# Querying for Paths in Graphs using Context-Free Path Queries

Jelle Hellings
Hasselt University
jelle.hellings@uhasselt.be

October 16, 2018

#### Abstract

Navigational queries for graph-structured data, such as the regular path queries and the context-free path queries, are usually evaluated to a relation of node-pairs (m,n) such that there is a path from m to n satisfying the conditions of the query. Although this relational query semantics has practical value, we believe that the relational query semantics can only provide limited insight in the structure of the graph data. To address the limits of the relational query semantics, we introduce the all-path query semantics and the single-path query semantics. Under these path-based query semantics, a query is evaluated to all paths satisfying the conditions of the query, or, respectively, to a single such path.

While focusing on context-free path queries, we provide a formal framework for evaluating queries on graphs using both path-based query semantics. For the all-path query semantics, we show that the result of a query can be represented by a finite context-free grammar annotated with node-information relevant for deriving each path in the query result. For the single-path query semantics, we propose to search for a path of minimum length. We reduce the problem of finding such a path of minimum length to finding a string of minimum length in a context-free language, and for deriving such a string we propose a novel algorithm.

Our initial results show that the path-based query semantics have added practical value and that query evaluation for both path-based query semantics is feasible, even when query results grow very large. For the single-path query semantics, determining strict worst-case upper bounds on the size of the query result remains the focus of future work.

### 1 Introduction

The graph data model is one of the most versatile and natural data models in use: graph-structured data is everywhere and examples can be found in family trees, social networks, process models, gene networks, XML data, and RDF data [1, 7, 9, 28]. For querying graphs, many different query languages have been developed, proposed, and researched [4, 5, 6, 8, 13, 14, 20, 21]. At their core, most graph query languages depend on navigating the graph. This graph navigation is usually performed by means of a regular expression that describes the allowed edge-labeling of the paths that should be traversed in the graph.

As the regular expressions have limited expressive power, we focus on a more expressive navigational query language, namely the context-free path queries that use context-free grammars to describe the labeling of paths [19, 21, 24, 29].

These navigational queries expressed by context-free path queries are usually evaluated to a relation of node-pairs (m,n) such that there is a path from m to n whose labeling is described by a context-free grammar—the relational query semantics, or to the truth value true whenever such a path exists—the boolean query semantics. Although many practical problems can be answered by navigational queries evaluated under the usual semantics, we believe that the relational query semantics and the boolean query semantics are limiting. The inability to view the paths of interest hampers the understanding of the data, makes query debugging harder, and makes it impossible to answer certain practical problems.

To address the limitations of the traditional query semantics, we introduce path-based query semantics. Concretely, we introduce the *all-path query semantics* and the *single-path query semantics*. Under the all-path query semantics, a query is evaluated to all paths satisfying the conditions of the query, and under the single-path query semantics one such path is chosen. The practical usage of these path-based query semantics can be illustrated by a simple example:

Example 1. Consider a collection of family trees represented by a graph in which the nodes represent peoples and the edges represent parent of and child of relations (between parents and their children). Consider the context-free grammar with the following production rules:

$$q \mapsto parentOf \ q \ childOf$$
,  $q \mapsto parentOf \ childOf$ .

Using the standard relational query semantics, the query  ${\bf q}$  evaluates to the relation of node-pairs (m,n) such that m and n are both k-th generation descendants of a common ancestor. Using the single-path query semantics that we propose, the query  ${\bf q}$  evaluates to a path from m to a common ancestor and from this common ancestor to n, showing why m and n are both k-th generation descendants of a common ancestor, while, at the same time, showing who this common ancestor is.

Observe that the context-free grammar used in Example 1 is well-known to not be expressible by a regular expression [23]. Still, this simple example is at the basis of practical queries that are used in, for example, bio-informatics [29].

For graph querying, path-based query semantics have only gained limited attention. For the regular expressions, Barcel et al. [5] introduced the extended regular path queries that have path variables for output. The main focus of Barcel et al. is, however, on the use of path variables for expressivity purposes, and path-based results are only studied in limited details. Recent work by Hofman et al. [22] provides an alternative to use path-based query semantics for debugging: to gain more insight in the behavior of regular path queries with respect to the expected behavior, Hofman et al. propose a technique based on separability. Although this approach addresses query debugging, it does not lift the other limitations of the relational and the boolean query semantics.

In the setting of model checking using CTL [9], path-based query semantics are widely used. Normally, CTL formulae are evaluated to true or false, indicating if the graph meets or not meets certain conditions. An important ability of CTL model checking algorithms is to not only answer CTL formulae with a

truth value, but to also answer with a witnesses or a counterexample for this truth value. These witnesses and counterexamples are represented by a path in the graph that shows why the graph does or doesn't meet the conditions expressed by the CTL formulae. Counterexamples and witnesses also exists for other modal logics, such as LTL. These path-based witnesses and counterexamples are especially useful in the analysis of the model checking results.

In this work we study path-based query semantics. We provide a formal framework for evaluating queries on graphs using the all-path query semantics and the single-path query semantics. To achieve this, we first show how to represent the query result under the all-path query semantics by a context-free grammar annotated with node-information, and we show that this context-free grammar can be used to derive exactly those paths that are in the query result.

For the single-path query semantics, we propose to search for a path of minimum length. As we can represent the set of all paths by an annotated context-free grammar, we reduce the problem of finding a path of minimum length matching the query conditions to finding a string of minimum length in a context-free language. For deriving such a string of minimum length, we propose a novel algorithm. We then proceed with the analysis of this minimum-length string derivation algorithm applied to annotated context-free grammars by analyzing the possible length of minimum-length paths. For annotated context-free grammars over the singleton alphabet, we show a close-to-strict worst-case upper bound on the length of minimum-length paths that is linear in the number of nodes in the graph. For general annotated context-free grammars we show that the worst-case upper bound on the length of minimum-length paths is at least quadratic in the number of nodes in the graph.

To test the behavior of the minimum-length path derivation algorithm in practice, we performed measurements on an initial implementation. These results show promise, as the initial implementation shows acceptable performance for a range of context-free path queries.

**Organization** In Section 2, we present the basic notions used throughout this paper. In Section 3, we present the context-free path queries together with their usual semantics, and we introduce the all-path and single-path query semantics. In Section 4 and Section 5 we introduce approaches to evaluate queries using the all-path query semantics and the single-path query semantics, respectively. In Section 6, we present our results on a small-scale implementation. In Section 7, we summarize our findings and propose directions for future work.

### 2 Preliminaries

We call a sequence  $\sigma_1 \dots \sigma_n$  of symbols a string. The length of string  $S = \sigma_1 \dots \sigma_n$ , denoted by ||S||, is n. The empty string is denoted by  $\varepsilon$  and we usually treat individual symbols as strings of length one. The concatenation of two strings  $S_1$  and  $S_2$  is denoted by  $S_1 \cdot S_2$ . If  $\Sigma$  is a set of symbols, then we denote the set of all strings made of symbols from  $\Sigma$  by  $\Sigma^*$ .

**Definition 1.** A graph is a triple  $G = (Q, \Sigma, \delta)$  with  $Q \cap \Sigma = \emptyset$ , in which Q is a finite set of nodes,  $\Sigma$  is a finite set of alphabet symbols used as edge labels, and  $\delta \subseteq Q \times \Sigma \times Q$  is a finite set of labeled edges.

If  $m \in \mathbb{Q}$  and  $\sigma \in \Sigma$ , then  $\delta(m, \sigma) = \{n \mid (m, \sigma, n) \in \delta\}$  denotes those nodes that have an incoming edge labeled with  $\sigma$  originating at m. If  $m \in \mathbb{Q}$  and  $S \in \Sigma^*$ , then

$$\delta^*(m, S) = \begin{cases} \{m\} & \text{if } S = \varepsilon; \\ \bigcup_{n \in \delta(m, \sigma)} \delta^*(n, S') & \text{if } S = \sigma \cdot S'. \end{cases}$$

The language of graph G with respect to  $m, n \in \mathbb{Q}$ , denoted by  $\mathcal{L}(\mathsf{G}; m, n)$ , is defined by

$$\mathcal{L}(\mathsf{G}; m, n) = \{ S \mid S \in \Sigma^* \land n \in \delta^*(m, S) \}.$$

Let  $G = (Q, \Sigma, \delta)$  be a graph. A path  $\pi = n_1 \sigma_1 \dots n_{i-1} \sigma_{i-1} n_i$  in G is a sequence with, for all  $1 \leq j < i$ ,  $(n_j, \sigma_j, n_{j+1}) \in \delta$ . We write  $n_1 \pi n_i$  to indicate that  $\pi$  starts at node  $n_1$  and ends at node  $n_i$ . The trace of  $\pi$  is defined by  $\operatorname{trace}(\pi) = \sigma_1 \dots \sigma_{i-1}$ . Observe that traces are strings over the alphabet  $\Sigma$ .

