Teach Formal Languages Together With Graph Querying for Great Power

Semyon Grigorev
Institute for Clarity in Documentation
Dublin, Ohio, USA
!!!@corporation.com

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

Ekaterina Shemetova
The Thørväld Group
Hekla, Iceland
larst@affiliation.org

ABSTRACT

A clear and well-documented LaTeX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the "acmart" document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

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

ACM Reference Format:
Semyon Grigorev, Egor Orachev, Vadim Abzalov, Rustam Azimov, and Ekaterina Shemetova. 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, 5 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Bridging the gap between fundamental disciplines and applications is one of the important problems of education in software engineering. Theory without applications — bad idea. At the same time, real-world problems requires huge amount of preparatory work before the !!!

Data analysis involves broad range of !!! . Graph analysis. Application of fundamental ones. Motivation for students which focused on applied areas to study fundamental disciplines

Formal languages is a basis for graph query languages. GQL ISO, Cypher, SPARQL, W3C

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

Integration of formal language theory + HPC + linear algebra + graphs to show interconnection between different areas. How concepts and ideas from one area can be applied in another one with significant effect.

For engineers, so it is important to equip theory with practical tasks to !!!motivate!!!.

Formal languages Closure properties. Regular languages are closed under intersection. Any finite representation of regular language is s representation of an answer.

Historically, programming languages and natural language processing form an area of formal languages theory application

In this work we describe our experience on such a course for third-year bachelor students. Software engineering. Structure of this work:

- Motivation: why we do exactly what we do and why we do it exactly such a way.
- Course structure: technical environment, Exercises and how they are related to !!!
- Discussion of existing course. Comparison with other related courses.
- Conclusion and future work.

2 MOTIVATION

We are aimed to create an applied course with relatively low prerequirements and strong fundamental !!! which allows students to touch !!!! !!! !!!

Why formal language constrained path querying. Formal languages is not only parsing. It allows Smoothly combine different areas Main: data analysis and formal languages. FLPQ part of the GQL ISO standard.

Immediate and direct usage of theory to solve practical tasks: Closure properties. Languages intersection. Results representation — languages representation. Complexity analysis.

Parsing algorithms and new old problems: incremental parsing, parallel parsing. In wew context (amount of data to process, variability of grammars, not only graphs etc). Disambiguation of grammar. Ambiguity not evil: sometimes ambiguous queries faster. Especially for querying. But not parsing.

Linear algebra provides a suitable abstraction level. Application level independent optimizations !!! HPC, matrices, easy to abstract. One can easily replace one library with another without significant algorithm changes (but technical effort for different API-s). As a

result, easy to apply for real-world data. Which also helps to solve a motivation problem (small synthetic tasks looks disconnected with real-world).

Selected algorithms can be expressed in terms of boolean matrices and vectors: easy to start. No need to create custom semirings. At the same time, native description of these algorithms requires custom semirings introduction. Conversion techniques. Weaker than semirings structure.

Language design and implementation. Tooling for language implementation. Parsing algorithm (LR, LL). GLL. ANTLR is not a magic.

Relations with other areas such as graph theory, dynamic graph problems, algebra.

Touch open problems (truly subcubic CFPQ) and motivate to study them (and respective fundamentals eg fine-grained complexity). Performance on real-life problems.

3 THE COURSE

The course was initially designed for third year bachelor students studying software engineering. But with tiny modification used also for first year master students also studying software engineering.

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.

Not only linear algebra, but classical algorithms, parsing techniques.

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.

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 implementation and its fundamental properties, 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 knowledges in formal semantics, type systems, experience in interpreters implementation.

3.2 Learning Outcomes

Upon completing this course, students will be able to ...

(1) Formal language constrained path querying, including RPQ, and CFPQ
(2) Basic theory on regular languages
(3) Basic theory on context-free languages
(4) Algorithms for RPQ
(5) Algorithms for CFPQ
(6) Graph analysis with linear algebra (GraphBLAS)
(7) Applications for RPQ, CFPQ
(8) Parsing algorithms: LL, LR, GLL
(9) Use ANTLR parser generator to create parsers., interpreters, etc.

