Conjunctive Visibly-Pushdown Path Queries

Martin Lange and Etienne Lozes

School of Elect. Eng. and Computer Science
University of Kassel*, Germany and LSV, ENS Cachan & CNRS, France

Abstract. We investigate an extension of conjunctive regular path queries in which path properties and path relations are defined by visibly pushdown automata. We study the problem of query evaluation for extended conjunctive visibly pushdown path queries and their subclasses, and give a complete picture of their combined and data complexity. In particular, we introduce a weaker notion called extended conjunctive reachability queries for which query evaluation has a polynomial data complexity. We also show that query containment is decidable in 2-EXPTIME for (non-extended) conjunctive visibly pushdown path queries.

1 Introduction

Querying is the central mechanism for extracting information from a knowledge or data base. Queries have therefore been extensively studied in the fields of knowledge representation and database theory. Some of the most important foundational issues regarding queries and their decision problems like query evaluation or query containment concern the decidability, computational complexity and — related to that — expressive power of querying languages.

Graph-structured data [14] and their querying problems occur in many application areas such as semi-structured data the semantic web social networks transportation networks biological networks program analysis etc [2]. A well-established logical formalism for querying graph-structured data is the one of *conjunctive regular path queries* [8, 5, 7]. In this formalism, queries can express conditions on paths in the data-graph by regarding them as words over the alphabet of relation names; it is then possible to ask for instance whether or not there is a path whose associated word belongs to a certain regular language. This formalism has been extended, to context-free languages [10], regular relations [4], and later to rational ones [3], yielding *extended conjunctive regular path queries*. This extension allows to express rich queries, like the existence of two paths of the same length.

In the formalism used by Barceló et al. [4], no two atoms in a non-extended query may use the same path variable. Intuitively, this makes queries easier and less expressive: they merely express *reachability* properties on graphs. We therefore suggest a new 2-dimensional nomenclature for such queries, distinguishing

* The European Research Council has provided financial support under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 259267.

path from *reachability* queries on one hand, and *extended* from *non-extended* queries on the other.

	path query	reachability query
non-extended	CPQ	CRQ
extended	ECPQ	ECRQ

Path queries may contain multiple occurrences of path variables, reachability queries may not. Extended queries speak about relations, non-extended ones about languages. In the notation of Barceló et al.'s [4], their conjunctive path queries correspond to CRQ in our setting, and their extended conjunctive path queries are the same as ours, i.e. ECPQ.

This paper deals with issues of decidability and computational complexity of extended conjunctive path queries over *visibly pushdown languages* (ECPQ[VPL]). Visibly pushdown languages [13, 1] form an interesting class that lies between the regular and context-free ones because it basically has the same closure and decidability properties as the regular ones. Our contributions regarding the decidability and complexity of query evaluation and containment are the following.

1. We show that ECPQ[VPL] query evaluation is undecidable.
2. We show that, for CPQ[VPL] and ECRQ[VPL] queries, query evaluation is P-complete w.r.t. data complexity, thus only a bit more expensive than it is for regular queries (NLOGSPACE).
3. We give upper and lower bounds for the combined complexity of query evaluation for each subclass of queries.
4. We consider the query containment problem; this problem is already undecidable for extended queries in the regular case [9]. So we focus on CRQ[VPL] queries. We show that query containment is decidable among these queries, with a complexity upper bound of 2-EXPTIME, close to the EXPSPACE-complete [5] complexity of query containment for CRQ[REG] queries [8, 5]).

2 Preliminaries

Let $\mathbb{N}$ denote the set of non-negative integers. As usual, Σ denotes a finite alphabet, Σ^* is the set of all finite words over Σ, ε is the empty word. We assume a fixed alphabet Σ for the rest of this paper and note that the concepts introduced herein are to be understood with respect to this alphabet but do not depend on its actual content for as long as $|\Sigma| \geq 2$.

Visibly-Pushdown Relations Let $\perp$ be a new symbol not occurring in Σ, and let $\Sigma_\perp := \Sigma \cup \{\perp\}$. Let $\overline{w} = (w_1, \ldots, w_k) \in (\Sigma^*)^k$, where $w_i = a_{i,1} \cdots a_{i,|w_i|}$ (and all $a_{i,j} \in \Sigma$). We define the string $[\overline{w}] \in (\Sigma_\perp^k)^*$ by $[\overline{w}] := b_1 \cdots b_n$, where n is the maximum of all $|w_i|$, and $b_j := (b_{j,1}, \ldots, b_{j,k})$, with $b_{j,i} = a_{i,j}$ if $j \leq |w_i|$, and $b_{j,i} = \perp$ if $j > |w_i|$. Intuitively, $[\overline{w}]$ is obtained by aligning all w_i to the left, and padding the unfilled space with $\perp$ symbols.

An alphabet Σ' is a visibly pushdown alphabet if it is partitioned into push, pop, and no-op symbols. A *visibly-pushdown automaton* [13, 1] (VPA) is a nondeterministic pushdown automaton whose stack action (push, pop, no-op) is determined by the input letter they read, according to its type push, pop or no-op. A k-ary relation $R \subseteq (\Sigma^*)^k$ is called a *visibly pushdown relation* if $(\Sigma^k)_\perp$ is a visibly pushdown alphabet and $\{[\overline{w}] \mid \overline{w} \in R\}$ is recognised by a VPA.