**Definition 2.** A context-free grammar is a triple  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  with  $\mathbf{N} \cap \Sigma = \emptyset$ , in which  $\mathbf{N}$  is a set of non-terminals,  $\Sigma$  is a finite set of alphabet symbols, and  $\mathbf{P}$  is a set of production rules.<sup>1</sup> In the above, a production rule is of the form  $\mathbf{a} \mapsto \mathbf{b} \ \mathbf{c} \ \text{or} \ \mathbf{a} \mapsto \sigma$ , in which  $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbf{N}$  and  $\sigma \in \Sigma$ .<sup>2</sup>

Production rules are to be interpreted as rewrite rules: if  $S = S_1 \cdot \mathbf{a} \cdot S_2$  is a string with  $S_1, S_2 \in (\mathbf{N} \cup \Sigma)^*$  and  $\mathbf{a} \in \mathbf{N}$ , and if  $(\mathbf{a} \mapsto S') \in \mathbf{P}$ , then S can be rewritten into  $S_1 \cdot S' \cdot S_2$  by application of  $\mathbf{a} \mapsto S'$ . We write  $S \to_{\mathbf{P}}^* S'$  if S can be rewritten into S' by a finite number of rewrites using production rules in  $\mathbf{P}$  and we write  $S \to_{\mathbf{P}}^* S'$  if  $S \to_{\mathbf{P}}^* S'$  and at least one rewrite step is necessary to rewrite S into S'. The language of a context-free grammar  $\mathbf{C}$  with respect to  $\mathbf{a} \in \mathbf{N}$ , denoted by  $\mathcal{L}(\mathbf{C}; \mathbf{a})$ , is defined by

$$\mathcal{L}(\mathbf{C}; \mathbf{a}) = \{ S \mid S \in \Sigma^* \land \mathbf{a} \to_{\mathbf{P}}^* S \}.$$

# 3 Context-free path queries

Let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar with  $\mathbf{a} \in \mathbf{N}$ . We say that  $\mathbf{a}$  is a context-free path query. Usually, these queries are evaluated using the boolean query semantics or the relational query semantics.<sup>3</sup>

- 1. Using boolean query semantics, the query **a** on graph **G** evaluates to the truth value of  $\exists m \exists n \ \mathcal{L}(\mathbf{C}; \mathbf{a}) \cap \mathcal{L}(\mathsf{G}; m, n) \neq \emptyset$ .
- 2. Using relational query semantics, the query **a** on graph **G** evaluates to the binary relation  $\{(m,n) \mid \mathcal{L}(\mathbf{C};\mathbf{a}) \cap \mathcal{L}(\mathsf{G};m,n) \neq \emptyset\}$ .

<sup>&</sup>lt;sup>1</sup>Usually, context-free grammars are defined with a dedicated start non-terminal. It is straightforward to specialize our results on context-free grammars to the setting with a dedicated start non-terminal.

 $<sup>^2</sup>$  To simplify the presentation, we assume that context-free grammars are in Chomsky Normal Form [23], and we exclude the derivation of  $\varepsilon$ . Unless stated otherwise, it is straightforward to generalize our results on context-free grammars to the setting that includes production rules of the form  $\mathtt{a} \mapsto \varepsilon$ .

 $<sup>^3</sup>$ Commonly, relational query semantics is referred to as path query semantics [14]. To avoid confusion with our path-based query semantics, we have chosen for a different naming in this paper.

We study two alternative ways of evaluating queries on graphs: the *all-path* query semantics and the single-path query semantics:

- 3. Using all-path query semantics, the query **a** on graph **G**, with respect to nodes  $m, n \in \mathbb{Q}$ , evaluates to the set of all paths  $m\pi n$  in **G** with  $\mathsf{trace}(\pi) \in \mathcal{L}(\mathbf{C}; \mathbf{a})$ .
- 4. Using single-path query semantics, the query a on graph G, with respect to nodes  $m, n \in \mathbb{Q}$ , evaluates to a single path  $m\pi n$  in G with  $\mathsf{trace}(\pi) \in \mathcal{L}(\mathbf{C}; \mathbf{a})$  (if such a path exists).

The following example illustrates the usages of these query semantics.

Example 2. Let G be a collection of family trees in which the nodes represent people and the edges represent familyOf relations (between parents and their children). We have the context-free grammar with the following production rules:

$$q \mapsto familyOf$$
,  $q \mapsto q q$ .

Depending on the semantics used, the query  ${\tt q}$  evaluated on  ${\tt G}$  answers various questions:

- 1. Using boolean query semantics: 'are there family members in these family trees?'
- 2. Using relational query semantics: 'provide all pairs of people that are related'
- 3. Using all-path query semantics: 'provide every way in which m and n are related.'
- 4. Using single-path query semantics: 'provide a proof that m and n are related,' or 'show how m and n are related.'

# 4 Answering queries using all-path query semantics

If a context-free path query is evaluated on cyclic graphs, then the query result can be an infinite set of paths. Hence, before we look into how to answer a query using the all-path query semantics, we need to determine how to represent such an infinite set of paths using a finite structure. Graphs are strongly related to finite automata and it is well-known that the intersection of the language of a finite automaton and the language of a context-free grammar is itself a language that can be represented by a context-free grammar:

**Lemma 1** (Bar-Hillel et al. [3]). Let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar with  $\mathbf{a} \in \mathbf{N}$  and let  $\mathsf{G} = (\mathsf{Q}, \Sigma, \delta)$  be a graph with  $m, n \in \mathsf{Q}$ . The language  $\mathcal{L}(\mathbf{C}; \mathbf{a}) \cap \mathcal{L}(\mathsf{G}; m, n)$  can be represented by a context-free grammar.

Lemma 1 only guarantees that there is a finite representation of the set of all traces of paths  $m\pi n$  in graph G with  $\mathrm{trace}(\pi) \in \mathcal{L}(\mathbf{C}; \mathbf{a})$ . As several paths can have the same trace, the set of traces cannot be directly mapped to a set

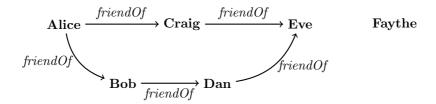


Figure 1: A social network in which persons (represented by nodes) can have friendOf-relations (represented by labeled edges).

of paths. To allow for a direct representation of the set of paths, we show how to construct a context-free grammar that is annotated with node-information relevant for the derivation of paths.

**Definition 3.** Let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar and let  $\mathsf{G} = (\mathsf{Q}, \Sigma, \delta)$  be a graph. We denote triples  $(\mathsf{a}, m, n) \in \mathbf{N} \times \mathsf{Q}^2$  by  $\mathsf{a}[m, n]$ . An annotated grammar over  $(\mathbf{C}, \mathsf{G})$  is a context-free grammar  $\mathbf{C}_{\mathsf{G}} = (\mathbf{N}_{\mathsf{G}}, \Sigma, \mathbf{P}_{\mathsf{G}})$  in which  $\mathbf{N}_{\mathsf{G}} \subseteq \mathbf{N} \times \mathsf{Q}^2$ ; each production rule in  $\mathbf{P}_{\mathsf{G}}$  is of the form  $\mathsf{a}[m, n] \mapsto \mathsf{b}[m, o] \ \mathsf{c}[o, n]$  or  $\mathsf{a}[m, n] \mapsto \sigma$ , with  $\mathsf{a}, \mathsf{b}, \mathsf{c} \in \mathbf{N}, m, n, o \in \mathsf{Q}$ , and  $\sigma \in \Sigma$ ; and that satisfies the following three properties:

- 1.  $a[m, n] \in \mathbf{N}_{\mathsf{G}}$  if and only if  $\mathcal{L}(\mathbf{C}; \mathbf{a}) \cap \mathcal{L}(\mathsf{G}; m, n) \neq \emptyset$ ,
- 2.  $(\mathtt{a}[m,n] \mapsto \mathtt{b}[m,o] \ \mathtt{c}[o,n]) \in \mathbf{P}_\mathsf{G}$  if and only if  $(\mathtt{a} \mapsto \mathtt{b} \ \mathtt{c}) \in \mathbf{P}_\mathsf{G}$
- 3.  $(a[m,n] \mapsto \sigma) \in \mathbf{P}_{\mathsf{G}}$  if and only if  $(m,\sigma,n) \in \delta$  and  $(a \mapsto \sigma) \in \mathbf{P}$ .

