GLL-based Context-Free Path Querying for Neo4j

Vlada Pogozhelskaya
Saint Petersburg State University
St. Petersburg, Russia
JetBrains Research
St. Petersburg, Russia
pogozhelskaya@gmail.com

Anna Vlasova
JetBrains Research
St. Petersburg, Russia
anna.vlasova@jetbrains.com

Semyon Grigorev
Saint Petersburg State University
St. Petersburg, Russia
JetBrains Research
St. Petersburg, Russia
s.v.grigoriev@spbu.ru
semyon.grigorev@jetbrains.com

ABSTRACT

We show that a fast GLL parsing algorithm provides *context-free path querying* for Neo4j graph database with reasonable performance. The proposed solution solves both the *reachability* and the *all paths* problems for the *all pairs* and the *multiple sources* cases. The evaluation on real-world graphs demonstrates that our solution is more than 25 times faster than the previous solution for Neo4j and is comparable, in some cases, with the linear algebra based solution for RedisGraph.

CCS CONCEPTS

• **Information systems** → **Graph-based database models**; **Query languages for non-relational engines**; • **Theory of computation** → **Grammars and context-free languages**; • **Mathematics of computing** → *Paths and connectivity problems.*

KEYWORDS

Graph database, context-free path querying, CFPQ, reachability problem, all paths problem, generalized LL, GLL

ACM Reference Format:
Vlada Pogozhelskaya, Anna Vlasova, and Semyon Grigorev. 2018. GLL-based Context-Free Path Querying for Neo4j. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Context-free path querying (CFPQ) allows one to use context-free grammars to specify constraints on paths in edge-labeled graphs. Context-free constraints are more expressive than the regular ones (RPQ) and can be used for graph analysis in such areas as bioinformatics (hierarchy analysis [16], similarity queries [13]), data provenance analysis [8], static code analysis [9, 17]. Although a lot of algorithms for CFPQ were proposed, poor performance on real-world graphs and bad integration with real-world graph databases and graph analysis systems still are problems which hinder the adoption of CFPQ.

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

The problem with the performance of CFPQ algorithms in real-world scenarios was pointed out by Jochem Kuijpers [6] as a result of an attempt to extend Neo4j graph database with CFPQ. Several algorithms, based on such techniques as LR parsing algorithm [10], dynamic programming [4], LL parsing algorithm [7], linear algebra [2], were implemented using Neo4j as a graph storage and evaluated. None of them was performant enough to be used in real-world applications.

Since Jochem Kuijpers pointed out the performance problem, it was shown that linear algebra based CFPQ algorithms, which operate over the adjacency matrix of the input graph and utilize parallel algorithms for linear algebraic operations, demonstrate good performance [14]. Moreover, the matrix-based CFPQ algorithm is a base for the first full-stack support of CFPQ by extending RedisGrpah graph database [14].

However adjacency matrix is not the only possible format for graph representation, and data format conversion may take a lot of time, thus it is not applicable in some cases. As a result, the development of a performant CFPQ algorithm for graph representations not based on matrices and its integration with real-world graph databases is still an open problem. Moreover, while the *all pairs context-free constrained reachability* is widely discussed in the community, such practical cases as the *all paths* queries and the *multiple sources* queries are not studied enough. Additionally, to the best of our knowledge, ways to provide parallel solutions based not on linear-algebra-oriented algorithms are still not investigated. In the multi- and many-core world and the big data era, it is important to provide a parallel solution for CFPQ.

It was shown that the generalized LL (GLL) parsing algorithm can be naturally generalized to CFPQ algorithm [3]. Moreover, this provides a natural solution not only for the *reachability* problem but also for the *all paths* problem. At the same time, there exists a high-performance GLL parsing algorithm [1] and its implementation in Iguana project[1]. In this paper, we generalize the parsing algorithm implemented in Iguana to get a high-performance CFPQ algorithm and integrate it with the Neo4j graph database. So, we make the following contributions in the paper.

- We provide an implementation of the GLL-based CFPQ algorithm. Implementation is based on the high-performance GLL parsing algorithm, which we modified to handle graphs. We provide modifications for both the *reachability* and the *all paths* cases.
- We integrate the implemented algorithm with Neo4j using Native Java API. Currently, we use Neo4j as a graph storage

[1] Iguana parsing framework: https://iguana-parser.github.io/. Accessed: 12.11.2021.

and do not provide the query language support. Implementing a query language extension amounts to a lot of additional effort and is a part of future work.
- We evaluate the proposed solution on several real-world graphs for both the *all pairs* and the *multiple sources* scenarios. Our evaluation shows that the proposed solution is more than 25 times faster than the previous solution for Neo4j and is comparable, in some cases, with the linear algebra based solution for RedisGraph. Moreover, performance of the *all paths* problem for *multiple sources* CFPQ is reasonable for relatively small start sets.