DB-Graphs A *Σ-labeled db-graph* (db-graph for short) is a directed graph $G = (V, E)$, where V is a finite set of nodes, and $E \subseteq V \times \Sigma \times V$ is a finite set of directed edges with labels from Σ. A *path* ρ between two nodes v_0 and v_n in G with $n \geq 0$ is a sequence $v_0 a_1 v_1 \dots v_{n-1} a_n v_n$ with $(v_i, a_{i+1}, v_{i+1}) \in E$ for $0 \leq i < n$. We define the *label* $\lambda(\rho)$ of the path ρ by $\lambda(\rho) := a_1 \cdots a_n$.

Extended Conjunctive Path Queries We generalise the definition of extended conjunctive regular path queries [4] to visibly pushdown relations. Fix a countable set of *node variables* and a countable set of *path variables*. Let $k \geq 1$. A k-dimensional *extended conjunctive visibly pushdown path query* Q is an expression of the form

$$\mathrm{Ans}(\overline{z}, \overline{\chi}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, \pi_i, y_i), \bigwedge_{1 \leq j \leq l} R_j(\overline{\omega}_j) \ , \tag{1}$$

such that $m \geq 1$, $l \geq 0$, and

1. there is a fixed partition of the alphabet $(\Sigma_\perp)^k$ into push, pop, and no-op symbols
2. each R_j is a k-dimensional visibly pushdown relation
3. $\overline{x} = (x_1, \dots, x_m)$ and $\overline{y} = (y_1, \dots, y_m)$ are tuples of (not necessarily distinct) node variables,
4. $\overline{\pi} = (\pi_1, \dots, \pi_m)$ is a tuple of distinct path variables,
5. $\overline{\omega}_1, \dots, \overline{\omega}_l$ are tuples of path variables, such that each $\overline{\omega}_j$ is a tuple of variables from $\overline{\pi}$, of the same arity as R_j,
6. $\overline{z}$ is a tuple of node variables among $\overline{x}$, $\overline{y}$, and
7. $\overline{\chi}$ is a tuple of path variables among those in $\overline{\pi}$.

The expression $\mathrm{Ans}(\overline{z}, \overline{\chi})$ is the *head*, and the expression to the right of $\leftarrow$ is the *body* of Q. If $\overline{z}$ and $\overline{\chi}$ are the empty tuple (i. e., the head is of the form $\mathrm{Ans}()$), Q is a *Boolean query*. The *relational* part of Q is $\bigwedge_{1 \leq i \leq m}(x_i, \pi_i, y_i)$, and the *labeling* part is $\bigwedge_{1 \leq j \leq l} R_j(\overline{\omega}_j)$. We denote the set of node variables in Q by $\mathrm{nvar}(Q)$. The size of the query is defined as $m + \sum_{1 \leq i \leq n} |R_i|$, where $|R_i|$ denotes the size (number of states and transitions) of the VPA representing the relation R_i.

ECPQ[VPL] will denote the class of all extended conjunctive visibly pushdown path queries, and CPQ[VPL] will denote the class of queries of dimension $k = 1$. A query is called an extended *reachability* query if $\overline{\omega_i} \cap \overline{\omega_j} = \emptyset$ for $i \neq j$, in other words if every path variable occurs at most in one relation constraint. In that case, we abbreviate $R(\boldsymbol{x}, \boldsymbol{y})$ for $(\boldsymbol{x}, \boldsymbol{\omega}, \boldsymbol{y}) \wedge R(\boldsymbol{omega})$. ECRQ[VPL] will

denote the class of extended visibly pushdown reachability queries. Finally, we write ECPQ[REG] for the class of extended conjunctive regular path queries, i.e. extended path queries where all relations are regular.

Example 1. Let $L_0 := \{a^n b^n \in \Sigma^* \mid n \geq 0\}$, $L = L_0^*$, and $R = \{(w_1, w_2) \in L \times L \mid w_1 \neq w_2\}$. The query $\text{Ans}(x, y) \leftarrow (x, \pi, y) \wedge L^*(\pi)$ is a CRQ[VPL] query. The query $\text{Ans}(x, y) \leftarrow (x, \pi_1, y) \wedge (x, \pi_2, y) \wedge R(\pi_1, \pi_2)$ is an ECRQ[VPL] query that would be hard to express as a CRQ[VPL] query or an ECPQ[REG] query. Finally, the query $\text{Ans}(x, y) \leftarrow (x, \pi_1, y) \wedge (x, \pi_2, y) \wedge (x, \pi_3, y_3) \wedge R(\pi_1, \pi_2) \wedge R(\pi_1, \pi_3) \wedge R(\pi_2, \pi_3)$ is an ECPQ[VPL] query.

Remark 1. Since visibly pushdown languages are closed under intersection, every CPQ[VPL] query is equivalent to a CRQ[VPL] query.

Query Evaluation and Query Containment The evaluation $Q(G)$ of a query Q over a db-graph G is intuitively obtained by interpreting all variables as quantified existentially, and path constraints as constraints on the words formed by the labels along the paths. Formally, for every db-graph G, every ECPQ[VPL] query Q (of the form described in (1)), every mapping σ from the node variables of Q to nodes in G, and every mapping μ from the path variables of Q to paths in G, we write $G, \sigma, \mu \models Q$ if

1. $\mu(\pi_i)$ is a path from $\sigma(x_i)$ to $\sigma(y_i)$ for every $1 \leq i \leq m$,
2. for each $\overline{\omega}_j = (\pi_{j_1}, \ldots, \pi_{j_k})$, $1 \leq j \leq l$, the tuple $(\lambda(\mu(\pi_{j_1})), \ldots, \lambda(\mu(\pi_{j_k})))$ belongs to the relation R_j.

