Context-Free Path Querying by Matrix Multiplication

Rustam Azimov and Semyon Grigorev

Saint Petersburg State University
7/9 Universitetskaya nab.
St. Petersburg, 199034 Russia
rustam.azimov19021995@gmail.com
Semen.Grigorev@jetbrains.com

Abstract. Graph data models are widely used in many areas, for example, bioinformatics, graph databases. In these areas, it is often required to process queries for large graphs. One of the most common graph queries are navigational queries. The result of query evaluation is a set of implicit relations between nodes of the graph, i.e. paths in the graph. A natural way to specify these relations is by specifying paths using formal grammars over the alphabet of edge labels. An answer to a context-free path query in this approach is usually a set of triples (A, m, n) such that there is a path from node m to node n, whose labeling is derived from a non-terminal A of the given context-free grammar. This type of queries is evaluated using the *relational query semantics*. Another example of path query semantics is the *single-path query semantics* which requires to present a single path from node m to node n, whose labeling is derived from a non-terminal A for all triples (A, m, n) evaluated using the relational query semantics. There is a number of algorithms for query evaluation which use these semantics but all of them perform poorly on large graphs. One of the most common technique for efficient big data processing is GPGPU, but these algorithms do not allow to use this technique efficiently. In this paper we show how the context-free path query evaluation using these query semantics can be reduced to the calculation of the matrix transitive closure. Also, we propose an algorithm for context-free path query evaluation which uses relational query semantics and is based on matrix operations that make it possible to speed up computations by using GPGPU.

Keywords: Transitive closure, CFPQ, graph databases, context-free grammar, GPGPU, matrix multiplication

1 Introduction

Graph data models are widely used in many areas, for example, bioinformatics [3], graph databases [15]. In these areas, it is often required to process queries for large graphs. The most common among graph queries are navigational queries. The result of query evaluation is a set of implicit relations between nodes

of the graph, i.e. paths in the graph. A natural way to specify these relations is by specifying paths using formal grammars (regular expressions, context-free grammars) over the alphabet of edge labels. Context-free grammars are actively used in graphs queries because of the limited expressive power of regular expressions.

The result of context-free path query evaluation is usually a set of triples (A, m, n) such that there is a path from node m to node n, whose labeling is derived from a non-terminal A of the given context-free grammar. This type of query is evaluated using the *relational query semantics* [11]. Another example of path query semantics is the *single-path query semantics* [12] which requires to present a single path from node m to node n whose labeling is derived from a non-terminal A for all triples (A, m, n) evaluated using the relational query semantics. There is a number of algorithms for context-free path query evaluation using these semantics [9, 11, 29, 23].

Existing algorithms for context-free path query evaluation w.r.t. these semantics demonstrate poor performance when applied to big data. One of the most common technique for efficient big data processing is *GPGPU* (General-Purpose computing on Graphics Processing Units), but these algorithms do not allow to use this technique efficiently. The algorithms for context-free language recognition had a similar problem until Valiant [24] proposed a parsing algorithm which computes a recognition table by computing matrix transitive closure. Thus, the active use of matrix operations (such as matrix multiplication) in the process of a transitive closure computation makes it possible to efficiently apply GPGPU computing techniques [6].

We address the problem of creating an algorithm for context-free path query evaluation using the relational and the single-path query semantics which allows us to speed up computations with GPGPU by using the matrix operations.

The main contribution of this paper can be summarized as follows:

- We show how the context-free path query evaluation w.r.t. the relational and the single-path query semantics can be reduced to the calculation of matrix transitive closure.
- We introduce an algorithm for context-free path query evaluation w.r.t. the relational query semantics which is based on matrix operations that make it possible to speed up computations by means of GPGPU.
- We provide a formal proof of correctness of the proposed algorithm.
- We show the practical applicability of the proposed algorithm by running different implementations of our algorithm on real-world data.

2 Preliminaries

In this section, we introduce the basic notions used throughout the paper.

Let Σ be a finite set of edge labels. Define an *edge-labeled directed graph* as a tuple $D = (V, E)$ with a set of nodes V and a directed edge-relation $E \subseteq V \times \Sigma \times V$. For a path π in a graph D we denote the unique word obtained by concatenating the labels of the edges along the path π as $l(\pi)$. Also, we write $n\pi m$ to indicate that a path π starts at node $n \in V$ and ends at node $m \in V$.

Following Hellings [11], we deviate from the usual definition of a context-free grammar in *Chomsky Normal Form* [7] by not including a special starting non-terminal, which will be specified in the path queries to the graph. Since every context-free grammar can be transformed into an equivalent one in Chomsky Normal Form and checking that an empty string is in the language is trivial it is sufficient to consider only grammars of the following type. A *context-free grammar* is a triple $G = (N, \Sigma, P)$, where N is a finite set of non-terminals, Σ is a finite set of terminals, and P is a finite set of productions of the following forms:

- $A \to BC$, for $A, B, C \in N$,
- $A \to x$, for $A \in N$ and $x \in \Sigma$.