We say that a non-terminal  $a[m,n] \in \mathbf{N}_{\mathsf{G}}$  can derive path  $\pi = n_1 \sigma_1 \dots \sigma_{i-1} n_i$  if it can derive the string  $S = \sigma_1 \dots \sigma_{i-1}$  such that, for each  $1 \leq j < i$ , the rewrite step producing  $\sigma_j$  used a production rule of the form  $(\mathfrak{b}[n_j, n_{j+1}] \mapsto \sigma_j) \in \mathbf{P}_{\mathsf{G}}$ .

We illustrate the concept of a annotated grammar with an example:

Example 3. Let G be the social network visualized in Figure 1 in which the nodes represent people and the edges represent friendOf relations. Alice wants to know how she can contact Eve via friends, via friends of friends, and so on. Hence, she writes a context-free grammar C with the following production rules P:

$$q \mapsto friendOf,$$
  $q \mapsto q q.$ 

For brevity, we refer to each person by the first letter of their name. The annotated grammar over (C, G) has the following non-terminals:

$$\mathtt{q}[A,B],\mathtt{q}[A,C],\mathtt{q}[A,D],\mathtt{q}[A,E],\mathtt{q}[B,D],\mathtt{q}[B,E],\mathtt{q}[C,E],\mathtt{q}[D,E].$$

The production rules  $\mathbf{P}_{\mathsf{G}}$  of the annotated grammar consists of the production rules that correspond to  $\mathit{friendOf}$ -edges in the social network:

$$\begin{split} \mathbf{q}[\mathbf{A},\mathbf{B}] &\mapsto \mathit{friendOf}, & \mathbf{q}[\mathbf{A},\mathbf{C}] &\mapsto \mathit{friendOf}, \\ \mathbf{q}[\mathbf{B},\mathbf{D}] &\mapsto \mathit{friendOf}, & \mathbf{q}[\mathbf{C},\mathbf{E}] &\mapsto \mathit{friendOf}, \\ \mathbf{q}[\mathbf{D},\mathbf{E}] &\mapsto \mathit{friendOf}. \end{split}$$

Furthermore, the following production rules of the annotated grammar express the combination of paths in the social network to form bigger paths:

$$\begin{split} \textbf{q}[A,D] &\mapsto \textbf{q}[A,B] \ \textbf{q}[B,D], \\ \textbf{q}[A,E] &\mapsto \textbf{q}[A,C] \ \textbf{q}[C,E], \\ \textbf{q}[B,E] &\mapsto \textbf{q}[B,D] \ \textbf{q}[D,E]. \end{split} \qquad \begin{aligned} \textbf{q}[A,E] &\mapsto \textbf{q}[A,B] \ \textbf{q}[B,E], \\ \textbf{q}[A,E] &\mapsto \textbf{q}[A,D] \ \textbf{q}[D,E], \end{aligned}$$

To produce a path from Alice to Eve, we can use this annotated grammar:

```
\begin{split} & q[Alice, Eve] \\ \to^*_{\mathbf{P}_{\mathsf{G}}} \{Rewrite \ q[Alice, Eve] \mapsto q[Alice, Bob] \ q[Bob, Eve] \} \\ & q[Alice, Bob] \ q[Bob, Eve] \\ \to^*_{\mathbf{P}_{\mathsf{G}}} \{Rewrite \ q[Bob, Eve] \mapsto q[Bob, Dan] \ q[Dan, Eve] \} \\ & q[Alice, Bob] \ q[Bob, Dan] \ q[Dan, Eve] \\ \to^*_{\mathbf{P}_{\mathsf{G}}} \{Rewrite \ q[Alice, Bob] \mapsto \mathit{friendOf}, \dots \} \\ & \mathit{friendOf friendOf}. \end{split}
```

The node-information in each annotated non-terminal allows us to conclude that there is a path from Alice to Eve of length three, namely the path

#### Alice friendOf Bob friendOf Dan friendOf Eve.

As Example 3 shows, an annotated non-terminal in an annotated grammar describes the mapping between the trace that is derived by the non-terminal and the first and last node of a path having this trace.

**Proposition 1.** Let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar, let  $G = (Q, \Sigma, \delta)$  be a graph, let  $\mathbf{C}_G = (\mathbf{N}_G, \Sigma, \mathbf{P}_G)$  be the annotated grammar over  $(\mathbf{C}, G)$ , let  $m\pi n$  be a path in G, and let  $\mathbf{a} \in \mathbf{N}$  be a non-terminal. We have  $trace(\pi) \in \mathcal{L}(\mathbf{C}; \mathbf{a})$  if and only if we can derive  $\pi$  from  $\mathbf{a}[m, n] \in \mathbf{N}_G$ .

As annotated grammars are context-free grammars, one can use existing context-free enumeration techniques [11, 15, 12, 25] to produce some of the paths represented by the annotated grammar. The efficiency of these techniques depend on the size of the annotated grammar. We observe the following worst-case upper bounds:

**Lemma 2.** Let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar, let  $G = (Q, \Sigma, \delta)$  be a graph, and let  $\mathbf{C}_G = (\mathbf{N}_G, \Sigma, \mathbf{P}_G)$  be the annotated grammar over  $(\mathbf{C}, G)$ . We have  $|\mathbf{N}_G| \leq |\mathbf{N}||Q|^2$  and  $|\mathbf{P}_G| \leq |\mathbf{P}||Q|^3 + \min(|\mathbf{N}|, |\mathbf{P}|)|\delta|$ .

We propose annotated grammars to represent the query result of a query using the all-path query semantics. Hence, we also need to show how to construct such an annotated grammar, which we do next.

**Theorem 1.** Let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar, let  $\mathsf{G} = (\mathsf{Q}, \Sigma, \delta)$  be a graph. We can construct the annotated grammar over  $(\mathbf{C}, \mathsf{G})$  in  $\mathcal{O}(|\mathbf{N}||\delta| + (|\mathbf{N}||\mathsf{Q}|)^3)$ .

*Proof.* We use the context-free recognizer for graphs of Hellings [21] to construct the set  $\mathbf{N}_{\mathsf{G}} = \{\mathbf{a}[m,n] \mid \mathcal{L}(\mathbf{C};\mathbf{a}) \cap \mathcal{L}(\mathsf{G};m,n) \neq \emptyset\}$  in  $\mathcal{O}(|\mathbf{N}||\delta| + (|\mathbf{N}||\mathbf{Q}|)^3)$ . Using

 $N_G$ , we can construct

$$\begin{split} \mathbf{P}_{\mathsf{G}} &= \{ (\mathtt{a}[m,n] \mapsto \sigma) \mid (\mathtt{a} \mapsto \sigma) \in \mathbf{P} \land (m,\sigma,n) \in \delta \} \cup \\ & \{ (\mathtt{a}[m,n] \mapsto \mathtt{b}[m,o] \ \mathtt{c}[o,n]) \mid (\mathtt{a} \mapsto \mathtt{b} \ \mathtt{c}) \in \mathbf{P} \land \mathtt{a}[m,n], \mathtt{b}[m,o], \mathtt{c}[o,n] \in \mathbf{N}_{\mathsf{G}} \}. \end{split}$$

For the construction of  $\mathbf{P}_{\mathsf{G}}$ , we represent  $\mathbf{N}_{\mathsf{G}}$  by a 3-dimensional boolean matrix and the set of production rules  $\mathbf{P}$  by two look-up structures (one for rules of the form  $\mathtt{a} \mapsto \sigma$  and one for rules of the form  $\mathtt{a} \mapsto \mathtt{b} \mathtt{c}$ ). These structures guarantee constant-time lookups for all the parts used in the definition of  $\mathbf{P}_{\mathsf{G}}$ . By the worst-case upper bounds on  $|\mathbf{P}_{\mathsf{G}}|$ , we conclude that we can construct  $\mathbf{P}_{\mathsf{G}}$  in  $\mathcal{O}(|\mathbf{N}||\delta| + (|\mathbf{N}||\mathbf{Q}|)^3)$ .

Observe that Theorem 1 also proves Lemma 1, and it does so by a direct context-free grammar construction. Usually, Lemma 1 is proven indirectly by using a pushdown automaton-based construction [23]. Our direct approach to proving Lemma 1 is essential; indeed, it is the direct construction of a context-free grammar in our proof that allows us to guarantee the structural properties in the result that we need for the derivation of paths.

# 5 Answering queries using single-path query semantics

Although querying for all paths can be useful in certain cases, it is often sufficient if the query answer contains a single such path: a single path is much easier to comprehend by end users and can already reveal the information end users are looking for. As the length of these paths is not necessarily upper bounded, a logical choice would be to choose a path that is as short as possible. Preferring such a path of minimal length over longer paths can provide additional practical value, as the following example illustrates.