We define the output of Q on G as

$$Q(G) \quad := \quad \{ \ \big(\sigma(\overline{z}), \mu(\overline{\chi})\big) \ \mid \ G, \sigma, \mu \models Q \ \}.$$

The query Q is contained in the query Q' if $Q(G) \subseteq Q'(G)$ for all db-graphs G.

The problem of query evaluation is: given a Boolean query Q and a db-graph G over the same underlying alphabet, decide whether or not $Q(G) = \{()\}$, or in other words, whether there are σ, μ such that $G, \sigma, \mu \models Q$. We distinguish the *combined complexity* from the *data complexity*. The former considers both Q and G to be the input, the latter considers the query to be fixed and measures the complexity of query evaluation only in terms of the size of the db-graph G. For complexity considerations we focus on the decision problem of query evaluation for Boolean queries, but in general, when this problem is decidable, it is possible to compute a representation of the output of a query.

3 The Complexity of Evaluating CPQ[VPL] Queries

We address the problem of evaluation of both CPQ[VPL] and CRQ[VPL] queries w.r.t. combined and data complexity. First note that, since every CPQ[VPL] query can be converted into a CRQ[VPL] query, the data complexity of query evaluation is the same for these two formalisms. It can also be noticed that, for a

given db-graph G, a CRQ[VPL] query Q, and a mapping σ, the model-checking problem $G, \sigma \models Q$ can be solved in polynomial time. Indeed, for two variables x, y, the language $L(\sigma(x), \sigma(y))$ of all paths from $\sigma(x)$ to $\sigma(y)$ is recognized by a NFA of size $|G|$, and G satisfies the atom (x, L, y) iff $L \cap L(\sigma(x), \sigma(y))$ is not empty. Since $G, \sigma \models Q$ can be decided in polynomial time, the data complexity of query evaluation is PTIME – the enumeration of all possible σ takes time $O(|G|^{|\mathsf{nvars}(Q)|})$. This upper bound is actually tight.

Theorem 1. *Evaluating CRQ[VPL] queries is P-complete (data complexity).*

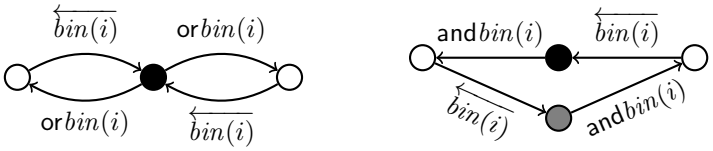
Proof. For a natural number n let $bin(n)$ denote its binary representation using a special symbol to mark the end of the code, e.g. $bin(13) = 1101\$$. Let $\overleftarrow{bin}(n)$ be its representation in reverse order using different symbols, i.e. $\overleftarrow{bin}(13) = \$'1'0'1'1'$. Finally, let $\#$ be an extra symbol. It is not hard to see that the language L_0 described recursively by

$$L_0' \;=\; \mathsf{set} + \mathsf{or}\,\{bin(n)\, L_0'\, \overleftarrow{bin}(n) \mid n \in \mathbb{N}\} + \Big(\mathsf{and}\,\{bin(n)\, L_0'\, \overleftarrow{bin}(n) \mid n \in \mathbb{N}\}\Big)^2$$

as well as $L_0 := \#L_0'$ is a VPL over the visibly pushdown alphabet with push-symbols \$, 1, 0, pop-symbols $\$'$, $1'$ $0'$ and no-op symbols $\#, \mathsf{set}, \mathsf{or}, \mathsf{and}$.

We present a logspace reduction from the Circuit Value Problem, known to be P-complete [12], to the problem of evaluating the query $\mathrm{Ans}() \leftarrow (x, \pi, x) \wedge L_0(\pi)$. We informally describe how to turn a circuit C into a db-graph G_C. W.l.o.g. we can assume that every internal gate in C has fan-in exactly 2, and is identified by a unique number i.

Every input gate that is set to 0 becomes a single node ●. Every input gate that is set to 1 becomes a node ●↩ set . The OR-gate with number i and the AND-gate with number i are translated as the filled nodes below



assuming the non-filled nodes represent the nodes associated to the inputs of these gates, and the gray node is a local auxiliary node. Additionally, we append a loop labeled with $\#$ to the output gate of the circuit. Note that this is a logspace reduction.

Now we have $Q_0(G_C) = \mathtt{true}$ iff there is a path starting and ending in the output gate's node that traverses through the graph such that in every OR-gate node it continues to one successor, and in every AND-gate node it traverses first through one input gate and then, after coming back to it goes through the other successor. The path can only traverse through input gates that are 1. Thus, this is the case iff the circuit evaluates to 1. □

We now consider the combined complexity of query evaluation. Note that even if a CPQ[VPL] query can be converted into an equivalent CRQ[VPL] query,

the original query might be exponentially more concise, so the two families of queries might have different combined complexities – and indeed, they do. Consider first evaluating a CRQ[VPL] query Q. Since $G, \sigma \models Q$ can be decided in polynomial time and σ can be guessed in polynomial time, query evaluation is in NP. Moreover, query evaluation is NP-hard for basic conjunctive queries [6]. As a consequence, CRQ[VPL] query evaluation is NP-complete wrt combined complexity. The situation is rather different for CPQ[VPL] queries.