Note that we omit the rules of the form $A \to \varepsilon$, where ε denotes an empty string. This does not restrict the applicability of our algorithm because only the empty paths $m\pi m$ correspond to an empty string ε.

We use the conventional notation $A \xrightarrow{*} w$ to denote that a string $w \in \Sigma^*$ can be derived from a non-terminal A by some sequence of applications of the production rules from P. The *language* of a grammar $G = (N, \Sigma, P)$ with respect to a start non-terminal $S \in N$ is defined by

$$L(G_S) = \{w \in \Sigma^* \mid S \xrightarrow{*} w\}.$$

For a given graph $D = (V, E)$ and a context-free grammar $G = (N, \Sigma, P)$ we define *context-free relations* $R_A \subseteq V \times V$, for every $A \in N$, such that

$$R_A = \{(n, m) \mid \exists n\pi m \ (l(\pi) \in L(G_A))\}.$$

We define a binary operation $(\ \cdot\ )$ on arbitrary subsets N_1, N_2 of N with respect to a context-free grammar $G = (N, \Sigma, P)$ as

$$N_1 \cdot N_2 = \{A \mid \exists B \in N_1, \exists C \in N_2 \text{ such that } (A \to BC) \in P\}.$$

Using this binary operation as a multiplication of subsets of N and union of sets as an addition, we can define a *matrix multiplication*, $a \times b = c$, where a and b are matrices of a suitable size that have subsets of N as elements, as

$$c_{i,j} = \bigcup_{k=1}^{n} a_{i,k} \cdot b_{k,j}.$$

According to Valiant [24], we define the *transitive closure* of a square matrix a as $a^+ = a_+^{(1)} \cup a_+^{(2)} \cup \cdots$ where $a_+^{(1)} = a$ and

$$a_+^{(i)} = \bigcup_{j=1}^{i-1} a_+^{(j)} \times a_+^{(i-j)}, \ i \geq 2.$$

We enumerate the positions in the input string s of Valiant's algorithm from 0 to the length of s. Valiant proposes the algorithm for computing this transitive

closure only for upper triangular matrices, which is sufficient since for Valiant's algorithm the input is essentially a directed chain and for all possible paths $n \pi m$ in a directed chain $n < m$. In the context-free path querying input graphs can be arbitrary. For this reason, we introduce an algorithm for computing the transitive closure of an arbitrary square matrix.

For convenience of further reasoning, we introduce another definition of the transitive closure of an arbitrary square matrix a as $a^{cf} = a^{(1)} \cup a^{(2)} \cup \cdots$ where $a^{(1)} = a$ and

$$a^{(i)} = a^{(i-1)} \cup (a^{(i-1)} \times a^{(i-1)}),\ i \geq 2.$$

These two transitive closure definitions are equivalent (a formal proof can be found in the appendix A). Further in this paper we use the transitive closure a^{cf} instead of a^+ and algorithm for computing a^{cf} also computes Valiant's transitive closure a^+.

3 Related works

Problems in many areas can be reduced to one of the formal-languages-constrained path problems [4]. For example, various problems of static code analysis [5, 25] can be formulated in terms of the context-free language reachability [20] or in terms of the linear conjunctive language reachability [28].

One of the well-known problems in the area of graph database analysis is the language-constrained path querying. For example, the regular language constrained path querying [21, 8, 2, 16], and the context-free language constrained path querying.

There is a number of solutions [11, 23, 29] for context-free path query evaluation w.r.t. the relational query semantics, which employ such parsing algorithms as CYK [13, 27] or Earley [10]. Another examples of path query semantics are single-path and *all-path query semantics*. The all-path query semantics requires to present all possible paths from node m to node n whose labeling is derived from a non-terminal A for all triples (A, m, n) evaluated using the relational query semantics. Hellings [12] presented an algorithms for the context-free path query evaluation using the single-path and the all-path query semantics. If a context-free path query w.r.t. the all-path query semantics is evaluated on cyclic graphs, then the query result can be an infinite set of paths. For this reason, in [12], annotated grammars are proposed as a possible solution.

In [9], the algorithm for context-free path query evaluation w.r.t. the all-path query semantics is proposed. This algorithm is based on the generalized top-down parsing algorithm — GLL [22]. This solution uses derivation trees for the result representation which is more native for grammar based analysis. The algorithms in [9, 12] for the context-free path query evaluation w.r.t. the all-path query semantics can also be used for query evaluation using the relational and the single-path semantics.