2 PRELIMINARIES

We use a directed edge-labelled graph as a data model. We denote a graph as $D = \langle V, E, L \rangle$, where V is a finite set of vertices, $E \subseteq V \times L \times V$ is a set of edges, and L is a set of edge labels. A path π in a graph D is a sequence of edges: $u \xrightarrow{l_0} \dots \xrightarrow{l_m} v$. We denote a path between vertices u and v as $u\pi v$. Function ω maps a path to a word by concatenating the labels of its edges: $\omega(\pi) = \omega(u \xrightarrow{l_0} \dots \xrightarrow{l_m} v) = l_0 \dots l_m$.

A context-free grammar is a 4-tuple $G = \langle \Sigma, N, P, S \rangle$, where Σ is a finite set of terminals, N is a finite set of nonterminals, $S \in N$ is the start nonterminal, and P is a set of productions. Each production has the following form: $M \to w$, where $M \in N$ is a left-hand side of production, and $w \in (\Sigma \cup N)^*$ is a right-hand side of production. $S \to w_1 \mid w_2$ is used instead of $S \to w_1; S \to w_2$. Some example grammars are presented in equations 5, 6, and 7. A derivation step is a production application: having a sequence of form $w_1 B w_2$, where $B \in N$ and $w_1, w_2 \in (\Sigma \cup N)^*$, and a production $B \to w_3$, one gets a sequence $w_1 w_3 w_2$, by replacing the left-hand side of the production with its right-hand side. A word w is derivable in a grammar if there is a sequence of derivation steps such that the initial sequence is the start nonterminal of the grammar and w is a final sequence. The language specified by the grammar G (denoted $\mathcal{L}(G)$) is a set of words derivable from the start nonterminal of the given grammar.

We can formulate a set of problems for the graph D and the context-free grammar G:

$$R = \{(u, v) \mid \text{exists a path } u\pi v \text{ in } D, \omega(\pi) \in \mathcal{L}(G)\} \quad (1)$$

$$Q = \{\pi \mid \text{exists a path } u\pi v \text{ in } D, \omega(\pi) \in \mathcal{L}(G)\} \quad (2)$$

$$R(I) = \{(u, v) \mid \text{exists a path } u\pi v \text{ in } D, u \in I, \omega(\pi) \in \mathcal{L}(G)\} \quad (3)$$

$$Q(I) = \{\pi \mid \text{exists a path } u\pi v \text{ in } D, u \in I, \omega(\pi) \in \mathcal{L}(G)\} \quad (4)$$

Here 1 is the *all pairs reachability problem*, 2 is the *all pairs all paths problem*. In 3 and 4, the additional parameter I denotes the set of start vertices. Thus 3 and 4 — are the *multiple source reachability* and the *multiple source all paths* problems respectively. Note, that for the *all paths* problems, the result Q can be an infinite set. Typically, the algorithms for these problems build a finite structure which contains all paths of the interest, not the explicit set of paths.

Our solution is based on the generalized LL parsing algorithm [11] which was shown to generalize well to graph processing [3]. We use Iguana project, which provides the Java implementation of a modified GLL algorithm [1], as a base for our solution. As a result of parsing, GLL constructs a Shared Packed Parse Forest (SPPF) — a special data structure which represents all possible derivations of the input in a compressed form. It was shown in [3] that SPPF can be naturally used as a finite representation of the *all paths* problems 2 and 4.

3 SOLUTION DESCRIPTION

In order to handle graphs instead of linear input (strings), we modify Iguana according to [3]. We also provide the ability to switch between the SPPF construction and reachability facts calculation. It avoids unnecessary memory usage when the paths calculation is not required.

As far as the original GLL is aimed to handle arbitrary (even ambiguous) context-free grammars, our solution can handle arbitrary grammars too. It makes the solution less restrictive with regard to a query specification language, thus being more user-friendly.

