sigma}$ consisting of binary relations E_σ where $E_\sigma(u, v)$ holds in G iff $u \xrightarrow{\sigma}_G v$.

Of course, CQ-definability implies CRPQ-definability. We next, show that the converse does not hold. To this end, consider the graph G_1



with $S_1 = \{(1, 2), (3, 4)\}$. Furthermore, let $\varphi = [a \vee b](x, y)$. Then, $q_\varphi(G_1) = S_1$. It should be obvious that there is no CQ defining S_1. The latter can also be formally proved employing the semantical characterization of CQ-definability in terms of closure under polymorhpisms (see, e.g., [31, 22]).

Specifically, and rephrased in our formalism, it is shown that a relation S is not CQ-definable over a graph G iff there exists a polymorphism of $G = (V, (E_\sigma)_{\sigma \in \Sigma})$ of arity at most n which does not preserve S and where $n = |S|$. To explain the latter statement we need to introduce some terminology. Let $\mathbf{s} = (\mathbf{s}_1, \ldots, \mathbf{s}_k)$ be a k-tuple of elements over V^n. Let $\text{proj}(\mathbf{s}) = \{(\mathbf{s}_1[i], \ldots, \mathbf{s}_k[i]) \mid 1 \leq i \leq n\}$, where $\mathbf{s}[i]$ denotes the ith component of tuple $\mathbf{s}$. Then, a function $h : V^n \to V$ preserves a k-ary relation R at $\mathbf{s}$ if $\text{proj}(\mathbf{s}) \subseteq R$ implies that $(h(\mathbf{s}_1), \ldots, h(\mathbf{s}_k)) \in R$. Furthermore, h preserves R if h preserves R at every k-tuple in $(V^n)^k$. We then say that h is a polymorphism of G (of arity n) if h preserves every E_σ.

We are now ready to prove that (G, S_1) is not CQ-definable. Define $h : V^2 \to V$ as follows: $h(i, i) = i$ for $i \in \{1, \ldots, 4\}$, $h(1, 3) = 2$ and $h(2, 4) = 3$. The remainder of h can be chosen arbitrarily. Observe that h does not preserve S_1. Indeed, take $\mathbf{s} = ((1, 3), (2, 4))$. Then $\text{proj}(\mathbf{s}) = S_1$ but $(h(1, 3), h(2, 4)) = (2, 3) \notin S_1$. Furthermore, h does preserve E_a and E_b. Indeed, the only sequence $\mathbf{s}$ for which $\text{proj}(\mathbf{s}) \subseteq E_a$ is $\mathbf{s} = ((1, 1), (2, 2))$ and $(h(1, 1), h(2, 2)) = (1, 2) \in S_1$. A similar reasoning applies for E_b. It then follows that (G, S_1) is not CQ-definable

As mentioned in the introduction, CQ-definability is shown to be complete for CONEXPTIME. The above shows that neither the upper bound nor the lower bound can be directly carried over to determine the complexity of CRPQ-definability.

Denote by UCQ the class of unions of conjunctive queries. We argue that UCQ-definability coincides with UCRPQ-definability. It suffices to show that on a given graph G, every CRPQ φ can be rewritten into an equivalent UCQ ψ, that is, $q_\varphi(G) = q_\psi(G)$. Thereto, let $\varphi = \exists \bar{z}(r_1(z_1, z_1') \wedge \cdots \wedge r_n(z_n, z_n'))$ be a CRPQ. From Proposition 3.1 it follows that for every r_i there is a set of words $W_i = \{w_{i,1}, \ldots, w_{i,k_i}\}$ with $q_{\mathcal{L}(r_i)}(G) = q_{W_i}(G)$. Denote by K_i the set $\{1, \ldots, k_i\}$, and let $K = K_1 \times \cdots \times K_n$. Then, define

$$\psi := \bigvee_{(j_1, \ldots j_n) \in K} \exists \bar{z}(w_{1,j_1}(z_1, z_1') \wedge \cdots \wedge w_{n,j_n}(z_n, z_n')).$$

Obviously, every conjunct $w(z, z')$ can be replaced by the CQ $\exists \bar{x} E_{\sigma_1}(z, x_1) \wedge \ldots \wedge E_{\sigma_n}(x_{n-1}, z')$ for $w = \sigma_1 \cdots \sigma_n$. It now follows that $q_\varphi(G) = q_\psi(G)$.

Therefore, we have the following corollary:

Corollary 5.1. DEF(*UCQ*) *is* CONP*-complete.*

6. RELATED WORK

As already mentioned in the Introduction, the present paper is similar in spirit to the work on language completeness of Banchilhon [2] and Paredaens [27] but then from a complexity point of view. Geerts and Poggi [18] study BP-completeness in the context of $\mathcal{K}$-relations, an extension of the relational model where tuples are assigned a unique value in a semiring [19]. CQ-definability has been studied in the context of constraint languages under a variety of names including the expressibility problem [31], PP-definability problem [29], and existential inverse satisfiability problem [12, 11]. CQ-definability is an instance of the structure identification problem [13] which asks whether a given relation can be "represented" by a formula in some logical formalism and has been shown to be CONEXPTIME-complete by Willard [31]. See also the note of Ten Cate and Dalmau [30] for a relationship with the product homomorphism problem which implies that CQ-definability is already CONEXPTIME-hard for unary queries over a fixed schema consisting of a single binary relation. Moreover, definability can be semantically characterized in terms of closure under polymorphisms [31, 22]. It would be interesting to come up with such language independent characterizations for CRPQ as well.