Our work is inspired by Valiant [24], who proposed an algorithm for general context-free recognition in less than cubic time. This algorithm computes the same parsing table as the CYK algorithm but does this by offloading the most

intensive computations into calls to a Boolean matrix multiplication procedure. This approach not only provides an asymptotically more efficient algorithm but it also allows us to effectively apply GPGPU computing techniques. Valiant's algorithm computes the transitive closure a^+ of a square upper triangular matrix a. Valiant also showed that the matrix multiplication operation ($\times$) is essentially the same as $|N|^2$ Boolean matrix multiplications, where $|N|$ is the number of non-terminals of the given context-free grammar in Chomsky normal form.

Hellings [11] presented an algorithm for the context-free path query evaluation using the relational query semantics. According to Hellings, for a given graph $D = (V, E)$ and a grammar $G = (N, \Sigma, P)$ the context-free path query evaluation w.r.t. the relational query semantics reduces to a calculation of the context-free relations R_A. Thus, in this paper, we focus on the calculation of these context-free relations. Also, Hellings [11] presented an algorithm for the context-free path query evaluation using the single-path query semantics which evaluates paths of minimal length for all triples (A, m, n), but also noted that the length of these paths is not necessarily upper bounded. Thus, in this paper, we evaluate an arbitrary paths for all triples (A, m, n).

Yannakakis [26] analyzed the reducibility of various path querying problems to the calculation of the transitive closure. He formulated a problem of Valiant's technique generalization to the context-free path query evaluation w.r.t. the relational query semantics. Also, he assumed that this technique cannot be generalized for arbitrary graphs, though it does for acyclic graphs.

Thus, the possibility of reducing the context-free path query evaluation using the relational and the single-path query semantics to the calculation of the transitive closure is an open problem.

4 Context-free path querying by the calculation of transitive closure

In this section, we show how the context-free path query evaluation using the relational query semantics can be reduced to the calculation of matrix transitive closure a^{cf}, prove the correctness of this reduction, introduce an algorithm for computing the transitive closure a^{cf}, and provide a step-by-step demonstration of this algorithm on a small example.

4.1 Reducing context-free path querying to transitive closure

In this section, we show how the context-free relations R_A can be calculated by computing the transitive closure a^{cf}.

Let $G = (N, \Sigma, P)$ be a grammar and $D = (V, E)$ be a graph. We enumerate the nodes of the graph D from 0 to $(|V| - 1)$. We initialize the elements of $|V| \times |V|$ matrix a with $\varnothing$. Further, for every i and j we set

$$a_{i,j} = \{A_k \mid ((i, x, j) \in E) \wedge ((A_k \to x) \in P)\}.$$

Finally, we compute the transitive closure

$$a^{cf} = a^{(1)} \cup a^{(2)} \cup \cdots$$

where

$$a^{(i)} = a^{(i-1)} \cup (a^{(i-1)} \times a^{(i-1)}),$$

for $i \geq 2$ and $a^{(1)} = a$. For the transitive closure a^{cf}, the following statements hold.

Lemma 1. *Let $D = (V, E)$ be a graph, let $G = (N, \Sigma, P)$ be a grammar. Then for any i, j and for any non-terminal $A \in N$, $A \in a^{(k)}_{i,j}$ iff $(i, j) \in R_A$ and $i\pi j$, such that there is a derivation tree of the height $h \leq k$ for the string $l(\pi)$ and a context-free grammar $G_A = (N, \Sigma, P, A)$.*

A formal proof of the lemma 1 can be found in the appendix B.

Theorem 1 *Let $D = (V, E)$ be a graph and let $G = (N, \Sigma, P)$ be a grammar. Then for any i, j and for any non-terminal $A \in N$, $A \in a^{cf}_{i,j}$ iff $(i, j) \in R_A$.*

Proof. Since the matrix $a^{cf} = a^{(1)} \cup a^{(2)} \cup \cdots$, for any i, j and for any non-terminal $A \in N$, $A \in a^{cf}_{i,j}$ iff there is $k \geq 1$, such that $A \in a^{(k)}_{i,j}$. By the lemma 1, $A \in a^{(k)}_{i,j}$ iff $(i, j) \in R_A$ and there is $i\pi j$, such that there is a derivation tree of the height $h \leq k$ for the string $l(\pi)$ and a context-free grammar $G_A = (N, \Sigma, P, A)$. This completes the proof of the theorem.

We can, therefore, determine whether $(i, j) \in R_A$ by asking whether $A \in a^{cf}_{i,j}$. Thus, we show how the context-free relations R_A can be calculated by computing the transitive closure a^{cf} of the matrix a.

4.2 The algorithm

In this section we introduce an algorithm for calculating the transitive closure a^{cf} which was discussed in Section 4.1.

Let $D = (V, E)$ be the input graph and $G = (N, \Sigma, P)$ be the input grammar.