Theorem 2. *Evaluating CPQ[VPL] queries is EXPTIME-complete (combined complexity).*

Proof. The emptiness of the intersection of a family of VPAs is EXPTIME-complete due to the EXPTIME-completeness of the emptiness problem of a familiy of tree automata. We show that this is also equivalent to CPQ[VPL] query evaluation upto logspace reductions. Assume first a fixed query Q in the form of a conjunct of k queries $Q_1, \ldots, Q_k$ with Q_i of the form $(x_i, \pi_i, y_i) \wedge L_{i,1}(\pi_i) \wedge \cdots \wedge L_{i,n}(\pi_i)$. Then, for every db-graph G and mapping σ, $G, \sigma \models Q$ iff $L(\sigma x, \sigma y) \cap L_{i,1} \cap \ldots L_{i,n} \neq \emptyset$ for all $i = 1, \ldots, k$, which shows the upper bound. Conversely, if $L_1, \ldots, L_n$ is a family of visibly pushdown languages, and if G is the graph restricted to a single node with a self loop labeled with Σ, then $G \models (x, \pi, x) \wedge L_1(\pi) \wedge \cdots \wedge L_n(\pi)$ iff $L_1 \cap \cdots \cap L_n \neq \emptyset$. □

4 The Complexity of Evaluating ECPQ[VPL] Queries

We now address the complexity of evaluating *extended* queries. We just saw that reachability and path queries are equally expressive in the non-extended case. In the extended setting, however, reachability and path queries have important differences. Let us first consider the richest language of extended path queries.

Theorem 3. *Evaluating ECPQ[VPL] is undecidable for dimension* 2 *and two path constraints (data and combined complexity).*

In other words, there is a rquery $Q \in$ ECPQ[VPL] containing two two-dimensional relations $R_1, R_2 \subseteq (\Sigma_\perp^2)^*$, such that query evaluation for Q is undecidable. Note that according to the definition of ECPQ[VPL], $\Sigma_\perp^2$ is partitioned into the same visibly pushdown alphabet for R_1 and R_2, and note moreover that the query Q is fixed.

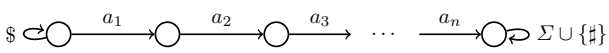
Proof. We claim that there are two visibly pushdown languages L_1, L_2 over two visibly pushdown alphabet Σ_1, Σ_2 containing the same symbols, but with different associated operations, such that the word problem for $L_0 := \{w \mid$ there is v with $wv \in L_1 \cap L_2\}$ is undecidable. Indeed, it is folklore that for every Turing machine M, there are pushdown languages L_1, L_2 such that $L_1 \cap L_2$ is the set of runs of M coded as words, and L_0 can be defined as the set of inputs accepted by a universal Turing machine M.

Now we reduce this problem to the problem of evaluating a fixed ECPQ[VPL] query. Let $\Sigma' := \Sigma \cup \{\$, \sharp\}$ with the two symbols not occurring in Σ. A symbol

$(\$, a)$ of $(\Sigma'_\perp)^2$ is a push (resp. pop, remain) symbol if a is a push (resp. pop, remain) symbol in L_1. Similarly, $(\#, a)$ is a push (resp. pop, remain) symbol if a is a push (resp. pop, remain) symbol in L_2. $w = a_1 \dots a_n \in \Sigma^*$. The fixed query we consider is $Q =$

$$\text{Ans}() \leftarrow (x, \pi, y) \wedge (x, \pi_\$, x) \wedge (y, \pi_\#, y) \wedge (L_1 \times \$^*)(\pi, \pi_\$) \wedge (L_2 \times \#^*)(\pi, \pi_\#).$$

Now, for every db-graph G of the form



it is not hard to see that $w \in L_0$ iff $Q(G) = \texttt{true}$. □

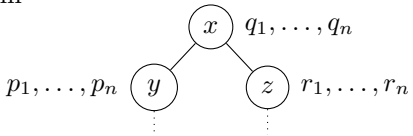
We now turn our attention to extended reachability queries. Evaluating an ECRQ[VPL] query of dimension k over a db-graph G is equivalent to evaluating a CRQ[VPL] query over G^k, which shows by Theorem 1 that the combined complexity of evaluating ECRQ[VPL] queries also is in EXPTIME. The same remark shows that the problem is in PTIME w.r.t. data complexity, and even P-complete by Theorem 1. For the combined complexity, we have get EXPTIME-hardness. The proof relies on EXPTIME-hardness of the following decision problem VpaDfaISect.

- INPUT: a VPA $\mathcal{A}$, and arbitrarily many DFAs $\mathcal{A}_1, \dots, \mathcal{A}_n$
- QUESTION: is $L(\mathcal{A}) \cap L(\mathcal{A}_1) \cap \dots \cap L(\mathcal{A}_n)$ empty?

