Teach Formal Languages Together With Graph Querying for Great Power

Nikolay Ponomarev
The Thørväld Group
Hekla, Iceland
larst@affiliation.org

Efim Kubishkin
The Thørväld Group
Hekla, Iceland
larst@affiliation.org

Egor Orachev
The Thørväld Group
Hekla, Iceland
larst@affiliation.org

Vadim Abzalov
The Thørväld Group
Hekla, Iceland
larst@affiliation.org

Rustam Azimov
The Thørväld Group
Hekla, Iceland
larst@affiliation.org

Semyon Grigorev
s.v.grigoriev@mail.spbu.ru
Saint Petersburg State University
St. Petersburg, Russia

ABSTRACT

Formal language theory has deep interconnection with graph structured data analysis: it is required to develop both query processing engines frontend (parser and lexer) and formal language constrained path querying (FLPQ) evaluation algorithms. Efficient solution of respective problems requires strong theory knowledge and applied skills not only in formal languages, but also other areas such as graph theory and high-performance computing. We propose a course that is developed for software engineers, and focused on formal language related aspects of graph analysis, including query parsing and FLPQ evaluation algorithms.

CCS CONCEPTS

• Social and professional topics → Computing education; **• Theory of computation → Formal languages and automata theory**; **• Information systems → Query languages**.

KEYWORDS

Graph querying, formal language theory, formal language constrained path querying, education

ACM Reference Format:

Nikolay Ponomarev, Efim Kubishkin, Egor Orachev, Vadim Abzalov, Rustam Azimov, and Semyon Grigorev. 2018. Teach Formal Languages Together With Graph Querying for Great Power. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Bridging the gap between fundamental disciplines and their applications is one of the important problems of education in software engineering. Graph structured data analysis involves broad range of areas including graph theory, high-performance computing, complexity analysis. Development of high-quality data analysis solutions requires both deep theory knowledge and strong applied experience. It is a challenge for education to provide comprehensive view of the area and to develop the necessary skills.

One of the disciplines that has deep interconnection with graph structured data analysis is a formal language theory. Formal languages is a fundamental part of graph query languages in, at least, two senses. First of all, formal language theory is necessary to design query engines frontend, namely parser and lexer. The second one is that formal language constrained path querying is an important part of modern graph query languages. An ability to utilize regular languages to specify path constrains (RPQ) is a part of GQL query language that is an ISO standard [14], and it is a part of SPARQL that is W3C standard [1]. Context-free languages constrained path querying (CFPQ) is studied intensively [13, 15, 16, 27] and respective proposal for Cypher language exists[1].

Even these two directions should be investigated together to find the best way to design query language to be able to express respective formal language constraints. At the same time, to apply formal languages for graph analysis efficiently we should involve graph theory and high-performance computing. Many questions regarding query language expressiveness power, query evaluation results representation, creation of respective algorithms that can handle huge graphs, may be answered easier if utilize techniques from these disciplines together.

In this work we propose a course[2] that is developed for software engineers, and focused on formal language related aspects of graph analysis, including query parsing and formal language constrained path query execution algorithms. This work is organized as follows.

- The first part is motivation where we argue why we do exactly what we do and why we do it exactly such a way.
- The second part is a course structure description that consists of technical environment description, high-level structure of the course, including exercises and tests examples.
- The third part is a discussion of existing course that includes current limitations and drawbacks, and comparison with other related courses.
- The last part is a brief conclusion and possible future improvements of the presented course.

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

[1]OpenCypher Path Pattern Queries that allows one to specify context-free constraints: https://github.com/opencypher/openCypher/pull/187.

[2]Materials for the course: https://github.com/FormalLanguageConstrainedPathQuerying/formal-lang-course.

2 MOTIVATION

We are aimed to create an engineering-oriented course with relatively low but strong fundamental prerequirements that allows students to investigate interconnection between formal languages, graph analysis, high performance computing techniques and respective fundamentals. To do so we provide formal-language-centric point of view on graph query engine. The idea of the course is to apply formal languages to design query engines and allows students to touch at least two areas of formal languages application: parsing and graph querying. At the same time, both parsing and formal language constrained path querying (FLPQ) [6] widely used not only in graph databases, but also in software analysis [23]. Moreover, graph databases become popular tool for code analysis [29]. So, we can smoothly combine different areas and demonstrate students deep interconnection between them, and as a result, the course or its' parts may be useful for broad range of students who study different areas of software engineering.