Algorithm 1 Context-free recognizer for graphs

```
1:  function contextFreePathQuerying(D, G)
2:      n ← a number of nodes in D
3:      E ← the directed edge-relation from D
4:      P ← a set of production rules in G
5:      T ← a matrix n × n in which each element is ∅
6:      for all (i, x, j) ∈ E do                    ▷ Matrix initialization
7:          T_{i,j} ← T_{i,j} ∪ {A | (A → x) ∈ P}
8:      while matrix T is changing do
9:          T ← T ∪ (T × T)                         ▷ Transitive closure T^{cf} calculation
10:     return T
```

Note that the matrix initialization in lines **6-7** of the Algorithm 1 can handle arbitrary graph D. For example, if a graph D contains multiple edges (i, x_1, j) and (i, x_2, j) then both the elements of a set $\{A \mid (A \to x_1) \in P\}$ and the elements of a set $\{A \mid (A \to x_2) \in P\}$ will be added to $T_{i,j}$.

We need to show that the Algorithm 1 terminates in a finite number of steps. Since each element of the matrix T contains no more than $|N|$ non-terminals, the total number of non-terminals in the matrix T does not exceed $|V|^2|N|$. Therefore, the following theorem holds.

Theorem 2 *Let $D = (V, E)$ be a graph and let $G = (N, \Sigma, P)$ be a grammar. Algorithm 1 terminates in a finite number of steps.*

Proof. It is sufficient to show, that the operation in the line **9** of the Algorithm 1 changes the matrix T only finite number of times. Since this operation can only add non-terminals to some elements of the matrix T, but not remove them, it can change the matrix T no more than $|V|^2|N|$ times.

Denote the number of elementary operations executed by the algorithm of multiplying two $n \times n$ Boolean matrices as $BMM(n)$. According to Valiant, the matrix multiplication operation in the line **9** of the Algorithm 1 can be calculated in $O(|N|^2 BMM(|V|))$. Denote the number of elementary operations executed by the matrix union operation of two $n \times n$ Boolean matrices as $BMU(n)$. Similarly, it can be shown that the matrix union operation in the line **9** of the Algorithm 1 can be calculated in $O(|N|^2 BMU(n))$. Since the line **9** of the Algorithm 1 is executed no more than $|V|^2|N|$ times, the following theorem holds.

Theorem 3 *Let $D = (V, E)$ be a graph and let $G = (N, \Sigma, P)$ be a grammar. Algorithm 1 calculates the transitive closure T^{cf} in $O(|V|^2|N|^3(BMM(|V|) + BMU(|V|)))$.*

4.3 An example

In this section, we provide a step-by-step demonstration of the proposed algorithm. For this, we consider the classical *same-generation query* [1].

The **example query** is based on the context-free grammar $G = (N, \Sigma, P)$ where:

- A set of non-terminals $N = \{S\}$.
- A set of terminals

$$\Sigma = \{subClassOf, subClassOf^{-1}, type, type^{-1}\}.$$

- A set of production rules P is presented in Figure 1.

Since the proposed algorithm processes only grammars in Chomsky normal form, we first transform the grammar G into an equivalent grammar $G' = (N', \Sigma', P')$ in normal form, where:

$$
\begin{aligned}
&0 : S \to subClassOf^{-1}\ S\ subClassOf \\
&1 : S \to type^{-1}\ S\ type \\
&2 : S \to subClassOf^{-1}\ subClassOf \\
&3 : S \to type^{-1}\ type
\end{aligned}
$$

Fig. 1. Production rules for the example query grammar.

- A set of non-terminals $N' = \{S, S_1, S_2, S_3, S_4, S_5, S_6\}$.
- A set of terminals

$$\Sigma' = \{subClassOf, subClassOf^{-1}, type, type^{-1}\}.$$

- A set of production rules P' is presented in Figure 2.

$$
\begin{aligned}
&0 :\ S \to S_1\ S_5 \\
&1 :\ S \to S_3\ S_6 \\
&2 :\ S \to S_1\ S_2 \\
&3 :\ S \to S_3\ S_4 \\
&4 : S_5 \to S\ S_2 \\
&5 : S_6 \to S\ S_4 \\
&6 : S_1 \to subClassOf^{-1} \\
&7 : S_2 \to subClassOf \\
&8 : S_3 \to type^{-1} \\
&9 : S_4 \to type
\end{aligned}
$$

Fig. 2. Production rules for the example query grammar in normal form.

We run the query on a graph presented in Figure 3.

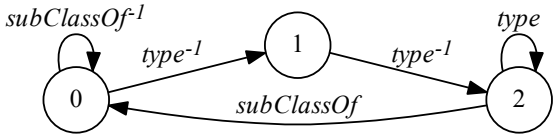

Fig. 3. An input graph for the example query.

We provide a step-by-step demonstration of the work with the given graph D and grammar G' of the Algorithm 1. After the matrix initialization in lines **6-7** of the Algorithm 1 we have a matrix T_0 presented in Figure 4.