Lemma 1. VpaDfaISect *is EXPTIME-hard.*

Proof. By a reducton the problem to decide whether or not the intersection of the languages of given top-down tree automata (TA) $\mathcal{A}_1, \dots, \mathcal{A}_n$ over some alphabet Σ is empty. This is known to be EXPTIME-hard [15]. For the sake of simplicity we assume that all the TA share a common state space Q, transition function δ and acceptance set $F \subseteq Q$. The n different TA are then simply given by n different initial states $q_1^I, \dots, q_n^I$. W.l.o.g. we furthermore assume that the underyling alphabet Σ has binary symbols a,b and a unique leaf symbol c, and that trees have height (number of nodes) at least 2.

Next we consider a particular encoding of trees t with node labels of the form $\Sigma \times Q^n$ as a word $\mathsf{rep}(t)$ over the alphabet $\widehat{\Sigma} = \{\overrightarrow{(x,y)}, \overleftarrow{(x,y)} \mid x, y \in \Sigma \text{ or } x, y \in Q\}$. This can easily be done by induction on the structure of t. If t is the single-leaf tree with Σ-label c and any label from Q^n then $\mathsf{rep}(t)$ is the empty word. If t is of the form



and t' and t'' are the left and right subtrees starting with y and z respectively, then we have

$$\mathsf{rep}(t) = \overrightarrow{\binom{x}{y}}\overrightarrow{\binom{q_1}{p_1}}\dots\overrightarrow{\binom{q_n}{p_n}}\ \mathsf{rep}(t')\ \overleftarrow{\binom{q_n}{p_n}}\dots\overleftarrow{\binom{q_1}{p_1}}\overleftarrow{\binom{x}{y}}\overrightarrow{\binom{x}{z}}\overrightarrow{\binom{q_1}{r_1}}\dots\overrightarrow{\binom{q_n}{r_n}}\ \mathsf{rep}(t'')\ \overleftarrow{\binom{q_n}{r_n}}\dots\overleftarrow{\binom{q_1}{r_1}}\overleftarrow{\binom{x}{z}}$$

It should be clear that the language $\{\mathsf{rep}(t) \mid t \text{ is a } \Sigma \times Q^n\text{-labelled tree } \}$ is recognisable by a VPA $\mathcal{B}$ over the visibly pushdown alphabet with push-symbols of the form $\overrightarrow{(\cdot,\cdot)}$, pop-symbols of the form $\overleftarrow{(\cdot,\cdot)}$ and no internal symbols. It uses $\widehat{\Sigma}$ as the stack alphabet and can then easily check the symmetries in the given word. It needs $n+1$ states to check that the length of each segment encoding a node label is $n+1$ while it pushes such an encoding onto the stack, and one more state in which it pops such an encoding and compares it to the $\overleftarrow{(\cdot,\cdot)}$-parts of the word.

It remains to be seen that on such words a DFA $\mathcal{B}_i$ can check whether or not the labelling in the i-th component of the state tuple forms an accepting run of the TA $\mathcal{A}_i$ on the tree given by the Σ-labels. We describe its behaviour informally, using the term *segment* to denote each part of length $n+1$ in such a word that enocdes the labels of two consecutive nodes in the tree.

1. It discards if the symbol at position $i+1$ of the entire word is not of the form $\overrightarrow{(q_i^I,\cdot)}$, i.e. the run does not start in the initial state. Otherwise it continues.
2. On a $\overrightarrow{\cdot}$-segment it remembers the symbol $\overrightarrow{(q,p)}$ at position $i+1$. If this segment is followed by another $\overrightarrow{\cdot}$-segment then it discards this information and continues. Otherwise the second components in this segment encode the label of a leaf node, and $\mathcal{B}_i$ discards if $p \notin F$. Otherwise it continues.
3. On a $\overleftarrow{\cdot}$-segment it remembers the symbols $\overleftarrow{(q,p)}$ and $\overleftarrow{(a,b)}$ at positions $n+1-i$ and $n+1$. If this segment is followed by another $\overleftarrow{\cdot}$-segment then it discards this information because this segment encodes the labels on a node and its right child. Otherwise this segment is followed by a $\overrightarrow{\cdot}$-segment, and these two encode the labels on a node with Σ-label a and its two children. $\mathcal{B}_i$ then also reads the symbol $\overrightarrow{(q',r)}$ at position i of this successor segment and discards if $q \neq q'$ or $(p,r) \notin \delta(q,a)$. Otherwise it continues reading the next segment.

It should be clear that each $\mathcal{B}_i$ only needs polynomial size in $\mathcal{A}_i$ and n. Moreover, we get that $L(\mathcal{B}) \cap \bigcap_{i=1}^n L(\mathcal{B}_i) \neq \emptyset$ iff there is a word $\mathsf{rep}(t)$ in this intersection which encodes a $\Sigma \times Q^n$-labelled tree such that the $(i+1)$-st components of the labels form an accepting run of the TA $\mathcal{A}_i$ on the tree given by the Σ-labels. This is the case iff $\bigcap_{i=1}^n L(\mathcal{A}_i) \neq \emptyset$. □

Theorem 4. *Evaluating ECRQ[VPL] queries is EXPTIME-hard (combined complexity).*

Proof. By a reduction from VpaDfaISect. Let $\mathcal{A}$ be a VPA and $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be n DFAs. We then construct a db-graph $G_{\mathcal{A}_1,\ldots,\mathcal{A}_n}$ as their disjoint union and add a self-loop labeled with a new symbol start_i to every starting state in $\mathcal{A}_i$ as well as a self-loop labeled with a new symbol end to each of its final states. Then consider the n-ary relation

$$R_n = \left\{ \begin{pmatrix} \mathsf{start}_1 \\ \vdots \\ \mathsf{start}_n \end{pmatrix} \begin{pmatrix} w \\ \vdots \\ w \end{pmatrix} \begin{pmatrix} \mathsf{end} \\ \vdots \\ \mathsf{end} \end{pmatrix} \mid w \in L(\mathcal{A}) \right\}.$$