While parsing is a typical application of formal language theory, FLPQ allows us to demonstrate immediate and direct usage of wider range of theoretical results to solve practical tasks. For example, such fundamental results, as closure properties (e.g. Bar-Hillel theorem [5] or the closure of regular languages under intersection) required to restrict graph query language expressive power and to provide respective query evaluation algorithms. Moreover, formal-language-centric view on query evaluation provides a native way to represent infinite result in convenient finite form with well-established tools for analysis: since result of languages intersection is a language, any finite representation (e.g. grammar or automata) of respective language class is a representation of an answer.

Having strong theoretical results, one should put them into user-friendly form to develop query language, and here discussion of language design and implementation, particularly parsing, required. Well-established parsing algorithm, such as LR(k) or LL(k), and actively developed generalized versions (e.g. Generalized LL (GLL) [3]) can be used not only for parsing, but also as a base for CFPQ algorithms [11, 17, 25]. At the same time, combination with graph analysis allows students to get a refreshed view on parsing algorithms and old problems that became actual again in the new context. For example, incrementalization and parallelization of parsing algorithms in the context of huge graph processing, variability of both grammars (queries) and graphs, requires discussion of dynamic graph problems, parallel and distributed graph processing. Another example is a grammar disambiguation that is classical problem for parser developers. But sometimes ambiguous queries faster then disambiguated versions, especially for reachability querying.

High-performance solutions are necessary for real-world graph analysis and one promising way to achieve required performance is an utilization of linear algebra [9]. Also, linear algebra provides a suitable abstraction level that hides lots of technical details but requires application level optimizations techniques to achieve high performance. So, we include a number of FLPQ algorithms that are based operations over boolean matrices and vectors to force students to touch high-performance graph analysis techniques without huge preliminary work. At the same time, native description of these algorithms requires custom semirings introduction that forces students to refresh linear algebra knowledge.

Thus, based on strong fundamentals of formal language theory we cover several applied aspects of graph query engines, such as query language parsing and formal language constrained path queries evaluation algorithms for different classes of languages. Also we touch other important applied and fundamental areas such as graph analysis, linear algebra, high-performance computing.

3 THE COURSE

The course was initially designed for third year bachelor students studying software engineering. But with tiny modification was also used for first year masters in software engineering. Presented version of the course was used in two universities during several years but is still under development.

3.1 Prerequirements

This course is designed for programming engineers so it requires basic engineering skills in Git version control system and GitHub infrastructure, including GitHub actions for CI, code review mechanisms and so on.

We expect an intermediate level in Python programming, including experience with testing frameworks, linters, formatters, dependency managers.

Also, we expect a basic level of linear algebra. Students should freely operates with matrices, vectors, semirings. They should know definitions and properties of matrix-matrix operations such as Kronecker product, elementwise operations, matrix-matrix multiplications, matrix-vector multiplication.

Basic level of graph theory is required, including definitions and properties of directed edge-labeled graphs and relative concepts like path and cycle.

We expect an experience in basic graph analysis algorithms analysis and implementation, such as traversal algorithms (BFS and DFS), path problems related, and reachability problem related algorithms (transitive closure computation, Dijkstra's algorithm, Floyd–Warshall algorithm).

Programming languages theory is also required. Particularly, basic knowledge in formal semantics, type systems, experience in interpreters implementation.

3.2 Learning Outcomes

Upon completing the course, students will be able to explain where and how formal language constrained path querying (including RPQ, CFPQ) can de applied, and how graph databases and static code analysis are interconnected.

Also students will be able to formulate FLPQ problem and explain differences between queries semantics. Additionally they will be able to explain interconnection between formal languages classes and fundamental aspects of FPLQ, including decidability and results representation.

Students will be able to operate with regular languages and its representations, particularly to convert regular expression to finite automata and back, to intersect regular languages, and to implement linear algebra based algorithms for RPQ.

The same for context-free languages and its representations. Students will be able to convert context-free grammar (CFG) to recursive state machine (RSM), to build derivation trees, to implement various algorithms for CFPQ, including linear algebra based and GLL based, and to explain how language and grammar properties interconnect with respective algorithms' properties.