We denote T_i as a matrix T after i-th loop iteration in lines **8-9** of the Algorithm 1. The calculation of the matrix T_1 is shown in Figure 5.

$$T_0 = \begin{pmatrix} \{S_1\} & \{S_3\} & \varnothing \\ \varnothing & \varnothing & \{S_3\} \\ \{S_2\} & \varnothing & \{S_4\} \end{pmatrix}$$

Fig. 4. Initial matrix for the example query.

$$T_0 \times T_0 = \begin{pmatrix} \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \{S\} \\ \varnothing & \varnothing & \varnothing \end{pmatrix}$$

$$T_1 = T_0 \cup (T_0 \times T_0) = \begin{pmatrix} \{S_1\} & \{S_3\} & \varnothing \\ \varnothing & \varnothing & \{S_3, S\} \\ \{S_2\} & \varnothing & \{S_4\} \end{pmatrix}$$

Fig. 5. The first iteration of computing the transitive closure for the example query.

When the algorithm at some iteration finds new paths in the graph D, then it adds corresponding nonterminals to the matrix T. For example, after the first loop iteration, non-terminal S is added to the matrix T. This non-terminal is added to the element with a row index $i = 1$ and a column index $j = 2$. This means that there is $i\pi j$ (a path π from node 1 to node 2), such that $S \xrightarrow{*} l(\pi)$. For example, such a path consists of two edges with labels $type^{-1}$ and $type$, and thus $S \xrightarrow{*} type^{-1}\ type$.

The calculation of the transitive closure is completed after k iterations when a fixpoint is reached: $T_{k-1} = T_k$. For the example query, $k = 6$, since $T_6 = T_5$. The remaining iterations of computing the transitive closure are presented in Figure 6.

$$T_2 = \begin{pmatrix} \{S_1\} & \{S_3\} & \varnothing \\ \{S_5\} & \varnothing & \{S_3, S, S_6\} \\ \{S_2\} & \varnothing & \{S_4\} \end{pmatrix}$$

$$T_3 = \begin{pmatrix} \{S_1\} & \{S_3\} & \{S\} \\ \{S_5\} & \varnothing & \{S_3, S, S_6\} \\ \{S_2\} & \varnothing & \{S_4\} \end{pmatrix}$$

$$T_4 = \begin{pmatrix} \{S_1, S_5\} & \{S_3\} & \{S, S_6\} \\ \{S_5\} & \varnothing & \{S_3, S, S_6\} \\ \{S_2\} & \varnothing & \{S_4\} \end{pmatrix}$$

$$T_5 = \begin{pmatrix} \{S_1, S_5, S\} & \{S_3\} & \{S, S_6\} \\ \{S_5\} & \varnothing & \{S_3, S, S_6\} \\ \{S_2\} & \varnothing & \{S_4\} \end{pmatrix}$$

Fig. 6. Remaining states of the matrix T.

Thus, the result of the Algorithm 1 for the example query is the matrix $T_5 = T_6$. Now, after constructing the transitive closure, we can construct the context-free relations R_A. These relations for each non-terminal of the grammar G' are presented in Figure 7.

$$
\begin{aligned}
R_S &= \{(0,0),(0,2),(1,2)\},\\
R_{S_1} &= \{(0,0)\},\\
R_{S_2} &= \{(2,0)\},\\
R_{S_3} &= \{(0,1),(1,2)\},\\
R_{S_4} &= \{(2,2)\},\\
R_{S_5} &= \{(0,0),(1,0)\},\\
R_{S_6} &= \{(0,2),(1,2)\}.
\end{aligned}
$$

Fig. 7. Context-free relations for the example query.

By the context-free relation R_S, we can conclude that there are paths in a graph D only from node 0 to node 0, from node 0 to node 2 or from node 1 to node 2, corresponding to the context-free grammar G_S. This conclusion is based on the fact that a grammar G'_S is equivalent to the grammar G_S and $L(G_S) = L(G'_S)$.

5 Conclusion and Future Work

In this paper, we shown how the context-free path query evaluation w.r.t. the relational and the single-path query semantics can be reduced to the calculation of matrix transitive closure. Also, we provided a formal proof of the correctness of the proposed reduction. In addition, we introduced an algorithm for computing this transitive closure, which allows us to efficiently apply GPGPU computing techniques. Finally, we shown the practical applicability of the proposed algorithm by running different implementations of our algorithm on real-world data.

We can identify several open problems for further research. In this paper we have considered only two semantics of context-free path querying but there are other important semantics, such as all-path query semantics [12] which requires to present all paths for all triples (A, m, n). Context-free path querying implemented with algorithm [9] can answer the queries in the all-path query semantics by constructing a parse forest. It is possible to construct a parse forest for a linear input by matrix multiplication [19]. Whether it is possible to generalize this approach for a graph input is an open question.