Example 4. Recall Example 3. If Alice used the query q to find out how she can get in contact with Eve via friends, friends of friends, and so on, then she probably wants to contact Eve without contacting to many other people. Hence, the provided answer via Bob and Dan is not optimal. The path Alice friendOf Craig friendOf Eve is shorter, and using this path Alice can get in contact with Eva by only contacting Craig.

Towards answering context-free path queries with a single path of minimum length, we proceed in two steps. In Section 5.1, we develop an approach to derive a string of minimum length from a context-free grammar. In Section 5.2, we apply this approach to derive strings of minimum length to annotated grammars, hence showing how to answer context-free path queries with a path of minimum length.

#### 5.1 Construction of strings of minimum length

The goal of this section is to provide an approach to finding a string of minimum length in a language defined by a context-free grammar.

**Definition 4.** If  $\mathcal{L}$  is a language, then the *min-length* of the language  $\mathcal{L}$ , denoted by  $\min \|\mathcal{L}\|$ , is defined by  $\min \|\mathcal{L}\| = \min\{\|S\| \mid S \in \mathcal{L}\}.$ 

Mclean et al. [26] showed that a string of minimum length in a context-free language can be computed. Their results do, however, not give a practical algorithm or complexity results for deriving strings of minimum length. Towards such a derivation algorithm, we introduce derivations using deterministic non-recursive production rules:

**Definition 5.** Let **P** be a set of production rules. We define heads(**P**) = {a |  $(a \mapsto S) \in \mathbf{P}$ }. We define the set of non-terminals derivable from a using the production rules in **P**, denoted by  $\langle \mathbf{a} \rangle_{\mathbf{P}}$ , as  $\langle \mathbf{a} \rangle_{\mathbf{P}} = \{ \mathbf{b} \mid \mathbf{b} \in \mathbf{N} \land \exists S_1 \exists S_2 \ \mathbf{a} \rightarrow_{\mathbf{P}}^+ S_1 \cdot \mathbf{b} \cdot S_2 \}$ . A set of production rules **P** is non-recursive if, for every  $\mathbf{a} \in \text{heads}(\mathbf{P})$ , we have  $\mathbf{a} \notin \langle \mathbf{a} \rangle_{\mathbf{P}}$ . A set of production rules **P** is deterministic non-recursive if it is non-recursive; if, for every  $\mathbf{a} \in \text{heads}(\mathbf{P})$ , there exists exactly one  $(\mathbf{a} \mapsto S) \in \mathbf{P}$ ; and if  $\mathbf{a} \in \text{heads}(\mathbf{P})$  implies that there exists a string  $S \in \Sigma^*$  such that  $\mathbf{a} \rightarrow_{\mathbf{P}}^* S$ .

Observe that a deterministic non-recursive set  $\mathbf{P}$  does not provide choices in how one rewrites a non-terminal  $\mathbf{a}$  into a string. As a consequence,  $\mathbf{P}$  rewrites each  $\mathbf{a} \in \mathsf{heads}(\mathbf{P})$  into a unique string  $S \in \Sigma^*$ . In this setting, we define  $\mathsf{string}(\mathbf{a}; \mathbf{P}) = S$ .

Example 5. Recall Example 3. The following set of production rules in the annotated grammar is deterministic non-recursive:

$q[A, B] \mapsto friendOf$ ,	$q[A, C] \mapsto friendOf$ ,
$\mathtt{q}[B,D] \mapsto \mathit{friendOf},$	$q[C, E] \mapsto \mathit{friendOf},$
$q[D, E] \mapsto \mathit{friendOf},$	$\mathtt{q}[A,D] \mapsto \mathtt{q}[A,B] \ \mathtt{q}[B,D],$
$q[A, E] \mapsto q[A, B] q[B, E],$	$q[B, E] \mapsto q[B, D] \ q[D, E].$

**Lemma 3.** Let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar, and let  $\mathbf{a} \in \mathbf{N}$  be a non-terminal with  $\mathcal{L}(\mathbf{C}; \mathbf{a}) \neq \emptyset$ . There exists a deterministic non-recursive set  $\mathbf{P}' \subseteq \mathbf{P}$  such that  $\|\mathsf{string}(\mathbf{a}; \mathbf{P}')\| = \min \|\mathcal{L}(\mathbf{C}; \mathbf{a})\|$ .

Proof (sketch). Let S be a string with  $\mathbf{a} \to_{\mathbf{P}}^* S$  and  $||S|| = \min ||\mathcal{L}(\mathbf{C}; \mathbf{a})||$ . Consider the derivation of S. If the derivation  $\mathbf{a} \to_{\mathbf{P}}^* S$  has a sequence of rewrite steps  $\mathbf{b} \to_{\mathbf{P}}^+ S_1 \cdot \mathbf{b} \cdot S_2$ , then, due to S having minimum length, we must have  $S_1 = S_2 = \varepsilon$ . Hence, we can remove all the rewrite steps involved in  $\mathbf{b} \to_{\mathbf{P}}^+ S_1 \cdot \mathbf{b} \cdot S_2$  and use the rewrite steps used to rewrite  $\mathbf{b}$  in  $S_1 \cdot \mathbf{b} \cdot S_2$  instead. If the derivation  $\mathbf{a} \to_{\mathbf{P}}^* S$  uses distinct production rules  $\mathbf{c} \mapsto S_1$  and  $\mathbf{c} \mapsto S_2$ , then, due to S having minimum length, both  $S_1$  and  $S_2$  are rewritten into equal length strings in  $\Sigma^*$ . Hence, we can choose one of the two production rules and use it for both rewrites of  $\mathbf{c}$ , the resulting string will have the same length as

**Corollary 1.** Let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar. There exists a deterministic non-recursive set  $\mathbf{P}' \subseteq \mathbf{P}$  such that for every non-terminal  $\mathbf{a} \in \mathbf{N}$  with  $\mathcal{L}(\mathbf{C}; \mathbf{a}) \neq \emptyset$ , we have  $\|\mathbf{string}(\mathbf{a}; \mathbf{P}')\| = \min \|\mathcal{L}(\mathbf{C}; \mathbf{a})\|$ .

We say that a deterministic non-recursive set satisfying the conditions of Corollary 1 is *minimizing*. Corollary 1 does not imply that each deterministic non-recursive set always produces strings of minimum length. With an example, we show that this is not the case:

Example 6. Recall Example 5. The provided set of deterministic non-recursive production rules  $\mathbf{P}'$  is not minimizing. We have  $\|\mathsf{string}(\mathsf{q}[A,E];\mathbf{P}')\| = 3$ , while a

shorter string of length two exists (via Craig). By replacing the production rule for q[Alice, Eve] by  $q[Alice, Eve] \mapsto q[Alice, Craig]$  q[Craig, Eve], the resulting set of production rules is minimizing.

Given a minimizing set of production rules  $\mathbf{P}'$ , it is straightforward to produce a string of minimum length for each non-terminal  $\mathbf{a} \in \mathsf{heads}(\mathbf{P}')$  that has such a string. The worst-case complexity of producing these strings is dominated by the length of the produced strings. By using the restrictions put on deterministic non-recursive sets, we can provide the following worst-case upper bounds on the length of strings of minimal length:

**Proposition 2.** Let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar and let  $N = \{a \mid a \in \mathbf{N} \land \mathcal{L}(\mathbf{C}; a) \neq \emptyset\}$  be the set of non-terminals that define a non-empty language. We have  $\max_{a \in N} (\min \|\mathcal{L}(\mathbf{C}; a)\|) \leq 2^{|\mathbf{N}|-1}$  and  $\sum_{a \in N} (\min \|\mathcal{L}(\mathbf{C}; a)\|) \leq 2^{|\mathbf{N}|} - 1$ .

Proof (sketch). Let  $|\mathbf{N}| = i$  and let  $\sigma \in \Sigma$ . We only have to consider the case where  $\mathbf{P}$  is a deterministic non-recursive set. Hence, we can order the non-terminals such that  $\mathbf{N} = \{\mathbf{a}_0, \dots, \mathbf{a}_{i-1}\}$  and we use the production rules  $\mathbf{a}_0 \mapsto \sigma$  and  $\mathbf{a}_j \mapsto \mathbf{a}_{j-1}$  and  $\mathbf{a}_{j-1}$ , for all  $1 \leq j \leq i-1$ .