Additionally, students will be able to use linear algebra for graph analysis. Particularly, they will be able to reduce RPQ and CFPQ related problems to boolean linear algebra, to analyze performance of respective algorithms, to explain importance of matrices formats and basic optimization techniques.

Finally, students will be able to develop query languages, including ability to use ANTLR parser generator to create parsers, to craft interpreters that use FLPQ algorithms to evaluate queries, provide type checking, and grammar (query) consistency checking. They will be able to explain basic parsing algorithms, including LL, LR, GLL, to describe differences between them, and limitations of respective tools.

3.3 The Structure

The course is structured with respect to typical formal language theory related course meaning that there is a hierarchy of languages and respective computation machines (Chomsky hierarchy) that often uses to organize materials. Almost all parts combine the theory, respective algorithms and its analysis, including discussion of performance-critical implementation details, possible optimization techniques. Brief content of the parts is provided below.

(1) **Introduction to formal languages** that includes basic definitions such as alphabet, word, language, basic operations over words and languages, including set-theoretic ones. Here we also introduce the classical Chomsky hierarchy of languages and respective computation machines.
(2) **Introduction to graphs and linear algebra** should be tuned with respect to initial level of students such that after this part students will be familiar with basic definitions and algorithms, such as definitions of directed and undirected edge-labelled graphs, paths, and cycles, formulation of reachability and paths problems, and respective algorithms. Additionally, graph representations, including adjacency matrix and its boolean decomposition, and basic graph analysis algorithms in terms of linear algebra, such as transitive closure and multiple-source BFS, should be introduced.
(3) **Introduction to formal language constrained path querying.** This part introduces the formal language constrained path querying (FLPQ) [6] problem statement in the most general form, and describes different semantics including reachability, all-paths, all-pairs, multiple-sources. Fundamental problems, such as infinite number of paths, and, as a result, inability to represent the answer explicitly as a set in some cases, decidability for different languages classes, also discussed here. To bring the gap between two areas we demonstrate that string parsing or recognition problem is a partial case of FLPQ. At the same time we show that there is a number of differences with classical languages processing, such as the fact that language is not fixed in graph querying: while in classical language processing cases we assume that the language is fixed and the string is varying, in graph querying both graph and language can varying. Also we discuss history of FLPQ from Mihalis Yannakakis and Thomas Reps to nowdays, including areas of applications, examples, difference and interconnection between static code analysis and graph databases.
(4) **Regular languages** and ways to specify them, such as regular expression, regular grammars, and finite automata, with transformations between them. We discuss formal properties of languages and show how they relate with RPQ. For example, the fact that regular languages are closed under intersection is the base of RPQ that allows one to consider respective algorithms and query evaluation results representation. Respective algorithms introduced here.
(5) **Context-Free languages** and ways to specify them, including grammars, recursive sate machines (RSMs) and conversions between them. Similarly to regular languages, we introduce formal properties that are important for CFPQ, such as Bar-Hillel theorem that claims context-free languages are closed under intersection with regular ones, and respective algorithms. Also we discuss differences and interconnections between CFPQ and parsing.
(6) **Discussion of well-known old challenges that became actual again.** Parallel and distributed parsing is not so hot problem, but parallel and distributed query processing is. Another problem is a handling changes in input that still actual for parsing but became more complex challenge in the context of graph querying. The main reason is that in FLPQ both graph and grammar can vary while in parsing only string can.
(7) **Query language implementation** where we introduce classical LL(k) and LR(k) parsing algorithms, among with generalized algorithms like GLL, and provide a comparison of respective language classes. We describe typical language processing workflow: lexing, parsing, abstract syntax tree construction, and interpretation (with respect to previously introduced FLPQ algorithms). ANTLR as one of the modern production-quality parser generation tool is introduced.
(8) **Beyond Context-Free languages and Chomsky hierarchy.** In this part language classes that are more expressive than context-free languages introduced. Namely, Multiple Context-Free languages (MCFL), Conjunctive and Boolean languages because these classes of languages are used for static code analysis [8, 30]. For now we discuss only basic definitions and properties without algorithms.