There has been a renewed interest in studying navigational query languages over graphs (see, e.g., [15, 21]). Losemann and Martens [25] study the semantics of property paths as defined in SPARQL [20]. Property paths are fundamental to SPARQL and are of the form $x \cdot r \cdot y$ where x and y are variables (to be interpreted by nodes), and r is a regular expression. Specifically, they study the complexity of deciding whether there exists a path from x to y matching r and the complexity of counting the number of paths from x to y matching r. In particular, they observe that in general adopting a simple walk semantics, where paths can not revisit nodes, increases the complexity. Evaluation of regular expressions over graphs under the simple path semantics has been studied before by Mendelzon and Wood [26]. Initially, conjunctive regular path queries have been studied mainly from the angle of query containment [9, 17, 14]. More recently, the conjunctive regular paths queries received renewed attention especially w.r.t. the complexity of query answering and expressiveness [5, 24, 4, 3].

7. CONCLUSIONS

We conclude by a discussion of some directions for future work. One immediate direction is the search for tractable cases. A recipe for tractability, of course, is to consider classes of languages/queries for which there are at most polynomially many elements which could distinguish any (G, S). Indeed, then one could simply enumerate all members and test if they define S. Examples are, for instance, SOREs when Σ is considered fixed. Another possibility for future work is to find matching bounds in the case of CRPQs and to try to find semantical characterizations for CRPQ-definability.

8. REFERENCES

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

[2] F. Bancilhon. On the completeness of query languages for relational data bases. In *MFCS*, pages 112–123. Springer, 1978.

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

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

[5] P. Barceló, J. Pérez, and J. L. Reutter. Relative expressiveness of nested regular expressions. In *AMW*, pages 180–195. CEUR-WS.org, 2012.

[6] G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. *ACM Transactions on the Web*, 4(4), 2010.

[7] G. J. Bex, F. Neven, T. Schwentick, and S. Vansummeren. Inference of concise regular expressions and DTDs. *ACM Transactions on Database Systems*, 35(2), 2010.

[8] P. Buneman, W. Fan, and S. Weinstein. Query optimization for semistructured data using path constraints in a deterministic data model. In *DBPL*, pages 208–223, 2000.

[9] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, pages 176–185, 2000.

[10] D. Colazzo, G. Ghelli, and C. Sartiani. Efficient inclusion for a class of XML types with interleaving and counting. *Inf. Syst.*, 34(7):643–656, 2009.

[11] N. Creignou, P. G. Kolaitis, and B. Zanuttini. Structure identification of boolean relations and plain bases for co-clones. *J. Comput. Syst. Sci.*, 74(7):1103–1115, 2008.

[12] V. Dalmau. Computational complexity of problems over generalized formulas, 2000. PhD thesis, Universitat Politécnica de Catalunya.

[13] R. Dechter and J. Pearl. Structure identification in relational data. *Artif. Intell.*, 58(1-3):237–270, 1992.

[14] A. Deutsch and V. Tannen. Optimization properties for classes of conjunctive regular path queries. In *DBPL*, pages 21–39. Springer, 2001.

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

[16] G. H. L. Fletcher, M. Gyssens, J. Paredaens, and D. V. Gucht. On the expressive power of the relational algebra on finite sets of relation pairs. *IEEE Trans. Knowl. Data Eng.*, 21(6):939–942, 2009.

[17] D. Florescu, A. Y. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *PODS*, pages 139–148, 1998.

[18] F. Geerts and A. Poggi. On database query languages for k-relations. *J. Applied Logic*, 8(2):173–185, 2010.

[19] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS 2007*, pages 31–40. ACM, 2007.

[20] S. Harris and A. Seaborne. SPARQL 1.1 query language. Tech. report, World Wide Web Consortium (W3C), January 2012.

[21] J. Hellings, B. Kuijpers, J. Van den Bussche, and X. Zhang. Walk logic as a framework for path query languages on graph databases. ICDT, 2013.

[22] P. Jeavons, D. A. Cohen, and M. Gyssens. How to determine the expressive power of constraints. *Constraints*, 4(2):113–131, 1999.

[23] D. Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 254–266. IEEE Computer Society, 1977.

[24] L. Libkin and D. Vrgoc. Regular path queries on graphs with data. In *ICDT*, pages 74–85. ACM, 2012.

[25] K. Losemann and W. Martens. The complexity of evaluating path expressions in SPARQL. In *PODS*, pages 101–112. ACM, 2012.

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

[27] J. Paredaens. On the expressive power of the relational algebra. *Inf. Process. Lett.*, 7(2):107–111, 1978.

[28] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9. ACM, 1973.

[29] B. ten Cate. Notes on AIM CSP workshop,[3] 2008.

[30] B. ten Cate and V. Dalmau. A note on the product homomorphism problem and CQ-definability. Manuscript, 2012. http://arxiv.org/abs/1212.3534

[31] R. Willard. Testing expressibility is hard. In *CP*, pages 9–23. Springer, 2010.

[3] `http://www.aimath.org/WWN/constraintsatis/constraintsatis.pdf`