As there exists a straightforward procedure to efficiently construct strings of minimum length from a minimizing set of production rules, we only need a procedure to construct such a set of production rules for a given a context-free grammar. Algorithm 1 provides such a procedure.

### **Algorithm 1** Construct a minimizing set of production rules for $C = (N, \Sigma, P)$

```
1: \mathbf{P}', cost := empty mapping, empty mapping
 2: new is a min-priority queue
 3: for all (a \mapsto \sigma) \in \mathbf{P} do
       if a \notin cost then
 4:
          cost[a], \mathbf{P}'[a] := 1, (a \mapsto \sigma)
 5:
          add a to new with priority 1
 7:
    while new \neq \emptyset do
       take a with minimum priority in new, and remove it from new
 8:
       for all (c \mapsto a b) \in P with b \in cost do
 9:
          PRODUCE(c \mapsto a b)
10:
       for all (c \mapsto b \ a) \in P with b \in cost \ do
11:
          PRODUCE(c \mapsto b \ a)
13: return \{\mathbf{P}'[\mathbf{a}] \mid \mathbf{a} \in \mathbf{P}'\}
Procedure PRODUCE(d \mapsto e f):
14: if d \notin cost then
        cost[d], \mathbf{P}'[d] := cost[e] + cost[f], (d \mapsto e f)
15:
       add d to new with priority cost[e] + cost[f]
    else if cost[d] > cost[e] + cost[f] then
17:
       cost[d], \mathbf{P}'[d] := cost[e] + cost[f], (d \mapsto e f)
18:
       lower priority of d in new to cost[e] + cost[f]
19:
```

**Proposition 3.** Let  $C = (N, \Sigma, P)$  be a context-free grammar. Algorithm 1 applied on C produces a minimizing set of production rules for C.

*Proof* (sketch). The main while-loop maintains the following invariants:

- 1. If  $a \in \mathbf{P}'$  and  $\mathbf{P}'[a] = (a \mapsto b \ c)$ , then  $b, c \in \mathbf{P}'$ .
- 2. If  $a \in \mathbf{P}'$  and  $\mathbf{P}'[a] = (a \mapsto b \ c)$ , then  $cost[a] \ge cost[b] + cost[c]$ , cost[a] > cost[b], and cost[a] > cost[c].
- 3. If  $a \in \mathbf{P}'$ , then  $\|\operatorname{string}(a; S)\| \leq \operatorname{cost}[a]$ .
- 4. Let m be the priority of the last element removed from new. No new element is inserted in new with priority less than or equal to m.
- 5. Let m be the priority of the last element removed from new. For all  $\mathbf{a} \in \mathbf{N}$  with  $\min \|\mathcal{L}(\mathbf{C}; \mathbf{a})\| \le m$ , we have  $cost[\mathbf{a}] = \min \|\mathcal{L}(\mathbf{C}; \mathbf{a})\|$ .

As each non-terminal is added to *new* at most once, Algorithm 1 terminates. At termination, Invariants 1–5 guarantee that the resulting set of production rules  $\{\mathbf{P}'[\mathbf{a}] \mid \mathbf{a} \in \mathbf{P}'\}$  is minimizing.

We observe that we cannot straightforwardly generalize Algorithm 1 to the setting that includes production rules of the form  $\mathbf{a} \mapsto \varepsilon$ : Invariants 2, 4, and 5 of the proof of Proposition 3 no longer hold (as they all require a strict ordering of the *cost* of non-terminals). This issue can be resolved by maintaining a timestamp on each non-terminal (such that a non-terminal has timestamp i if it was the i-th change to  $\mathbf{P}'$ ), and change the relevant invariants to not require a strict ordering on the *cost* of non-terminals, but a strict ordering on the pairs (*cost*, timestamp) of non-terminals.

**Theorem 2.** Let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar. Algorithm 1 constructs a minimizing set of production rules for  $\mathbf{C}$  in  $\mathcal{O}(|\mathbf{N}|(|\mathbf{N}|\log(|\mathbf{N}|)+|\mathbf{P}|))$ .

Proof. We represent costs as an array holding  $|\mathbf{N}|$  integers. The costs used in cost and new are integers in the range  $1, \ldots, 2^{|\mathbf{N}|-1}$ . We can represent each of these integers using  $\log(2^{|\mathbf{N}|}) = |\mathbf{N}|$  bits. The initialization steps perform  $\mathcal{O}(|\mathbf{P}|)$  steps. The while-loop will, in the worst case, visit every non-terminal once. For each of these non-terminals, one insertion into and one removal from the priority queue new is performed. The inner for-loops will visit every production rule twice, causing at most  $2|\mathbf{P}|$  decrease key operations on priority queue new. When using a Fibonacci heap for a priority queue holding at most e elements, each insert and removal costs  $\mathcal{O}(\log(e))$  and each decrease key operation costs an amortized  $\mathcal{O}(1)$  heap operations [10, 16]. Hence, a total of  $\mathcal{O}(|\mathbf{N}|\log(|\mathbf{N}|) + |\mathbf{P}|)$  heap operations are performed. Taking the size of the integers representing priorities into account, the heap operations cost  $\mathcal{O}(|\mathbf{N}|(|\mathbf{N}|\log(|\mathbf{N}|) + |\mathbf{P}|))$ .  $\square$ 

Using Theorem 2 and Proposition 2, we conclude the following:

Corollary 2. Let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar, let  $N = \{\mathbf{a} \mid \mathbf{a} \in \mathbf{N} \land \mathcal{L}(\mathbf{C}; \mathbf{a}) \neq \emptyset\}$  be the set of non-terminals that define a non-empty language, and let  $L = \sum_{\mathbf{a} \in N} \min \|\mathcal{L}(\mathbf{C}, \mathbf{a})\|$  be the combined length of a string of minimum length for each non-terminal in N. We can construct strings of minimum length for all non-terminals in N in  $\mathcal{O}(|\mathbf{N}|(|\mathbf{N}|\log(|\mathbf{N}|) + |\mathbf{P}|) + L) = \mathcal{O}(|\mathbf{N}|(|\mathbf{N}|\log(|\mathbf{N}|) + |\mathbf{P}|) + 2^{|\mathbf{N}|})$ .

### 5.2 Construction of paths of minimum length

We can already answer queries with single paths of minimum length by first constructing an annotated grammar and then applying Algorithm 1. This approach has high overhead due to the explicit construction and storing of the annotated grammar. To reduce this overhead, we adapt Algorithm 1 to the setting of query evaluation using the single-path query semantics. The resulting algorithm, Algorithm 2, operates on a normal context-free grammar and a graph, and derives the necessary details of the annotated grammar in place. If necessary, Algorithm 2 can use straightforward bookkeeping to also construct  $N_G$  and  $P_G$ , this without increasing the asymptotic complexity of the algorithm.

**Algorithm 2** Construct a minimizing set of production rules for the annotated grammar over  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  and  $\mathsf{G} = (\mathsf{Q}, \Sigma, \delta)$ 

```
1: \mathbf{P}', cost := empty mapping, empty mapping
 2: new is a min-priority queue
 3: for all (a \mapsto \sigma) \in \mathbf{P} and (m, \sigma, n) \in \delta do
       if a[m,n] \notin cost then
 4:
           cost[\mathtt{a}[m,n]], \mathbf{P}'[\mathtt{a}[m,n]] := 1, (\mathtt{a}[m,n] \mapsto \sigma)
 5:
          add a[m, n] to new with priority 1
 6:
    while new \neq \emptyset do
 7:
       take a[m, n] with minimum priority in new, and remove it from new
 8:
 9:
       for all (c \mapsto a b) \in P with b[n, o] \in cost do
10:
          PRODUCE(c[m, o] \mapsto a[m, n] b[n, o])
       for all (c \mapsto b \ a) \in \mathbf{P} with b[o, m] \in cost \ do
11:
          PRODUCE(c[o, n] \mapsto b[o, m] \ a[m, n])
12:
13: return \{ \mathbf{P}'[\mathsf{a}[m,n]] \mid \mathsf{a}[m,n] \in \mathbf{P}' \}
Procedure PRODUCE(d[u, w] \mapsto e[u, v] f[v, w]):
14: if d[u, w] \notin cost then
                                       := cost[e[u,v]] + cost[f[v,w]], (d[u,w])
       cost[d[u, w]], \mathbf{P'}[d[u, w]]
       e[u,v] f[v,w]
       add d[u, w] to new with priority cost[e[u, v]] + cost[f[v, w]]
16:
17: else if cost[d[u, w]] > cost[e[u, v]] + cost[f[v, w]] then
       cost[\mathtt{d}[u,w]], \mathbf{P}'[\mathtt{d}[u,w]] := cost[\mathtt{e}[u,v]] + cost[\mathtt{f}[v,w]], (\mathtt{d}[u,w])
       e[u,v] f[v,w]
       lower priority of d[u, w] in new to cost[e[u, v]] + cost[f[v, w]]
19:
```

Before we fully analyze Algorithm 2, we use Lemma 2 and Proposition 2 to conclude the following naive worst-case upper bound on the length of paths of minimal length:

Corollary 3. Let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar with  $\mathbf{a} \in \mathbf{N}$  and let  $\mathsf{G} = (\mathsf{Q}, \Sigma, \delta)$  be a graph with  $m, n \in \mathsf{Q}$ , such that  $\mathcal{L} = \mathcal{L}(\mathbf{C}; \mathbf{a}) \cap \mathcal{L}(\mathsf{G}; m, n) \neq \emptyset$ . We have  $\min \|\mathcal{L}\| \leq 2^{|\mathbf{N}||\mathbf{Q}|^2 - 1}$ .