3.4 Exercises

Exercises are focused on FLPQ algorithms implementation and evaluation rather that basic concepts implementation. So we force students to use libraries such as PyFormLang and sciPy to operate with languages, automata, or matrices. The main part of exercises is focused on reachability problem for different classes of languages because the reachability problem is often simpler to implement than path problem. For example it does not require special semirings in linear algebra based algorithms, the boolean one is enough. Different variations, such as all-pairs and multiple source, included.

Almost the all tasks are conceptually interconnected: starting from basic FLPQ algorithms, through its evaluation, simple graph analysis system will be created as a result of the course. As an additional bonus, such scheme limits tasks skipping because some of them are necessary to complete another one.

Brief description of tasks are presented below in order that corresponds to the structure represented in the previous section.

(1) Implementation of all-pairs RPQ with reachability semantics, using Kronecker product, that is the basic finite automaton intersection algorithm.
(2) Implementation of multiple-source linear-algebra-based RPQ algorithm [7].
(3) RPQ algorithms evaluation and performance analysis including analysis of sparse matrix representation formats.
(4) Hellings's CFPQ algorithm [12] implementation that is a pretty simple algorithm without linear algebra and will be used as a baseline for evaluation.
(5) Matrix multiplication based CFPQ algorithm [4] implementation that requires grammar in Chomsky normal form.
(6) Kronecker product based CFPQ algorithm [21] implementation that utilizes RSM for grammar representation.
(7) Implementation of GLL-based CFPQ algorithm [2] that does not use linear algebra and utilize RSM for query representation.
(8) Evaluation performance analysis of implemented CFPQ algorithms with focus on different algorithms comparison.
(9) Implementation of parser for simple predefined graph query language using ANTLR. The language is focused on formal language constrained path querying, not a subset of GQL.
(10) Interpreter of simple graph query language implementation that uses previously implemented algorithms for queries evaluation and provides some additional static query checks.

Exercises can be splitted in subtasks or equipped with additional introductory tasks, for example, with simple challenges aimed to investigate a new library. All exercises also grouped in blocks that are three in total: Regular Languages, Context-Free Languages, and Parsing techniques. This division is formal to introduce tests. While blocks are almost synchronized with lectures, we do not separate introduction block because first two modules (1 and 2) size significantly varies with respect to initial level of students.

3.5 Tests

Each block equipped with test to check related basic knowledge. There is a bunch of questions for each block and each student randomly gets one of them and should provide an answer in 5 minutes. This allows us to check that student has mastered basics related to exercises from the respective block. Tests are used to weight students' score in a block of exercises. The main idea is that if the student can not pass test, then it is highly possible that exercises, even be passed, done with cheats. Examples of questions are presented below.

(1) Convert the given regular expression to finite automaton.
(2) Provide a derivation tree for the given string and grammar.
(3) Convert the given context-free grammar to RSM.

3.6 Environment

The course designed for engineers and include a number of exercises that requires unified coding environment to for all students to simplify work of tutor and mentors. We chose Python programming language as one of the most popular language, particularly among students. Additionally, all the required libraries available in Python and provide easy to use and well-documented interface. Namely, we need libraries for formal languages, sparse linear algebra, parsers creation, and we choose the following ones.

PyFormLang[3] [24] is used to provide basic formal languages concepts such as regular expressions, finite automata, context-free grammars, recursive automata, and operations over them such as automata minimization, regular expression to finite automata conversion, grammar to normal forms conversion and so on.

SciPy[4] is used for sparse boolean linear algebra. It provides different formats for sparse matrices representations, thus allows us to demonstrate correlation between matrix representation format and performance of matrix-based algorithms.

We use ANTLR[5] [22] as a parser generation tool. ANTLR is one of the modern tools for parser development that supports Python as a target language: it can generates parser in Python and appropriate runtime libraries are provided.

Also we use CFPQ-Data[6] for algorithms evaluation. This dataset allows us to provide real-world graphs and queries from such areas as RDF analysis and static code analysis.

Initial project structure with dependencies and checkers configured is provided as a GitHub repository[7] to be forked by students. The repository contains configured actions for CI, supplementary code, placeholders for exercises, functions signatures to implement, and other stuff to minimize preparation to assignments completing. Rye[8] is used for dependencies management.

Automation is done using GitHub actions that trigger on pushes and pull requests, and includes tests execution, code style guide checking. Note that actions should be extended by students to handle parser generation. It allows us to automate control of assignments completing and use code review mechanisms to discuss assignments with students.