In our algorithm, we calculate the matrix transitive closure naively, but there are algorithms for the transitive closure calculation, which are asymptotically more efficient. Therefore, the question is whether it is possible to apply these

algorithms for the matrix transitive closure calculation to the problem of context-free path querying.

Also, there are conjunctive [18] and Boolean grammars [17], which have more expressive power than context-free grammars. Conjunctive language and Boolean path querying problems are undecidable [11] but our algorithm can be trivially generalized to work on this grammars because parsing with conjunctive and Boolean grammars can be expressed by matrix multiplication [19]. It is not clear what a result of our algorithm applied to this grammars would look like. Our hypothesis is that it would produce the upper approximation of a solution. Also, path querying problem w.r.t. the conjunctive grammars can be applied to static code analysis [28].

From a practical point of view, matrix multiplication in the main loop of the proposed algorithm may be performed on different GPGPU independently. It can help to utilize the power of multi-GPU systems and increase the performance of the context-free path querying.

There is an algorithm [14] for transitive closure calculation on directed graphs which generalized to handle graph sizes inherently larger then the DRAM memory available on the GPU. Therefore, the question is whether it is possible to apply this approach to the matrix transitive closure calculation in the problem of context-free path querying.

Acknowledgments

We are grateful to Dmitri Boulytchev, Ekaterina Verbitskaia, Marina Polubelova, Dmitrii Kosarev and Dmitry Koznov for their careful reading, pointing out some mistakes, and invaluable suggestions. This work is supported by grant from JetBrains Research.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc., 1995.
2. S. Abiteboul and V. Vianu. Regular path queries with constraints. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 122–133. ACM, 1997.
3. J. W. Anderson, Á. Novák, Z. Sükösd, M. Golden, P. Arunapuram, I. Edvardsson, and J. Hein. Quantifying variances in comparative rna secondary structure prediction. *BMC bioinformatics*, 14(1):149, 2013.
4. C. Barrett, R. Jacob, and M. Marathe. Formal-language-constrained path problems. *SIAM Journal on Computing*, 30(3):809–837, 2000.
5. O. Bastani, S. Anand, and A. Aiken. Specification inference using context-free language reachability. In *ACM SIGPLAN Notices*, volume 50, pages 553–566. ACM, 2015.
6. S. Che, B. M. Beckmann, and S. K. Reinhardt. Programming gpgpu graph applications with linear algebra building blocks. *International Journal of Parallel Programming*, pages 1–23, 2016.

7. N. Chomsky. On certain formal properties of grammars. *Information and control*, 2(2):137–167, 1959.
8. W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu. Adding regular expressions to graph reachability and pattern queries. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 39–50. IEEE, 2011.
9. S. Grigorev and A. Ragozina. Context-free path querying with structural representation of result. *arXiv preprint arXiv:1612.08872*, 2016.
10. D. Grune and C. J. H. Jacobs. *Parsing Techniques (Monographs in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
11. J. Hellings. Conjunctive context-free path queries. 2014.
12. J. Hellings. Querying for paths in graphs using context-free path queries. *arXiv preprint arXiv:1502.02242*, 2015.
13. T. Kasami. An efficient recognition and syntaxanalysis algorithm for context-free languages. Technical report, DTIC Document, 1965.
14. G. J. Katz and J. T. Kider Jr. All-pairs shortest-paths for large graphs on the gpu. In *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pages 47–55. Eurographics Association, 2008.
15. A. Mendelzon and P. Wood. Finding regular simple paths in graph databases. *SIAM J. Computing*, 24(6):1235–1258, 1995.
16. M. Nolé and C. Sartiani. Regular path queries on massive graphs. In *Proceedings of the 28th International Conference on Scientific and Statistical Database Management*, page 13. ACM, 2016.
17. A. Okhotin. Boolean grammars. *Information and Computation*, 194(1):19–48, 2004.
18. A. Okhotin. Conjunctive and boolean grammars: the true general case of the context-free grammars. *Computer Science Review*, 9:27–59, 2013.
19. A. Okhotin. Parsing by matrix multiplication generalized to boolean grammars. *Theoretical Computer Science*, 516:101–120, 2014.
20. T. Reps. Program analysis via graph reachability. *Information and software technology*, 40(11):701–726, 1998.
21. J. L. Reutter, M. Romero, and M. Y. Vardi. Regular queries on graph databases. *Theory of Computing Systems*, 61(1):31–83, 2017.
22. E. Scott and A. Johnstone. Gll parsing. *Electronic Notes in Theoretical Computer Science*, 253(7):177–189, 2010.
23. P. Sevon and L. Eronen. Subgraph queries by context-free grammars. *Journal of Integrative Bioinformatics*, 5(2):100, 2008.
24. L. G. Valiant. General context-free recognition in less than cubic time. *Journal of computer and system sciences*, 10(2):308–315, 1975.
25. G. Xu, A. Rountev, and M. Sridharan. Scaling cfl-reachability-based points-to analysis using context-sensitive must-not-alias analysis. In *ECOOP*, volume 9, pages 98–122. Springer, 2009.
26. M. Yannakakis. Graph-theoretic methods in database theory. In *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 230–242. ACM, 1990.
27. D. H. Younger. Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208, 1967.
28. Q. Zhang and Z. Su. Context-sensitive data-dependence analysis via linear conjunctive language reachability. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, pages 344–358. ACM, 2017.
29. X. Zhang, Z. Feng, X. Wang, G. Rao, and W. Wu. Context-free path queries on rdf graphs. In *International Semantic Web Conference*, pages 632–648. Springer, 2016.