Using Corollary 3, we conclude the following:

**Proposition 4.** Let  $C = (N, \Sigma, P)$  be a context-free grammar and let  $G = (Q, \Sigma, \delta)$  be a graph. Algorithm 2 constructs a minimizing set of production rules

for the annotated grammar  $\mathbf{C}_{\mathsf{G}} = (\mathbf{N}_{\mathsf{G}}, \Sigma, \mathbf{P}_{\mathsf{G}})$  over  $(\mathbf{C}, \mathsf{G})$  in  $\mathcal{O}(|\mathbf{N}_{\mathsf{G}}|(|\mathbf{N}_{\mathsf{G}}|\log(|\mathbf{N}_{\mathsf{G}}|) + |\mathbf{P}_{\mathsf{G}}|))$ , and, hence, in  $\mathcal{O}(|\mathbf{N}||\mathsf{Q}|^2((|\mathbf{N}||\mathsf{Q}|^2)\log(|\mathbf{N}||\mathsf{Q}|^2) + |\mathbf{P}||\mathsf{Q}|^3 + \min(|\mathbf{N}|, |\mathbf{P}|)|\delta|)$ .

**Corollary 4.** Let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar with  $\mathbf{a} \in \mathbf{N}$  and let  $\mathsf{G} = (\mathsf{Q}, \Sigma, \delta)$  be a graph with  $m, n \in \mathsf{Q}$ , such that  $\mathcal{L} = \mathcal{L}(\mathbf{C}; \mathbf{a}) \cap \mathcal{L}(\mathsf{G}; m, n) \neq \emptyset$ . We can construct a path  $m\pi n$  such that  $\mathsf{trace}(\pi) \in \mathcal{L}$  and  $\|\mathsf{trace}(\pi)\| = \min \|\mathcal{L}\|$  in

$$\begin{split} &\mathcal{O}(|\mathbf{N}||\mathbf{Q}|^2((|\mathbf{N}||\mathbf{Q}|^2)\log(|\mathbf{N}||\mathbf{Q}|^2)+|\mathbf{P}||\mathbf{Q}|^3+\min(|\mathbf{N}|,|\mathbf{P}|)|\delta|)+\|\mathit{trace}(\pi)\|) = \\ &\mathcal{O}(|\mathbf{N}||\mathbf{Q}|^2((|\mathbf{N}||\mathbf{Q}|^2)\log(|\mathbf{N}||\mathbf{Q}|^2)+|\mathbf{P}||\mathbf{Q}|^3+\min(|\mathbf{N}|,|\mathbf{P}|)|\delta|)+2^{|\mathbf{N}||\mathbf{Q}|^2-1}). \end{split}$$

Observe that the upper bound of Corollary 3 is very loose: Proposition 2 depends on production rules of the form  $\mathtt{a} \to \mathtt{b}\,\mathtt{b}$ , whereas, in general, annotated grammars only allow for such production rules in very restricted cases. Hence, we look at ways to improve the worst-case upper bound observed by Corollary 3. As an initial step, we consider languages defined over singleton alphabets:

**Proposition 5.** Let  $\Sigma$  be an alphabet with  $|\Sigma| = 1$ , let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar with  $\mathbf{a} \in \mathbf{N}$ , and let  $\mathsf{G} = (\mathsf{Q}, \Sigma, \delta)$  be a graph with  $m, n \in \mathsf{Q}$ , such that  $\mathcal{L} = \mathcal{L}(\mathbf{C}; \mathbf{a}) \cap \mathcal{L}(\mathsf{G}; m, n) \neq \emptyset$ . In the worst case, we have  $|\mathsf{Q}|2^{|\mathbf{N}|-1} \leq \min \|\mathcal{L}\| \leq |\mathsf{Q}|(2^{2|\mathbf{N}|-1}+1)$ .

Proof. First, we prove the lower bound. Let  $\Sigma = \{\sigma\}$ , let  $Q = \{n_0, \dots, n_{|Q|-1}\}$ , let  $\delta = \{(n_i, \sigma, n_{i+1 \mod |Q|}) \mid 0 \le i \le |Q|-1\}$ , let  $\mathbf{N} = \{\mathbf{a}_0, \dots, \mathbf{a}_{|\mathbf{N}|-1}\}$ , and let  $\mathbf{P} = \{\mathbf{a}_0 \mapsto \sigma, \mathbf{a}_0 \mapsto \mathbf{a}_0 \ \mathbf{a}_0\} \cup \{\mathbf{a}_i \mapsto \mathbf{a}_{i-1} \ \mathbf{a}_{i-1} \ | \ 1 \le i \le |\mathbf{N}|-1\}$ . With these definitions we have  $\mathcal{L}(\mathsf{G}; n_i, n_i) = \{\sigma^{k|Q|} \ | \ 0 \le k\}$ , for every  $1 \le m$ 

With these definitions we have  $\mathcal{L}(\mathsf{G}; n_i, n_i) = \{\sigma^{k|\mathsf{Q}|} \mid 0 \leq k\}$ , for every  $1 \leq i \leq |\mathsf{Q}|$ , and we have  $\mathcal{L}(\mathsf{C}; \mathsf{a}_0) = \{\sigma^k \mid 1 \leq k\}$ . Hence, the string S' of minimum length such that  $\mathsf{a}_0[n_i, n_i] \to_{\mathbf{P}}^* S'$  is  $S' = \sigma^{|\mathsf{Q}|}$ . Each  $\mathsf{a}_j$ ,  $1 \leq j \leq |\mathsf{N}| - 1$ , will be rewritten in a string of exactly  $2^j$   $\mathsf{a}_0$  non-terminals. Hence, the string  $S'_j$  of minimum length such that  $\mathsf{a}_j[n_i, n_i] \to_{\mathbf{P}}^* S'_j$  is  $S'_j = \sigma^{|\mathsf{Q}|2^j}$ , and we conclude  $||S'_{|\mathsf{N}|-1}|| = |\mathsf{Q}|2^{|\mathsf{N}|-1}$ .

The upper bound is proven using a result of Pighizzini et al. [27]: for each context-free grammar  $\mathbf{C}$  with y non-terminals, there exists a finite automaton  $\mathsf{G}' = (\mathsf{Q}', \Sigma, \delta')$  with initial state m, final states F, and with  $|\mathsf{Q}'| \leq 2^{2|\mathbf{N}|-1} + 1$  such that  $(\bigcup_{n \in F} \mathcal{L}(\mathsf{G}; m, n)) = \mathcal{L}(\mathbf{C}; \mathbf{a})$ . We use  $\mathsf{G}'$  to represent  $\mathbf{C}$  and we apply the well-known product construction for the intersection of finite automata on  $\mathsf{G}'$  and  $\mathsf{G}$ . The resulting finite automaton has  $|\mathsf{Q}||\mathsf{Q}'| = |\mathsf{Q}|(2^{2|\mathbf{N}|-1} + 1)$  states, proving the upper bound.

Due to Proposition 5, we can conclude that in the case of unlabeled graphs, the complexity of query evaluation with the single-path query semantics is polynomial in terms of the graph size and exponential in terms of the query size. Observe, however, that the exponential complexity in terms of the query size follows straightforward from the succinctness of context-free grammars (as compared to regular expressions and finite automata).

In the labeled case, we can still use Proposition 5 to get worst-case lower bounds on  $\min \|\mathcal{L}(\mathbf{C}; \mathbf{a}) \cap \mathcal{L}(\mathsf{G}; m, n)\|$ . The worst-case upper bound provided by Proposition 5 can, however, not be generalized to arbitrary alphabets, which we show next.