We use only an open tests implemented using Pytest framework and they consist of of two types. The first one is a set of ordinary unit tests that check corner cases of algorithms. The second one is a set of property-like tests that use the fact that students should implements algorithms for closely related problems. Thus different algorithms from different assignments should return the same results for randomly generated input. This way we can simplify testing system (no private tests) and avoid implementations fitting.

4 DISCUSSION

Smooth integration of different areas together with clear problems and challenges allows students to be involved in related research during course or right after it. Evaluation of matrix-based CFPQ algorithm, represented by Nikita Mishin, Iaroslav Sokolov et al. in "Evaluation of the Context-Free Path Querying Algorithm Based on Matrix Multiplication" [19] is an improved results of experiments

[3] PyFormLang repository: https://github.com/Aunsiels/pyformlang
[4] SciPy home page: https://scipy.org/
[5] ANTLR (ANother Tool for Language Recognition) home page: https://www.antlr.org/
[6] CFPQ-Data project: https://github.com/FormalLanguageConstrainedPathQuerying/CFPQ_Data
[7] In Russian https://github.com/FormalLanguageConstrainedPathQuerying/formal-lang-course
[8] Rye project home page: https://rye.astral.sh

done as the course exercises. Similarly, Egor Orachev [21], and Ilia Muravev [20] done research as a development of exercises.

Courses that discuss fundamentals of graph querying often limited to RPQ-related classes (e.g. CRPQ), as in "Foundations of Graph Path Query Languages" by Diego Figueira [10], and do not cover other language classes. Some courses combine different techniques for graph processing, including parallel processing models, approximation techniques, path expressions and so on[9,10], rather than focusing on particular class of queries and respective techniques. Query languages discussion is a typical part of data analysis related courses, but with focus on particular databases, query languages and its applications. While SQL fundamentals and optimization techniques often included into such courses, discussion of graph querying is limited[11].

Also we want to highlight some drawbacks and weakness of our course. The first one is that practice with non-linear-algebra-based algorithm for FLPQ and parsing is very limited. But in the context of CFPQ these algorithms are important because they can natively solve all-paths multiple-sources queries, but require special techniques that cannot be natively inferred from linear-algebra-based algorithms. Currently, only GLL-based algorithm implementation is included into exercises, but in restricted reachability version.

Some subtasks of the last part — interpreter implementation — requires special knowledge on programming language theory (e.g. type theory) and experience in programming languages processing. One of possible solution is to make this part more configurable and to introduce less specific subtasks. So, interpreter development can be splitted into basic tasks that do not require advanced programming language theory, and advanced ones.

The proposed course, especially structure of exercises, hides basics of some concepts, such as sparse linear algebra or automata implementation. But this way we learn to use existing libraries that is a useful skill for engineers, and allow students to touch real-world problems and tasks without huge amount of preliminary work.

Regarding environment, one of the drawbacks is that manual control needed to check wether the requested algorithm implemented by students. It is necessary because exercises include several algorithms for the same (or similar) problems, for example four different algorithms for CFPQ, and possible students' cheat is to resubmit single one implementation with slightly changed top level API.

5 CONCLUSION AND FUTURE WORK

We describe the course that brings together formal language theory and graph analysis and involves linear algebra and high performance computing techniques in such a way that smoothly combines different areas with focus on applied graph analysis problems. Note that while this course has been taught for several years now, there is a room for improvements.

One of the important technical improvements is to extend testing system to provide performance testing. For now, performance analysis of the implemented algorithms can be done only in respective tasks on algorithms evaluation and comparison. There is no automatic control on performance of implemented algorithms. So, students not forced to provide not naïve solutions. Moreover, they often provide solutions with trivial performance issues: no early exit in transitive-closure-like procedures, no analysis of sparse matrix format (so, randomly selected format is used) and so on. Additionally, we generate a bunch of property-based tests and these missed optimizations can slow down CI check (and even local ones) to tens of minutes, and also lead to failure on CI for some students.