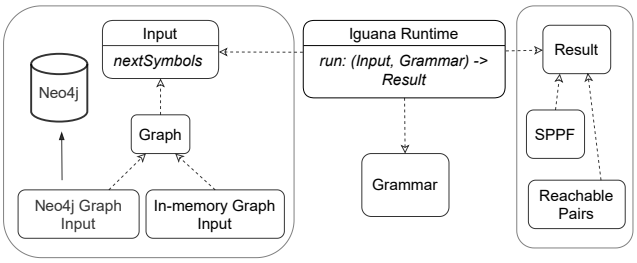

Figure 1: Architecture of the solution

The main parts of the solution are presented in figure 1. `Iguana Runtime` takes an `Input` and a context-free `Grammar` to produce the `Result`. `Input` abstracts a data structure with the ability to get the next symbols for the given position. One example of `Input` is `Graph`, in which the position is a vertex, and the next symbols are the labels of its outgoing edges. We implement two different graphs. The first is a simple in-memory graph, and the second is a wrapper for Neo4j. Communication with the database is done using the Neo4j Native Java API. We used an embedded database, which means it is run inside of the application and is not used as an external service. Note that the architecture is extensible, and one can provide their own implementation of `Graph` to enable context-free path querying for a new graph storage.

Along with the existing modifications of GLL we made a Neo4j-specific one. Neo4j returns the outgoing edges of a vertex as a `Stream` and it is important to prevent early stream forcing, thus we chaged all GLL internals to ensure that. This also has an added benefit that the query result is a stream, and thus it is possible to get the results on demand.

The *multiple sources* querying is natural for GLL as far as this algorithm is *directed* as opposed to the naive linear algebra based algorithms. So, to support the multiple sources context-free path querying, we need only to be able to specify the start vertices for traversing, and no nontrivial modifications of GLL are required.

4 EVALUATION

To assess the applicability of the proposed solution we evaluate it on a number of real-world graphs and queries. We run all experiments on the PC with Ubuntu 20.04 installed. It has Intel Core

i7-6700 CPU, 3.4GHz, 4 threads (hyper-threading is turned off), and DDR4 64Gb RAM. We use Java HotSpot(TM) 64-Bit server virtual machine (build 15.0.2+7-27, mixed mode, sharing). JVM was configured to use 55Gb of heap memory: both `xms` and `xmx` are set to 55Gb. We use Neo4j 4.0.3. Almost all configurations of Neo4j is default. We only set the total off-heap transaction memory (`tx_state_max_off_heap_memory` parameter) to 24Gb, and `pagecache_warmup_enabled` is set to `true`.

4.1 Dataset Description

For the evaluation, we use a number of graphs from CFPQ_Data[2] data set. We selected a number of graphs related to RDF analysis, as well as a number of graphs extracted from the Linux sources which related to static code analysis problems. A detailed description of the graphs, namely the number of vertices and edges and the number of edges labeled by tags used in queries, is in table 1.

All queries used in our evaluation are variats of *same-generation query*. For the *RDF* graphs we use the same queries as in the other works for CFPQ algorithms evaluation: G_1 (5), G_2 (6), and *Geo* (7). The queries are expressed as context-free grammars where S is a nonterminal, *subClassOf, type, broaderTransitive,* $\overline{subClassOf}$, $\overline{type}$, $\overline{broaderTransitive}$ are terminals or the labels of edges. We denote the inverse of an x relation and the respective edge as $\overline{x}$.

$$\begin{aligned} S \rightarrow & \overline{subClassOf}\ S\ subClasOf \mid \overline{type}\ S\ type \\ & \mid \overline{subClassOf}\ subClassOf \mid \overline{type}\ type \end{aligned} \tag{5}$$

$$S \rightarrow \overline{subClassOf}\ S\ subClasOf \mid subClassOf \tag{6}$$

$$\begin{aligned} S \rightarrow & broaderTransitive\ S\ \overline{broaderTransitive} \\ & \mid broaderTransitive\ \overline{broaderTransitive} \end{aligned} \tag{7}$$

For *Program analysis* graphs we use a *PointsTo* query (8) which describes a points-to analysis [17].

$$\begin{aligned} M &\rightarrow \overline{d}\ V\ d \\ V &\rightarrow (M?\ \overline{a})^*\ M?\ (a\ M?)^* \end{aligned} \tag{8}$$

4.2 Scenarios Description

We evaluate the proposed solution on the following scenarios which cover both OLTP and OLAP workloads: **all-pairs reachability**, **multiple sources reachability**, and **multiple sources all paths**. We omit **all-pairs all path** because it seems impractical: the detailed analysis is often required only for paths within a specific subgraph, not the entire graph. The important case of the *single source* querying is a partial case of the multiple sources querying.