A Equivalence of transitive closure definitions

To show the equivalence of a^{cf} and a^+ definitions of transitive closure, we introduce the partial order $\succeq$ on matrices with fixed size that have subsets of N as elements. For square matrices a, b of the same size we denote $a \succeq b$ iff $a_{i,j} \supseteq b_{i,j}$, for every i, j. For these two definitions of transitive closure, the following lemmas and theorem hold.

Lemma 2. *Let $G = (N, \Sigma, P)$ be a grammar, let a be a square matrix. Then $a^{(k)} \succeq a_+^{(k)}$ for any $k \geq 1$.*

Proof. (Proof by Induction)

Basis: The statement of the lemma holds for $k = 1$, since

$$a^{(1)} = a_+^{(1)} = a.$$

Inductive step: Assume that the statement of the lemma holds for any $k \leq (p-1)$ and show that it also holds for $k = p$ where $p \geq 2$. For any $i \geq 2$

$$a^{(i)} = a^{(i-1)} \cup (a^{(i-1)} \times a^{(i-1)}) \Rightarrow a^{(i)} \succeq a^{(i-1)}.$$

Hence, by the inductive hypothesis, for any $i \leq (p-1)$

$$a^{(p-1)} \succeq a^{(i)} \succeq a_+^{(i)}.$$

Let $1 \leq j \leq (p-1)$. The following holds

$$(a^{(p-1)} \times a^{(p-1)}) \succeq (a_+^{(j)} \times a_+^{(p-j)}),$$

since $a^{(p-1)} \succeq a_+^{(j)}$ and $a^{(p-1)} \succeq a_+^{(p-j)}$. By the definition,

$$a_+^{(p)} = \bigcup_{j=1}^{p-1} a_+^{(j)} \times a_+^{(p-j)}$$

and from this it follows that

$$(a^{(p-1)} \times a^{(p-1)}) \succeq a_+^{(p)}.$$

By the definition,

$$a^{(p)} = a^{(p-1)} \cup (a^{(p-1)} \times a^{(p-1)}) \Rightarrow a^{(p)} \succeq (a^{(p-1)} \times a^{(p-1)}) \succeq a_+^{(p)}$$

and this completes the proof of the lemma.

Lemma 3. *Let $G = (N, \Sigma, P)$ be a grammar, let a be a square matrix. Then for any $k \geq 1$ there is $j \geq 1$, such that $(\bigcup_{i=1}^{j} a_+^{(i)}) \succeq a^{(k)}$.*

Proof. (Proof by Induction)

Basis: For $k = 1$ there is $j = 1$, such that

$$a_+^{(1)} = a^{(1)} = a.$$

Thus, the statement of the lemma holds for $k = 1$.

Inductive step: Assume that the statement of the lemma holds for any $k \leq (p-1)$ and show that it also holds for $k = p$ where $p \geq 2$. By the inductive hypothesis, there is $j \geq 1$, such that

$$(\bigcup_{i=1}^{j} a_+^{(i)}) \succeq a^{(p-1)}.$$

By the definition,

$$a_+^{(2j)} = \bigcup_{i=1}^{2j-1} a_+^{(i)} \times a_+^{(2j-i)}$$

and from this it follows that

$$(\bigcup_{i=1}^{2j} a_+^{(i)}) \succeq (\bigcup_{i=1}^{j} a_+^{(i)}) \times (\bigcup_{i=1}^{j} a_+^{(i)}) \succeq (a^{(p-1)} \times a^{(p-1)}).$$

The following holds

$$(\bigcup_{i=1}^{2j} a_+^{(i)}) \succeq a^{(p)} = a^{(p-1)} \cup (a^{(p-1)} \times a^{(p-1)}),$$

since

$$(\bigcup_{i=1}^{2j} a_+^{(i)}) \succeq (\bigcup_{i=1}^{j} a_+^{(i)}) \succeq a^{(p-1)}$$

and

$$(\bigcup_{i=1}^{2j} a_+^{(i)}) \succeq (a^{(p-1)} \times a^{(p-1)}).$$

Therefore there is $2j$, such that

$$(\bigcup_{i=1}^{2j} a_+^{(i)}) \succeq a^{(p)}$$

and this completes the proof of the lemma.