Note that it can be recognised by a VPA $\mathcal{A}'$ of linear size in the size of $\mathcal{A}$. Then $L(\mathcal{A}) \cap \bigcap_{i=1}^{n} L(\mathcal{A}_i) \neq \emptyset$ iff $G_{\mathcal{A}_1,\ldots,\mathcal{A}_n} \models \bigwedge_{i=1}^{n}(x_i, \pi_i, y_i) \wedge R_n(\pi_1, \ldots, \pi_n)$. □

5 Query Containment

We now consider the problem of query containment. For ECRQ[REG] (extended conjunctive regular reachabilty queries), Freydenberger and Schweikardt showed that this problem is undecidable. In the remainder, we focus on query containment for CRQ[VPL] queries where no path variables occur in the head. That is, we consider the problem $Q_1 \subseteq Q_2$ for two CRQ[VPL] queries Q_1, Q_2 of the form $Q_h =$

$$\mathrm{Ans}(z_1, \ldots, z_n) \leftarrow \bigwedge_{i=1\ldots m_h} (x_{h,i}, L_{h,i}, y_{h,i}).$$

Note that the distinguished variables $z_1, \ldots, z_n$ are the same for the two queries. We assume that these are the only variables shared by the queries – the non-distinguished variables are assumed properly renamed. We also assume that the languages $L_{h,i}$ are recognised by VPAs working with a fixed visibly pushdown alphabet Σ. The set $\mathcal{V}_i := \mathsf{nvars}(Q_i)$ is the set of all variables of Q_i. A tuple $G = (V, E, \sigma)$ is a *canonical candidate* of Q_1 if $\sigma : \mathcal{V}_1 \to V$ and G is the union of m_1 cycle-free paths $\pi_1, \ldots, \pi_n$, one for each conjunct of Q_1, such that π_i goes from $\sigma(x_i)$ to $\sigma(y_i)$. A canonical candidate of Q_1 is a *canonical model* of Q_1 if moreover $\lambda(\pi_i) \in L_{1,i}$. In particular, if G is a canonical model of Q_1, then $\sigma(\boldsymbol{z}) \in Q_1(G)$. The following result has been proved in several places, see for instance [11].

Lemma 2. *$Q_1 \subseteq Q_2$ iff for all canonical model $G = (V, E, \sigma)$ of Q_1, there is $\sigma' : \mathcal{V}_2 \to V$ such that $G, \sigma' \models Q_2$ and $\sigma(z) = \sigma'(z)$ for all distinguished variables z.*

We now follow the automata-based approach [5] for deciding query containment. The main idea of this approach is that canonical db-graphs can be represented by means of words. To every canonical db-graph $G = (V, E, \sigma)$, we associate the word $w_G := \$d_1 w_1 d_1' \$ d_2 w_2 d_2' \$ \ldots \$ d_{m_1} w_{m_1} d_{m_1}' \$$ where the new symbol $\$ \notin \Sigma$ acts as a separator and $d_1, d_1', \ldots, d_{m_1}, d_{m_1}'$ range over $\mathcal{D}_1 := 2^{\mathcal{V}_1}$. The i-th block of w_G contains $\$ d_i w_i d_i' \$$ if $d_i = \sigma^{-1}(\sigma(x_{1,i}))$, $d_i' = \sigma^{-1}(\sigma(y_{1,i}))$, and w_i labels the path associated to the atom $(x_{1,i}, L_{1,i}, y_{1,i})$. In the extended visibly pushdown alphabet $\Sigma \cup \{\$\} \cup \mathcal{D}_1$, the symbols $\$$ and $\mathcal{D}_1$ are considered no-op symbols.

Lemma 3. *Let Q_1 be a query of size n. Then $L(Q_1) := \{w_G \mid G$ is a canonical model of $Q_1\}$ is recognised by a VPA with $2^{O(n \log n)}$ states.*

Proof. Let L_{part} be the language of words w over $\Sigma \cup \{\$\} \cup \mathcal{D}_1$ such that the set of $\mathcal{D}_1$ symbols of w define a partition of $\mathcal{D}_1$. Then L_{part} is recognised by an NFA that guesses the partition P of $\mathcal{V}_1$ and then checks that the set of $\mathcal{D}_1$ symbols of

w is P. Since there are $2^{O(|\mathcal{V}_1|\log(|\mathcal{V}_1|))}$ different partitions of $\mathcal{V}_1$, the automaton is of size $2^{O(n\log n)}$.