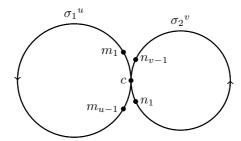


Figure 2: The double-cyclic graph: two cycles, one having u edges labeled with  $\sigma_1$ , and one having v edges labeled with  $\sigma_2$ . The two cycles are connected via a shared node c.

**Proposition 6.** Let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar with  $\mathbf{a} \in \mathbf{N}$ , and let  $\mathsf{G} = (\mathsf{Q}, \Sigma, \delta)$  be a graph with  $m, n \in \mathsf{Q}$ , such that  $\mathcal{L} = \mathcal{L}(\mathbf{C}; \mathbf{a}) \cap \mathcal{L}(\mathsf{G}; m, n) \neq \emptyset$ . In the worst case, we have  $|\mathsf{Q}|^2 2^{|\mathbf{N}|} \leq 64 \min \|\mathcal{L}\|$ .

*Proof.* Choose the well-known context-free language  $\mathcal{L} = \{\sigma_1^k \sigma_2^k \mid 1 \leq k\}$ . The context-free grammar  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  with  $\mathbf{N} = \{\mathbf{a}, \mathbf{a}', \mathbf{a}_1, \mathbf{a}_2, \mathbf{b}_1, \dots, \mathbf{b}_{|\mathbf{N}|-4}\}$  and

$$\mathbf{P} = \{ \mathtt{a} \mapsto \mathtt{a}_1 \ \mathtt{a}_2, \mathtt{a} \mapsto \mathtt{a}_1 \ \mathtt{a}', \mathtt{a}' \mapsto \mathtt{a} \ \mathtt{a}_2, \mathtt{a}_1 \mapsto \sigma_1, \mathtt{a}_2 \mapsto \sigma_2 \} \cup \\ \{ \mathtt{b}_1 \mapsto \mathtt{a} \ \mathtt{a} \} \cup \{ \mathtt{b}_j \mapsto \mathtt{b}_{j-1} \ \mathtt{b}_{j-1} \ | \ 1 < j \leq |\mathbf{N}| - 4 \}$$

has  $\mathcal{L}(\mathbf{C}; \mathbf{a}) = \mathcal{L}$ . Choose a k with  $1 \leq k$  and choose |Q| = u + v - 1 with  $u = 2^k + 1$  and v = u - 1. Let  $\mathsf{G} = (\mathsf{Q}, \Sigma, \delta)$  with  $\mathsf{Q} = \{c, m_1, \ldots, m_{u-1}, n_1, \ldots n_{v-1}\}$  and

$$\delta = \{(c, \sigma_1, m_1), (m_{u-1}, \sigma_1, c)\} \cup \{(m_i, \sigma_1, m_{i+1}) \mid 1 \le i < u - 1\} \cup \{(c, \sigma_2, n_1), (n_{v-1}, \sigma_2, c)\} \cup \{(n_i, \sigma_2, n_{i+1}) \mid 1 \le i < v - 1\}.$$

The resulting graph is visualized in Figure 2.

Let  $c\pi c$  be a path in  $\mathsf{G}$  with  $\operatorname{trace}(\pi) \in \mathcal{L}(\mathbf{C}; \mathsf{a})$ . Due to the definition of  $\mathsf{a}$ , we must have  $\operatorname{trace}(\pi) = S_1 \cdot S_2$  with  $S_1 = \sigma_1{}^x$  and  $S_2 = \sigma_2{}^x$ , for  $1 \leq x$ . Due to the structure of the graph,  $S_1$  must be the trace of a path  $c\pi_1c$  and  $S_2$  must be the trace of a path  $c\pi_2c$  in graph  $\mathsf{G}$ . From these constraints, we conclude  $\mathcal{L}(\mathsf{C}; \mathsf{a}) \cap \mathcal{L}(\mathsf{G}; c, c) = \{\sigma_1{}^{k \operatorname{lcm}(u,v)} \sigma_2{}^{k \operatorname{lcm}(u,v)} \mid 1 \leq k\}$ . Observe that  $u = 2^k$  and  $v = 2^k + 1$  are coprime, hence, we have  $\operatorname{lcm}(u,v) = uv$ .

Each  $b_j$ ,  $1 \le j \le |\mathbf{N}| - 4$ , will be rewritten in a string of exactly  $2^j$  a non-terminals. As only node c has outgoing edges labeled with both  $\sigma_1$  and  $\sigma_2$ , each  $b_j[c,c]$  will be rewritten in a string of exactly  $2^j$  a[c,c] non-terminals. Hence, we conclude  $\min \|\mathcal{L}(\mathbf{C}; b_j) \cap \mathcal{L}(\mathsf{G}; c,c)\| = uv2^j$ , and we conclude

$$\min \|\mathcal{L}(\mathbf{C}; \mathbf{b}_{|\mathbf{N}|-4}) \cap \mathcal{L}(\mathsf{G}; c, c)\| = uv2^{|\mathbf{N}|-4} > v^2 \frac{2^{|\mathbf{N}|}}{16} = \left(\frac{|\mathbf{Q}|}{2}\right)^2 \frac{2^{|\mathbf{N}|}}{16} = \frac{|\mathbf{Q}|^2 2^{|\mathbf{N}|}}{64}.$$

For labeled graphs, we do not yet have a better worst-case upper bound than the naive upper-bound provided by Corollary 3.

Open Problem 1. Let  $\mathbf{C} = (\mathbf{N}, \Sigma, \mathbf{P})$  be a context-free grammar with  $\mathbf{a} \in \mathbf{N}$ , and let  $\mathsf{G} = (\mathsf{Q}, \Sigma, \delta)$  be a graph with  $m, n \in \mathsf{Q}$ , such that  $\mathcal{L} = \mathcal{L}(\mathbf{C}; \mathbf{a}) \cap \mathcal{L}(\mathsf{G}; m, n) \neq 0$ 

 $\emptyset$ . What is the strict worst-case upper bound on min  $\|\mathcal{L}\|$ ? Or, equivalently, what is the strict worst-case upper bound on the length of a shortest string in the intersection of the language of a context-free grammar with x non-terminals and the language of a finite automaton with y states?

We conjecture that, as in the unlabeled case, the complexity of query evaluation with the single-path query semantics is polynomial in terms of the graph size and exponential in terms of the query size.

# 6 Experimental results

To provide insight in the practical behavior of path-based query evaluation, we have implemented algorithms for the evaluation of queries using the single-path query semantics.<sup>4</sup> We primarily focus on the running time of Algorithm 2, as the cost of producing the paths of interest heavily depends on whether one wants to produce a path for a particular node pair or for all node pairs. We perform three different tests:

1. We compare two context-free grammars that both evaluate to the positive transitive closure (under the relational query semantics):

$$\begin{aligned} \mathbf{q}_1 &\mapsto \mathbf{a} \ \mathbf{q}_1, & & \mathbf{q}_1 &\mapsto \sigma, \\ \mathbf{q}_2 &\mapsto \mathbf{q}_2 \ \mathbf{q}_2, & & & \mathbf{q}_2 &\mapsto \sigma. \end{aligned} \qquad \mathbf{a} \mapsto \sigma;$$

Observe that the context-free grammar  $q_1$  is linear and non-ambiguous, whereas the context-free grammar with non-terminal  $q_2$  is non-linear and highly ambiguous. We measure the running time of constructing minimizing sets of annotated production rules for these queries on the cyclic graphs of Proposition 5.

- 2. We compare the context-free grammar  $q_1$ , which produces dense result sets, with the language  $\mathcal{L} = \{\sigma\sigma\sigma\}$ , which produces sparse result sets. We measure the running time of constructing minimizing sets of annotated production rules for these queries on the cyclic graphs of Proposition 5.
- 3. We derive the longest path of minimum length that matches the following context-free grammar:

$$q \mapsto a \ q',$$
  $q' \mapsto q \ b,$   $q \mapsto a \ b,$ 

Similar context-free grammars are used in Example 1 and Proposition 6. We evaluate these queries on the double-cyclic graphs of Proposition 6, for which we know that the query q will produce paths with a high minimum length. We measure both the running time of constructing a minimizing set of annotated production rules for q on the double-cyclic graphs, and the running time for deriving the longest path of minimum length from the resulting minimizing set.

<sup>&</sup>lt;sup>4</sup>The algorithms are implemented in C++. Measurements where performed on a system with an Intel Core i5-4670 CPU, running at a maximum of 3.8GHz, and with 16GB of main memory. The source code will be made available under an open-source license.