The next technical challenge is to replace sciPy with python-graphblas[12] in order to enforce studying of specific tools for high-performance graph analysis. It is not clear, whether sciPy should be replaced, or python-graphblas should be provided as an optional alternative for sciPy because sciPy is easier for beginners, but python-graphblas allows one to pay more attention on performance. Also, sciPy provide straightforward control of matrix format, that is important for performance analysis, while in python-graphblas such a control is quiet tricky.

Also we want to show some ways to extend the course. First of all, more algorithms and related tasks can be added. For example, multiple sources version of linear algebra based algorithm for CFPQ, proposed by Arseniy Terekhov et al [28]. Another candidates to be added are path problem related linear algebra based algorithms for both RPQ and CFPQ. All these algorithms allows students to touch new types of problems and investigate linear algebra based approach to graph analysis deeper. But introduction of some of these algorithms is related to migration to python-graphblas because sciPy is not enough to implement them: some specific operations and ways to custom semirings specification and utilization is not provided in this library.

Important way to extend the course is to add more materials on languages beyond context-free, such as multiple context-free, boolean and conjunctive languages. These languages play important role in static analysis, and deeper studying is important to realize boundaries of expressivity power of graph query languages. Also, discussion of these languages leads to nontrivial decidability analysis for FLPQ-related problems. As a result, it leads to introduction of approximation algorithms that is an important class of algorithms not covered by current version of the course. Not only theory, but also respective algorithmic exercises should be added.

Data for algorithms evaluation also should be extended to represent more different areas of FLPQ applications. For example, for CFPQ it is necessary to add biological data [26], data provenance related graphs and queries [18], more code analysis related data.

All of the above leads to big number of exercises and one of possible solution is to make significant number of them optional. But finally it should be possible to configure consistent subset of exercises in terms that even subset of tasks allows student to create self-contained application for graph analysis. Globally we want to achieve high flexibility of materials such that we would be able to use specific submodules in other related courses. For example, in courses on formal languages, or static code analysis.

[9]The university of Edinburgh, Querying Large Graphs course: http://www.drps.ed.ac.uk/16-17/dpt/cxinfr11121.htm.

[10]University of Buffalo, Data Models and Query Languages course: https://catalogs.buffalo.edu/preview_course_nopop.php?catoid=1&coid=1061.

[11]Charles University, Query Languages course (https://www.ksi.mff.cuni.cz/~svoboda/courses/241-NDBI049/), Advanced Database Systems course(https://www.ksi.mff.cuni.cz/~svoboda/courses/241-NIE-PDB/).

[12]Python wrapper for SuiteSparse:GraphBLAS: https://github.com/python-graphblas/python-graphblas.

REFERENCES

[1] 2013. *SPARQL 1.1 Query Language.* Technical Report. W3C. http://www.w3.org/TR/sparql11-query

[2] Vadim Abzalov, Vlada Pogozhelskaya, Vladimir Kutuev, and Semyon Grigorev. 2023. GLL-based Context-Free Path Querying for Neo4j. arXiv:2312.11925 [cs.DB] https://arxiv.org/abs/2312.11925

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

[4] 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)*. Association for Computing Machinery, New York, NY, USA, Article 5, 10 pages. https://doi.org/10.1145/3210259.3210264

[5] Yehoshua Bar-Hillel, M. Perles, and E. Shamir. 1961. On Formal Properties of Simple Phrase Structure Grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung* 14 (1961), 143–172. Reprinted in Y. Bar-Hillel. (1964). *Language and Information: Selected Essays on their Theory and Application*, Addison-Wesley 1964, 116–150.

[6] Chris Barrett, Riko Jacob, and Madhav Marathe. 2000. Formal-Language-Constrained Path Problems. *SIAM J. Comput.* 30, 3 (2000), 809–837. https://doi.org/10.1137/S0097539798337716 arXiv:https://doi.org/10.1137/S0097539798337716

[7] Georgiy Belyanin and Semyon Grigoriev. 2024. Single-Source Regular Path Querying in Terms of Linear Algebra. arXiv:2412.10287 [cs.DS] https://arxiv.org/abs/2412.10287

[8] Giovanna Kobus Conrado, Adam Husted Kjelstrøm, Jaco van de Pol, and Andreas Pavlogiannis. 2025. Program Analysis via Multiple Context Free Language Reachability. *Proc. ACM Program. Lang.* 9, POPL, Article 18 (Jan. 2025), 30 pages. https://doi.org/10.1145/3704854