Let $\{x\}^\uparrow$ denote the set of $\mathcal{D}_1$ symbols d such that $x \in d$. The language $L_{\mathsf{query}} := \$\{x_1\}^\uparrow L_{1,1}\{y_1\}^\uparrow\$ \dots \$\{x_{m_1}\}^\uparrow L_{1,m_1}\{y_{m_1}\}^\uparrow\$$ is recognised by a VPA with $O(|Q_1|)$ states. Since $L(Q_1) = L_{\mathsf{part}} \cap L_{\mathsf{query}}$, this shows the result. □

Let $G = (V, E, \sigma)$ be a canonical db-graph of Q_1, and let $\sigma' : \mathcal{V}_1 \cup \mathcal{V}_2 \to V$ be an extension of σ. The pair (G, σ') can be represented as a word $w_{G,\sigma'}$ over the alphabet $\Sigma \cup \$ \cup \mathcal{D}_2$ with $\mathcal{D}_2 := 2^{\mathcal{V}_1 \cup \mathcal{V}_2}$ following the same idea as before, except that now symbols $d \in \mathcal{D}_{2\setminus 1} := 2^{\mathcal{V}_2 \setminus \mathcal{V}_1}$ can occur anywhere – a variable $x \in \mathcal{V}_2 \setminus \mathcal{V}_1$ can be mapped to a node $\sigma'(x)$ that is not in the image of σ. Let $\pi_{2\to 1}$ be the substitution that sends $d \in \mathcal{D}_2$ to $d \cap V_1$ if $d \cap V_1 \neq \emptyset$, ϵ otherwise. Note that if L is definable by a VPA, then $\pi_{2\to 1}(L)$ is definable by a VPA, because $\pi_{2\to 1}$ only changes or erases no-op symbols.

Let $L(Q_2|Q_1)$ be the set of words $w_{G,\sigma'}$ such that $G, \sigma' \models Q_2$ and G is a canonical candidate for Q_1. Then, by virtue of Lemma 3, $Q_1 \subseteq Q_2$ iff $L(Q_1) \subseteq \pi_{2\to 1}(L(Q_2|Q_1))$.

Example 2. Consider the query Q_1 whose unique canonical model is the graph G such that $w_G = \$\{x_1\}aa\{y_1\}\$\{x_2\}cc\{y_2\}\$\{y_1\}b\{x_2\}\$$. Let Q_2 be the query (x, abc, z). Then $G, \sigma' \models Q_2$ for σ' such that $w_{G,\sigma'} =$

$$\$\{x_1\}a\{x\}a\{y_1\}\$\{x_2\}c\{z\}c\{y_2\}\$\{y_1\}b\{x_2\}\$,$$

so $Q_1 \subseteq Q_2$ (assuming G is the unique canonical model of Q_1).

As seen in the example above, it is not straightforward to recognise $L(Q_2|Q_1)$ with a VPA, because checking the atom (x, L, y) requires an L-path to be guessed that might be coded in different blocks, in any order, and the extremities of this path can be at intermediate positions inside some blocks. This motivates the following definition.

Definition 1. *A* jumping *visibly pushdown automaton (JPA) over a visibly pushdown alphabet Σ is a tuple $\mathcal{A} = (Q, \Gamma, \delta, q_I, F, J)$ such that $(Q, \Gamma, \delta, q_I, F)$ is a VPA and $J \subseteq Q$ is a set of* jumping states*.*

Intuitively, a JPA reads a word on a tape from left to right managing the stack as usual, but when it enters a jumping state, the head of the tape can instantaneously move to any position, while erasing the content of the stack. Formally, a configuration of the run of a jumping VPA over a nested word $w \in \Sigma^*$ is a tuple $(q, s, i) \in Q \times \Gamma^* \times \{1, \dots, |w|\}$; it is accepting if $q \in F$. The configuration (q, s, i) leads to the configuration (q', s', i'), $(q, s, i) \vdash_w (q', s', i')$, if there is a transition from $(q, a, \alpha, q') \in \delta$ such that a is the i-th symbol of w, α is some stack action that transforms s into $\alpha(s)$, and either $i' = i + 1$ and $s' = \alpha(s)$, or $q' \in J$ and $s' = \bot$. A nested word w is accepted if $(q_I, \bot, 1) \vdash_w^* (q, s, i)$ for some accepting configuration (q, s, i).

Lemma 4. *$L(Q_2|Q_1)$ is recognised by a JPA with $O(|Q_1| + |Q_2|)$ states.*

Proof. We first assume that Q_2 contains only one atom (x, L, y). Let $\mathcal{A} = (Q, \Sigma, \delta, q_I, F)$ be the VPA representing L. We informally define a JPA $\mathcal{B}$ that accepts $w_{G,\sigma'}$ iff $G, \sigma' \models (x, L, y)$. In the first step, the automaton jumps anywhere in the word. Then it checks that the symbol it reads is a $\mathcal{D}_2$ symbol d that contains x, and start running $\mathcal{A}$. When it meets a $\mathcal{D}_2$ symbol d, it may jump anywhere else provided the next symbol it will read is a $\mathcal{D}_2$ symbol d' such that $d \cap d' \cap \mathcal{V}_1 \neq \emptyset$. It accepts when it reads a symbol d that contains y.

Clearly, if $w_{G,\sigma'}$ is accepted by $\mathcal{B}$, the piece of w_G over which $\mathcal{B}$ ran corresponds to a path satisfying the atom (x, L, y). Conversely, assume that $G, \sigma' \models (x, L, y)$. Then there is a word $w_1 w_2 \dots w_n$ in L and blocks indices $i_1, \dots, i_n$ such that w_1 is a suffix of the word of i_1th block of $w_{G,\sigma'}$, $w_2, \dots, w_{n-1}$ are the words of the blocks $i_2, \dots, i_{n-1}$, and w_n is a prefix of the word of the i_nth block. To ensure that an accepting run of $\mathcal{A}$ over $w_1 w_2 \dots w_n$ can be mimicked by an accepting run of $\mathcal{B}$, it must be checked that the w_i are well nested, so that it does not harm to erase the stack when jumping. For $k = 2, \dots n-1$, w_k is well-nested because it belongs to the visibly pushdown language L_{1,i_k}, and w_1 and w_n are also well-nested because they are prefixes of well-nested words of L and L_{1,i_n} respectively.