The start sets for the multiple sources querying are generated from all vertices of a graph with a random permutation which splits them into chunks of a specific size. We use chunks of size 1, 10, 50, 100, 500, and 1000. We use the same chunks for both *reachability* and *all paths* cases.

To check the correctness of our solution and to force the result stream, we compute the number of reachable pairs for each query.

[2]CFPQ_Data is a public set of graphs and grammars for CFPQ algorithms evaluation. GitHub repository: https://github.com/JetBrains-Research/CFPQ_Data. Accessed: 12.11.2021.

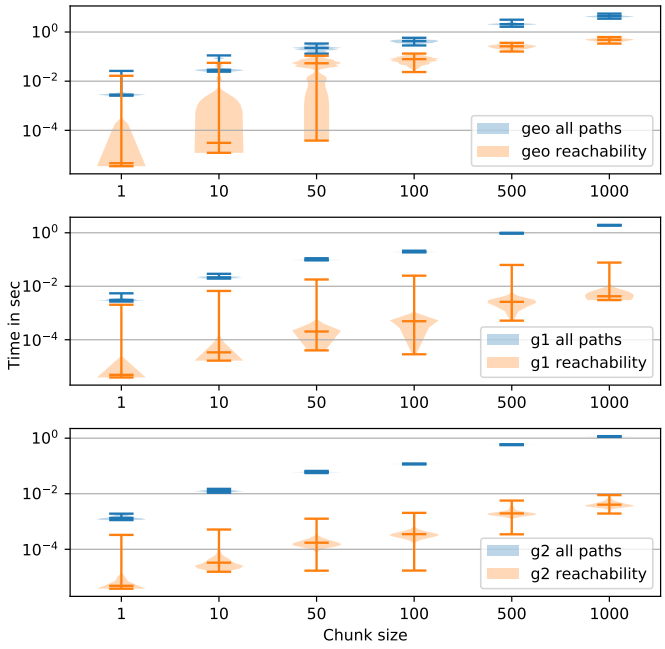

Figure 2: Multiple source CFPQ for *Geospecies* graph and *Geo* grammar: reachability and all paths

4.3 Evaluation Results

The results of the **all pairs reachability** queries evaluation are presented in table 2. Sign '–' in cells means that the respective query is not applicable to the graph, so time is not measured.

We can see that query evaluation time depends not only on a graph size or its sparsity, but also on an inner structure of the graph. For example, the relatively small graph *go_hierarchy* fully consists of edges used in queries G_1 and G_2, thus evaluation time for these queries is significantly bigger than for some bigger but more sparse graphs. Note that the size of the answer is not a good metric, because, for example, answers for *Geo* query, and *Enzyme* and *Geospecies* graphs, are calculated faster than the answers for *go_hierarchy*. We think that the creation of relevant metrics for CFPQ queries evaluation time prediction is a challenging problem by itself and should be tackled in the future.

Previous similar solution requires more then 6900 seconds to evaluate the *Geo* query for the *Geospecies* graph [6]. Thus we significantly improved the performance of CFPQ for Neo4j.

Also we can see that queries related to code analysis can be evaluated in reasonable time even for relatively big graphs. Thus, CFPQ can be used to improve Neo4j-based code analysis systems.

The results of the **multiple source reachablity and all paths** queries on *Geospecies* graph are presented in figure 2. For each size of chunks we evaluate all three queries for all chunks and represent results using standard violin plot with median.

First of all, we can see that the single-source queries are reasonably fast in both the *reachability* and the *all paths* cases: median time is less than 10^{-4} seconds for reachability queries, and is less than 10^{-2} seconds for *all paths* queries. Even for queries G_1 and

	Graph name	$\|V\|$	$\|E\|$	#subClassOf	#type	#broaderTransitive	#a	#d
RDF	Go hierarchy	45 007	490 109	490 109	0	0	–	–
	Enzyme	48 815	86 543	8 163	14 989	8 156	–	–
	Eclass_514en	239 111	360 248	90 962	72 517	0	–	–
	Geospecies	450 609	2 201 532	0	89 065	20 867	–	–
	Go	582 929	1 437 437	94 514	226 481	0	–	–
	Taxonomy	5 728 398	14 922 125	2 112 637	2 508 635	0	–	–
Program analysis	Init	2 446 224	2 112 809	–	–	–	481 994	1 630 815
	Drivers	4 273 803	3 707 769	–	–	–	858 568	2 849 201
	Kernel	11 254 434	9 484 213	–	–	–	1 981 258	7 502 955