[9] Timothy A. Davis. 2019. Algorithm 1000: SuiteSparse:GraphBLAS: Graph Algorithms in the Language of Sparse Linear Algebra. *ACM Trans. Math. Softw.* 45, 4, Article 44 (dec 2019), 25 pages. https://doi.org/10.1145/3322125

[10] Diego Figueira. 2022. *Foundations of Graph Path Query Languages*. Springer International Publishing, Cham, 1–21. https://doi.org/10.1007/978-3-030-95481-9_1

[11] 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)*. Association for Computing Machinery, New York, NY, USA, Article 10, 7 pages. https://doi.org/10.1145/3166094.3166104

[12] Jelle Hellings. 2014. Conjunctive Context-Free Path Queries.. In *ICDT*, Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy (Eds.). OpenProceedings.org, 119–130. http://dblp.uni-trier.de/db/conf/icdt/icdt2014.html#Hellings14

[13] Jelle Hellings. 2025. Explaining results of path queries on graphs: Single-path results for context-free path queries. *Information Systems* 128 (2025), 102475. https://doi.org/10.1016/j.is.2024.102475

[14] ISO/IEC 39075:2024 2024. *Information technology – Database languages – GQL*. Standard. International Organization for Standardization, Geneva, CH. https://www.iso.org/standard/76120.html

[15] 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)*. Association for Computing Machinery, New York, NY, USA, 121–132. https://doi.org/10.1145/3335783.3335791

[16] 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

[17] Ciro M. Medeiros, Martin A. Musicante, and Umberto S. Costa. 2019. LL-based query answering over RDF databases. *Journal of Computer Languages* 51 (2019), 75–87. https://doi.org/10.1016/j.cola.2019.02.002

[18] Hui Miao and Amol Deshpande. 2019. Understanding Data Science Lifecycle Provenance via Graph Segmentation and Summarization. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 1710–1713. https://doi.org/10.1109/ICDE.2019.00179

[19] Nikita Mishin, Iaroslav Sokolov, Egor Spirin, Vladimir Kutuev, Egor Nemchinov, Sergey Gorbatyuk, and Semyon Grigorev. 2019. Evaluation of the Context-Free Path Querying Algorithm Based on Matrix Multiplication. In *Proceedings of the 2nd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)* (Amsterdam, Netherlands) *(GRADES-NDA'19)*. Association for Computing Machinery, New York, NY, USA, Article 12, 5 pages. https://doi.org/10.1145/3327964.3328503

[20] Ilia Muravev. 2024. Optimization of the Context-Free Language Reachability Matrix-Based Algorithm. arXiv:2401.11029 [cs.PL] https://arxiv.org/abs/2401.11029

[21] Egor Orachev, Ilya Epelbaum, Rustam Azimov, and Semyon Grigorev. 2020. Context-Free Path Querying by Kronecker Product. In *Advances in Databases and Information Systems*, Jérôme Darmont, Boris Novikov, and Robert Wrembel (Eds.). Springer International Publishing, Cham, 49–59.

[22] Terence Parr. 2013. *The Definitive ANTLR 4 Reference* (2nd ed.). Pragmatic Bookshelf.

[23] Andreas Pavlogiannis. 2023. CFL/Dyck Reachability: An Algorithmic Perspective. *ACM SIGLOG News* 9, 4 (Feb. 2023), 5–25. https://doi.org/10.1145/3583660.3583664

[24] Julien Romero. 2021. Pyformlang: An Educational Library for Formal Language Manipulation. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 576–582. https://doi.org/10.1145/3408877.3432464

[25] 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.

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

[27] 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

[28] 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 *EDBT*. 487–492.

[29] Raoul-Gabriel Urma and Alan Mycroft. 2015. Source-code queries with graph databases—with application to programming language usage and evolution. *Science of Computer Programming* 97 (2015), 127–134. https://doi.org/10.1016/j.scico.2013.11.010 Special Issue on New Ideas and Emerging Results in Understanding Software.

[30] Qirun Zhang and Zhendong Su. 2017. Context-sensitive data-dependence analysis via linear conjunctive language reachability. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages* (Paris, France) *(POPL '17)*. Association for Computing Machinery, New York, NY, USA, 344–358. https://doi.org/10.1145/3009837.3009848