Let us assume now that Q_2 is $\bigwedge_{i=1}^{n}(x_i, L_i, y_i)$, and let $\mathcal{B}_i$ be the JPA associated to the ith atom following the previous construction. Then we define $\mathcal{B}$ as the automaton that executes B_1, then B_2, ..., then B_n. Clearly $|\mathcal{B}| = O(|\mathcal{B}_1| + \dots + |\mathcal{B}_n|)$ and $L(\mathcal{B}) = \bigcap_{i=1}^{n} L(\mathcal{B}_i)$, which shows the result. □

Observe now that if a JPA has an accepting run over a word w, then it has an accepting run over w with at most $|J|$ jumps, where J is its set of jumping states. This leads to the following result.

Lemma 5. *For every JPA $\mathcal{A}$ with n states, there is a VPA $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$ and $|\mathcal{B}| = 2^{O(n \log n)}$.*

Proof. Let $\mathcal{A}$ be a fixed JPA. Observe that if $\mathcal{A}$ has an accepting run over a word w, then it has an accepting run with at most $|J|$ jumps, where J is the set of jumping states. Indeed, if during a run $\mathcal{A}$ jumps in jumping state q from position i_1 to position j_1 and later from i_2 to j_2 in the same jumping state q, then a shorter run is obtained by jumping from i_1 to j_2 directly.

Consider a sequence $S = (q_1, q'_1)(q_2, q'_2) \dots (q_{|S|}, q'_{|S|})$ of $|S| \leq n$ pairs of set of states of $\mathcal{A}$. We say that S is accepting if q_1 is the initial state, $q'_{|S|}$ is accepting, and $q'_i = q_{i+1} \in J$ for each $i = 1, \dots, |S|-1$. There are thus $2^{O(n \log n)}$ accepting sequences. For a fixed sequence S, and for each $i = 1, \dots, |S|$, there is a VPA $\mathcal{A}_{S,i}$ with $O(n)$ states that accepts a word w iff w contains a factor w' and there is a run of $\mathcal{A}$ from q_i to q'_i over w'. There is also a VPA $\mathcal{A}_S$ that accepts $\bigcap_{i=1}^{n} L(\mathcal{A}_{S,i})$ and such that $|\mathcal{A}_S| = 2^{O(n \log n)}$. Let $\mathcal{B}$ be the automaton that accepts $\bigcup \{L(\mathcal{A}_S) \mid S \text{ is an accepting sequence}\}$. Then $L(\mathcal{B}) = L(\mathcal{A})$ and $|\mathcal{B}| = 2^{O(n \log n)}$.

To sum up, Lemmas 4 and 5 show that $L(Q_2|Q_1)$ can be recognised by a VPA of exponential size, and so can $L(Q_1)$ (Lemma 3). Language inclusion

between two VPAs can be decided in exponential time, so the inclusion $L(Q_1) \subseteq L(Q_2|Q_1)$ can be decided in 2-EXPTIME.

Theorem 5. *Containment for CRQ[VPL] queries is decidable in 2-EXPTIME.*

Acknowledgments. We would like to thank Nicole Schweikardt for introducing the world of conjunctive path queries to us and for some interesting discussions on that topic.

References

1. R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. STOC'04*, pages 202–211. ACM, 2004.
2. R. Angles and C. Gutiérrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1), 2008.
3. P. Barceló, D. Figueira, and L. Libkin. Graph logics with rational relations. *Logical Methods in Computer Science*, 9(3), 2013.
4. P. Barceló, L. Libkin, A. W. Lin, and P. T. Wood. Expressive languages for path queries over graph-structured data. *ACM Trans. Database Syst.*, 37(4):31, 2012.
5. D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proc. KR'00*, pages 176–185. Morgan Kaufmann, 2000.
6. A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. STOC'77*, pages 77–90, 1977.
7. A. Deutsch and V. Tannen. Optimization properties for classes of conjunctive regular path queries. In *Proc. DBPL'01*, volume 2397 of *LNCS*, pages 21–39. Springer, 2001.
8. D. Florescu, A. Y. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *Proc. PODS'98*, pages 139–148, 1998.
9. D. Freydenberger and N. Schweikardt. Expressiveness and static analysis of extended conjunctive regular path queries. *J. Comp. Syst. Sci.*, 79(6):892–909, 2013.
10. J. Hellings. Conjunctive context-free path queries. In *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014.*, pages 119–130, 2014.
11. P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proc. PODS'98*, pages 205–213. ACM, 1998.
12. R. E. Ladner. The circuit value problem is log-space complete for P. *SIGACT News*, 6(2):18–20, 1975.
13. K. Mehlhorn. Pebbling mountain ranges and its application to DCFL-recognition. In *Proc. ICALP'80*, volume 85 of *LNCS*, pages 422–435. Springer, 1980.
14. A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. In *Proc. VLDB'89*, pages 185–193. Morgan Kaufmann, 1989.
15. H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal of Computing*, 19(3):424–437, 1990.