Table 1: Graphs for evaluation: number of vertices and edges, and number of edges with specific label

Graph name	G_1		G_2		*Geo*		*PointsTo*	
	time (sec)	#answer	time (sec)	#answer	time (sec)	#answer	time (sec)	#answer
Go hierarchy	564,72	588 976	2813,50	738 937	–	–	–	–
Enzyme	0,19	396	0,17	8163	8,54	14 267 542	–	–
Eclass_514en	295,06	90 994	279,80	96 163	–	–	–	–
Geospecies	2,64	85	2,00	0	256,86	226 669 749	–	–
Go	11,18	640 316	10,00	659 501	–	–	–	–
Taxonomy	43,72	151 706	29,58	2 112 637	–	–	–	–
Init	–	–	–	–	–	–	113,35	3 783 769
Drivers	–	–	–	–	–	–	736,81	18 825 025
Kernel	–	–	–	–	–	–	850,46	16 747 731

Table 2: Single thread all-pairs reachability performance results: time in seconds, #answer is a number of reachable pairs

G_2, which return relatively small answers, time required for the all paths queries is bigger than for reachability queries. Moreover, time grows with the size of a chunk. Thus it is helpful to choose between the *reachability* and *all paths* modes to improve performance even for small start sets.

The median time is less than 1 second for all chunk sizes for the *reachability* problem for the *Geo* query. It is also true for the chunk sizes of less than 100 for the *all paths* problem.

Comparison with CFPQ implementation for RedisGraph graph database [15] shows that our solution is comparable with matrix-based algorithms. Note that our solution is sequential, while RedisGraph uses SuiteSparse:GraphBLAS[3] which is highly parallel. While the *all pairs reachability* queries are slower for Neo4j, the *multiple sources* queries for small chunks can be evaluated in comparable time. Moreover, our solution can compute all paths, while the solution for RedisGraph can solve only the reachability problem. Note that RedisGraph provides full stack support of CFPQ, so time for such steps as execution plan building is included into measured time, while we measure only query execution time for Neo4j, omitting Cypher parsing, execution plan building, etc.

Finally, we conclude that not only the linear algebra based CFPQ algorithms can be used in real-world graph analysis systems. It is not likely to outperform the linear algebra based CFPQ algorithms for the *all pairs reachability*. But the GLL-based algorithm is promising for the *multiple sources* queries with a relatively small start set, especially in the *all paths* case.

[3]SuiteSparse:GraphBLAS is a high performance implementation of GraphBLAS API in C language. Home page of project: https://people.engr.tamu.edu/davis/GraphBLAS.html. Accessed: 12.11.2021.

5 CONCLUSION

In this work we present a GLL-based context-free path querying algorithm for Neo4j graph database. The implemenattion is available on GitHub: https://github.com/JetBrains-Research/GLL4Graph.

In future, we plan to investigate how to solve the important problems of the single path and the shortest path CFPQ. We also plan to adopt the GLL algorithm [12] which handles grammars in the Extended Backus-Naur Form [5] directly to make graph query languages more user-friendly.

The most important direction for future work is to find a way to parallelize the GLL-based CFPQ algorithm. The naive data parallel version improves the *Geo* query on *Geospecies* graph by 30% as compared with the sequential version. Unfortunatelly, is has no effect for the *Go hierarchy* because of the more complex dependencies in the graph. The efficient parallelization of the GLL-based CFPQ algorithm is a challenging problem for future research.

ACKNOWLEDGMENTS

The research was supported by the Russian Science Foundation, grant №18-11-00100.

We thank Adrian Johnstone for his pointing out the Generilized LL algorithm in our discussion at Parsing@SLE–2013 which gave us the motivation to develop the presented solution.

REFERENCES

[1] Ali Afroozeh and Anastasia Izmaylova. 2015. Faster, Practical GLL Parsing. In *Compiler Construction*, Björn Franke (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 89–108.

[2] Rustam Azimov and Semyon Grigorev. 2018. Context-free Path Querying by Matrix Multiplication. In *Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)* (Houston, Texas) *(GRADES-NDA '18)*. ACM, New York, NY, USA, Article 5, 10 pages. https://doi.org/10.1145/3210259.3210264

[3] Semyon Grigorev and Anastasiya Ragozina. 2017. Context-free Path Querying with Structural Representation of Result. In *Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia* (St. Petersburg, Russia) *(CEE-SECR '17)*. ACM, New York, NY, USA, Article 10, 7 pages. https://doi.org/10.1145/3166094.3166104

[4] Jelle Hellings. 2014. Conjunctive context-free path queries. In *Proceedings of ICDT'14*. 119–130.