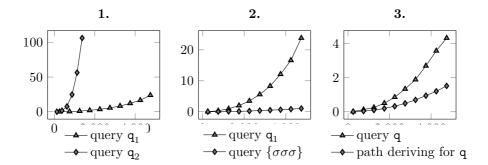


Figure 3: Results of test measurements on Algorithm 2 and on deriving paths of minimum length. The horizontal axis displays the size of the graph (|Q|), the vertical axis displays the running time (s), and the plot titles matches with the test descriptions in Section 6.

We remark that these tests illustrate extreme behavior: the queries apply to the entirety of the graph and all nodes and edges will participate in the outcome. This does not reflect all practical applications, where one can often expect that queries are much more selective.

The measurements for these three tests are summarized in Figure 3. On the one hand we see that Algorithm 2 can evaluate queries on large graphs, even if the resulting paths are large or if many paths are produced. For example, query q evaluated on double-cyclic graphs of 4750 nodes gives a total of  $11 \cdot 10^6$  paths, where the longest path consists of  $11 \cdot 10^6$  edges, this while the running time of Algorithm 2 is only 4.3s and the longest path is derived from the resulting minimizing set in only 1.5s. Hence, in this case, the cost for answering query q using the single-path query semantics is at most 5.8s.

On the other hand, we see that the performance of Algorithm 2 is heavily influenced by the ambiguity of the context-free grammar. We see that query  $\mathbf{q}_1$  evaluates magnitudes faster than query  $\mathbf{q}_2$ , even though the context-free languages underlying these queries are equivalent. The measurements on  $\mathbf{q}_1$  and  $\mathcal{L}$  show that evaluating  $\mathcal{L}$  is faster, which is unsurprising as  $\mathcal{L}$  is a much simpler query. Still, the difference in running time for these two queries is relatively small.

### 7 Conclusions and future work

To address the limits of the traditional query semantics for navigational query languages such as the context-free path queries, we proposed path-based query semantics. We studied two such path-based query semantics, namely the all-paths query semantics and the single-paths query semantics, and we provided a formal framework for evaluating queries on graphs using both path-based query semantics. Our initial results show that the path-based query semantics have added practical value and a small-scale experiment on an implementation of the main query evaluation algorithms show that query answering is feasible, even when query results grow very large.

In conclusion, we believe that our work opens the door for further study of

path-based query semantics. Besides the open problem already stated in this work—determining strict worst-case upper bounds on the size of the query result under the single-path query semantics—several other directions for future work have our interest.

- 1. Can we use more efficient algorithms for the problems outlined in this paper? Can we, for example, apply techniques used for context-free parsing [18, 30] or for Datalog query evaluation [2, 17]?
- 2. All algorithms outlined in this paper are bottom-up. Can we derive top-down algorithms or, in general, goal-oriented algorithms for answering queries for a given pair of nodes?
- 3. Our measurements showed that two different context-free grammars for the same context-free language can have huge differences in the running time for query evaluation. Can we optimize context-free grammars to guarantee better performance? Can we provide more efficient query evaluation for deterministic or for unambiguous context-free grammars?
- 4. Are there approximation algorithms for evaluating queries using the single-path query semantics that guarantee to produce paths whose length is close to the length of paths of minimum length, while having a much lower complexity? Our initial work on this topic shows that straightforward naive methods exist to efficiently produce a deterministic non-recursive set of production rules. Although such deterministic non-recursive sets of production rules guarantee a worst-case upper bound on path lengths, the length of the resulting paths is not necessary close to optimal.
- 5. To which extent can we adopt path-based query evaluation such that it exploits parallel hardware, distributed computing, and/or specialized acceleration hardware?
- 6. Can we generalize path-based query semantics to query languages that do not query based on path structures, but query based on patterns in graphs (such as Datalog and the navigational expressions [14]), and can we provide efficient query evaluation for such graph-based query semantics?

## References

- [1] Uri Alon. An Introduction to Systems Biology: Design Principles of Biological Circuits. Chapman and Hall/CRC, 2006.
- [2] Francois Bancilhon and Raghu Ramakrishnan. An amateur's introduction to recursive query processing strategies. In *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*, SIGMOD '86, pages 16–52. ACM, 1986.
- [3] Yehoshua Bar-Hillel, Micha A. Perles, and Eli Shamir. On formal properties of simple phrase structure grammars. Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung, 14:143–172, 1961.

- [4] Pablo Barceló. Querying graph databases. In Proceedings of the 32nd Symposium on Principles of Database Systems, PODS '13, pages 175–188. ACM, 2013.
- [5] Pablo Barceló, Leonid Libkin, Anthony W. Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems*, 37(4):31:1–31:46, 2012.
- [6] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández. Michael Kay, and Jonathan Robie and Jérôme Siméon. XML path language (XPath) 2.0 (second edition). http://www.w3.org/TR/2010/REC-xpath20-20101214/. W3C Recommendation 14 December 2010 (Link errors corrected 3 January 2011).
- [7] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, and John Cowan. Extensible markup language (XML) 1.1 (second edition). W3C recommendation, W3C, 2006. http://www.w3.org/TR/2006/REC-xm111-20060816.
- [8] James Clark and Steve DeRose. XML path language (XPath) version 1.0. http://www.w3.org/TR/1999/REC-xpath-19991116/. W3C Recommendation 16 November 1999.
- [9] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. The MIT Press, 1999.
- [10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2009.
- [11] Pál Dömösi. Unusual algorithms for lexicographical enumeration. *Acta Cybernetica*, 14(3):461–468, 2000.
- [12] YunMei Dong. Linear algorithm for lexicographic enumeration of CFG parse trees. Science in China Series F: Information Sciences, 52(7):1177–1202, 2009.
- [13] Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3):4–11, 1997.
- [14] George H. L. Fletcher, Marc Gyssens, Dirk Leinders, Jan Van den Bussche, Dirk Van Gucht, Stijn Vansummeren, and Yuqing Wu. Relative expressive power of navigational querying on graphs. In *Proceedings of the 14th International Conference on Database Theory*, pages 197–207, 2011.
- [15] Christophe Costa Florncio, Jonny Daenen, Jan Ramon, Jan Van den Bussche, and Dries Van Dyck. Naive infinite enumeration of context-free languages in incremental polynomial time. *Journal of Universal Computer Science*, 21(7):891–911, 2015.
- [16] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

- [17] Todd J. Green, Shan Shan Huang, Boon Thau Loo, and Wenchao Zhou. Datalog and recursive query processing. Foundations and Trends in Databases, 5:105–195, 2012.
- [18] Dick Grune and Ceriel J. H. Jacobs. Parsing Techniques. Monographs in Computer Science. Springer New York, 2008.
- [19] David Harel, Amir Pnueli, and Jonathan Stavi. Propositional dynamic logic of nonregular programs. *Journal of Computer and System Sciences*, 26(2):222–243, 1983.
- [20] Steven Harris and Andy Seaborne. SPARQL 1.1 query language. W3C recommendation, W3C, 2013. http://www.w3.org/TR/2013/REC-sparql11-query-20130321.
- [21] Jelle Hellings. Conjunctive context-free path queries. In *Proceedings of the* 17th International Conference on Database Theory (ICDT 2014), pages 119–130, 2014.
- [22] Piotr Hofman and Wim Martens. Separability by short subsequences and subwords. In 18th International Conference on Database Theory (ICDT 2015), volume 31 of Leibniz International Proceedings in Informatics (LIPIcs), pages 230–246. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
- [23] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation, 3th edition*. Pearson, 2007.
- [24] Martin Lange. Model checking propositional dynamic logic with all extras. Journal of Applied Logic, 4(1):39–49, 2006.
- [25] Erkki Mäkinen. On lexicographic enumeration of regular and context-free languages. *Acta Cybernetica*, 13(1):55–61, 1997.
- [26] Michael J. Mclean and Daniel B. Johnston. An algorithm for finding the shortest terminal strings which can be produced from non-terminals in context-free grammars. In *Combinatorial Mathematics III*, volume 452 of *Lecture Notes in Mathematics*, pages 180–196. Springer Berlin Heidelberg, 1975.
- [27] Giovanni Pighizzini, Jeffrey Shallit, and Ming wei Wang. Unary context-free grammars and pushdown automata, descriptional complexity and auxiliary space lower bounds. *Journal of Computer and System Sciences*, 65(2):393–414, 2002.
- [28] Guus Schreiber and Yves Raimond. RDF 1.1 primer. W3C working group note, W3C, 2014. http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624.
- [29] Petteri Sevon and Lauri Eronen. Subgraph queries by context-free grammars. *Journal of Integrative Bioinformatics*, 5(2), 2008.
- [30] Leslie G. Valiant. General context-free recognition in less than cubic time. Journal of Computer and System Sciences, 10(2):308–315, 1975.