Theorem 4 *Let $G = (N, \Sigma, P)$ be a grammar, let a be a square matrix. Then $a^+ = a^{cf}$.*

Proof. By the lemma 2, for any $k \geq 1$, $a^{(k)} \succeq a_+^{(k)}$. Therefore

$$a^{cf} = a^{(1)} \cup a^{(2)} \cup \cdots \succeq a_+^{(1)} \cup a_+^{(2)} \cup \cdots = a^+.$$

By the lemma 3, for any $k \geq 1$ there is $j \geq 1$, such that

$$(\bigcup_{i=1}^{j} a_+^{(i)}) \succeq a^{(k)}.$$

Hence

$$a^+ = (\bigcup_{i=1}^{\infty} a_+^{(i)}) \succeq a^{(k)},$$

for any $k \geq 1$. Therefore

$$a^+ \succeq a^{(1)} \cup a^{(2)} \cup \cdots = a^{cf}.$$

Since $a^{cf} \succeq a^+$ and $a^+ \succeq a^{cf}$,

$$a^+ = a^{cf}$$

and this completes the proof of the theorem.

B Proof of the lemma 1

Proof. (Proof by Induction)

Basis: Show that the statement of the lemma holds for $k = 1$. For any i, j and for any non-terminal $A \in N$, $A \in a_{i,j}^{(1)}$ iff there is $i\pi j$ that consists of a unique edge e from node i to node j and $(A \to x) \in P$ where $x = l(\pi)$. Therefore $(i, j) \in R_A$ and there is a derivation tree of the height $h = 1$, shown in Figure 8, for the string x and a context-free grammar $G_A = (N, \Sigma, P, A)$. Thus, it has been shown that the statement of the lemma holds for $k = 1$.

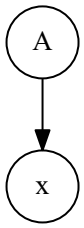

Fig. 8. The derivation tree of the height $h = 1$ for the string $x = l(\pi)$.

Inductive step: Assume that the statement of the lemma holds for any $k \leq (p-1)$ and show that it also holds for $k = p$ where $p \geq 2$. For any i, j and for any non-terminal $A \in N$,

$$A \in a_{i,j}^{(p)} \text{ iff } A \in a_{i,j}^{(p-1)} \text{ or } A \in (a^{(p-1)} \times a^{(p-1)})_{i,j},$$

since

$$a^{(p)} = a^{(p-1)} \cup (a^{(p-1)} \times a^{(p-1)}).$$

Let $A \in a^{(p-1)}_{i,j}$. By the inductive hypothesis, $A \in a^{(p-1)}_{i,j}$ iff $(i,j) \in R_A$ and there exists $i\pi j$, such that there is a derivation tree of the height $h \leq (p-1)$ for the string $l(\pi)$ and a context-free grammar $G_A = (N, \Sigma, P, A)$. The statement of the lemma holds for $k = p$, since the height h of this tree is also less than or equal to p.

Let $A \in (a^{(p-1)} \times a^{(p-1)})_{i,j}$. By the definition of the binary operation $(\cdot)$ on arbitrary subsets, $A \in (a^{(p-1)} \times a^{(p-1)})_{i,j}$ iff there are r, $B \in a^{(p-1)}_{i,r}$ and $C \in a^{(p-1)}_{r,j}$, such that $(A \to BC) \in P$. Hence, by the inductive hypothesis, there are $i\pi_1 r$ and $r\pi_2 j$, such that $(i,r) \in R_B$ and $(r,j) \in R_C$, and there are the derivation trees T_B and T_C of heights $h_1 \leq (p-1)$ and $h_2 \leq (p-1)$ for the strings $w_1 = l(\pi_1)$, $w_2 = l(\pi_2)$ and the context-free grammars G_B, G_C respectively. Thus, the concatenation of paths π_1 and π_2 is $i\pi j$, where $(i,j) \in R_A$ and there is a derivation tree of the height $h = 1 + max(h_1, h_2)$, shown in Figure 9, for the string $w = l(\pi)$ and a context-free grammar G_A.

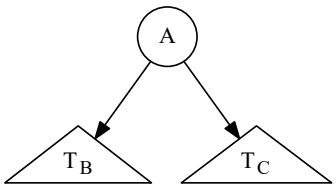

Fig. 9. The derivation tree of the height $h = 1 + max(h_1, h_2)$ for the string $w = l(\pi)$, where T_B and T_C are the derivation trees for strings w_1 and w_2 respectively.

The statement of the lemma holds for $k = p$, since the height $h = 1 + max(h_1, h_2) \leq p$. This completes the proof of the lemma.