[5] Kees Hemerik. 2009. Towards a Taxonomy for ECFG and RRPG Parsing. In *Language and Automata Theory and Applications*, Adrian Horia Dediu, Armand Mihai Ionescu, and Carlos Martín-Vide (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 410–421.

[6] Jochem Kuijpers, George Fletcher, Nikolay Yakovets, and Tobias Lindaaker. 2019. An Experimental Study of Context-Free Path Query Evaluation Methods. In *Proceedings of the 31st International Conference on Scientific and Statistical Database Management* (Santa Cruz, CA, USA) *(SSDBM '19)*. ACM, New York, NY, USA, 121–132. https://doi.org/10.1145/3335783.3335791

[7] Ciro M. Medeiros, Martin A. Musicante, and Umberto S. Costa. 2018. Efficient Evaluation of Context-Free Path Queries for Graph Databases. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing* (Pau, France) *(SAC '18)*. Association for Computing Machinery, New York, NY, USA, 1230–1237. https://doi.org/10.1145/3167132.3167265

[8] H. Miao and A. Deshpande. 2019. Understanding Data Science Lifecycle Provenance via Graph Segmentation and Summarization. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 1710–1713.

[9] Jakob Rehof and Manuel Fähndrich. 2001. Type-Base Flow Analysis: From Polymorphic Subtyping to CFL-Reachability. *SIGPLAN Not.* 36, 3 (Jan. 2001), 54–66. https://doi.org/10.1145/373243.360208

[10] Fred C. Santos, Umberto S. Costa, and Martin A. Musicante. 2018. A Bottom-Up Algorithm for Answering Context-Free Path Queries in Graph Databases. In *Web Engineering*, Tommi Mikkonen, Ralf Klamma, and Juan Hernández (Eds.). Springer International Publishing, Cham, 225–233.

[11] Elizabeth Scott and Adrian Johnstone. 2010. GLL Parsing. *Electronic Notes in Theoretical Computer Science* 253, 7 (2010), 177–189. https://doi.org/10.1016/j.entcs.2010.08.041 Proceedings of the Ninth Workshop on Language Descriptions Tools and Applications (LDTA 2009).

[12] Elizabeth Scott and Adrian Johnstone. 2018. GLL syntax analysers for EBNF grammars. *Science of Computer Programming* 166 (2018), 120–145. https://doi.org/10.1016/j.scico.2018.06.001

[13] Petteri Sevon and Lauri Eronen. 2008. Subgraph Queries by Context-free Grammars. *Journal of Integrative Bioinformatics* 5, 2 (2008), 157 – 172. https://doi.org/10.1515/jib-2008-100

[14] Arseniy Terekhov, Artyom Khoroshev, Rustam Azimov, and Semyon Grigorev. 2020. Context-Free Path Querying with Single-Path Semantics by Matrix Multiplication. In *Proceedings of the 3rd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)* (Portland, OR, USA) *(GRADES-NDA'20)*. Association for Computing Machinery, New York, NY, USA, Article 5, 12 pages. https://doi.org/10.1145/3398682.3399163

[15] Arseniy Terekhov, Vlada Pogozhelskaya, Vadim Abzalov, Timur Zinnatulin, and Semyon V. Grigorev. 2021. Multiple-Source Context-Free Path Querying in Terms of Linear Algebra. In *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, Yannis Velegrakis, Demetris Zeinalipour-Yazti, Panos K. Chrysanthis, and Francesco Guerra (Eds.). OpenProceedings.org, 487–492. https://doi.org/10.5441/002/edbt.2021.56

[16] Xiaowang Zhang, Zhiyong Feng, Xin Wang, Guozheng Rao, and Wenrui Wu. 2016. Context-Free Path Queries on RDF Graphs. In *The Semantic Web – ISWC 2016*, Paul Groth, Elena Simperl, Alasdair Gray, Marta Sabou, Markus Krötzsch, Freddy Lecue, Fabian Flöck, and Yolanda Gil (Eds.). Springer International Publishing, Cham, 632–648.

[17] Xin Zheng and Radu Rugina. 2008. Demand-driven Alias Analysis for C. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Francisco, California, USA) *(POPL '08)*. ACM, New York, NY, USA, 197–208. https://doi.org/10.1145/1328438.1328464