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 the all parts combines algorithms and respective theory. The following parts

(1) **Introduction to formal languages** that includes basic definitions such as *alphabet*, *word*, *language*, basic operations over words and languages. Also we introduce the hierarchy of languages in tis block.
(2) **Introduction to graphs and linear algebra** block that should be tuned with respect to initial level of students. Path problems, reachability problems. Respective algorithms. Matrices, boolean decomposition, edge-labelled graphs, directed, undirected. Transitive closure in terms of linear algebra, BFS in terms of linear algebra. Kronecker product and graph product.
(3) **Introduction to formal language constrained path querying.** This block introduces the formal language constrained path querying (FLPQ) [**?**] problem statement in the most general form, and describes different semantics including reachability, all-paths, all-pairs, multiple-sources. History (Form Thomas Reps and Mihalis Yannakakis) and applications (examples, differences in static code analysis). Fundamental problems, such as infinite number of paths (so, inability to represent the answer explicitly as a set in some cases), desidability for different languages classes, discussed in this block. Also here we discuss 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.
(4) **Regular languages** and the ways to specify them, such as regular expression, regular grammars, and finite automata, with transformations between them. Closure properties. Algorithms for regular path queries. All pairs that exactly the languages intersection. Multiple source.
(5) **Context-Free languages**. Grammars, RSM-s, closure properties. Derivation tree and SPPF as a way to represent paths. Context-Free path queries. Closure properties. Language representations (grammars, RSMs). Bar-Hillel theorem. Complexity. Partial cases (Bradford, Pavlogianis, etc).

(6) **Discussion of well-known old challenges that becomes actual again.** Parallel and distributed processing. Basic ideas from graphs and matrices to dynamic.
(7) **Query language design and implementation.** Parsing algorithms. Parsing techniques. ANTLR. Query language design on the top of previously implemented algorithms.
(8) **Beyond Context-Free languages and Chomsky hierarchy.** Multiple Context-Free languages (MCFL). Conjunctive and boolean languages. These classes of languages are used for static code analysis [**?**].

Modules grouped in blocks that are three in total, all listed below.

(1) Regular Languages
(2) Context-free Languages
(3) Parsing techniques

This division is formal to introduce tests. We do not introduce introduction block because first two modules (1 and 2) size significantly varying with respect to initial level of students. So it includes in the first block for now. 1, 3

3.4 Exercises

Exercises are oriented to algorithms implementation and evaluation, not theory. The main part of exercises is focused on reachability problem for different classes of languages as on a simplest one (in comparison with path problem). Different variations, including all-pairs and multiple source. This problem can be naturally described using boolean matrices.

We are focused on FLPQ algorithms implementation rather that basic concepts implementation. So we allow students to use libraries such as PyFormLang and sciPy to operate with languages, automata, or matrices.

We argue that algorithms and approaches (but not libraries) applicable for real-world problems. Almost the all tasks are conceptually connected: starting from basics, through algorithms, to simple graph analysis system. Interconnection between parts. Inability to skip tasks

Tasks are presented below.

(1) To implement all-pairs RPQ with reachability semantics, using Kronecker product. Basic FA intersection algorithm. (commutativity of Kronecker product)
(2) To implement RPQ multiple-source BFS-based. Another algorithm for automata intersection. Different versions of multiple-source BFS problem (set-to-set, etc).
(3) RPQ evaluation and performance analysis. Different matrices formats etc. All-pairs vs multiple sources. Advanced: GPU or GraphBLAS. Easy switch.
(4) To implement CFPQ Hellings. Pretty simple algorithm without linear algebra. Baseline for comparison with other algorithms.
(5) To implement CFPQ matrices (associativity and commutativity of operations), normal form for grammar.
(6) To implement CFPQ tensors (unification of RPQ and CFPQ), RSF introduction.
(7) To implement GLL-based CFPQ algorithm with reachability semantics. RSM for query representation.
(8) To evaluate implemented CFPQ algorithm and to provide performance analysis. Different algorithms comparison. Advanced: GPU or GraphBLAS
(9) Query language design. Introduction of GQL and other real-world languages. Query language implementation. Graph query language introduction. GQL. Language design principles and problems.
(10) Interpreter. Other language implementation related tasks. Type inference.

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 regarding blocks Regular, Context-Free, Parsing techniques.

3.5 Tests

The course equipped with short tests on the main topics to check basic knowledge. Three tests in total: one test before each block.

Each block contains a set of questions on basic concepts, theory, algorithms. Examples of questions are presented below.

(1) To show whether Kronecker product is commutative operation.
(2) To convert simple regular expression to finite automaton.
(3) To provide a derivation tree for the given string and grammar.
(4) To convert context-free grammar to recursive state machine.
(5) To provide an example of *k-MCFL(r)* language for the given k and r.

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 almost the all of which were used to complete exercises from the respective block.

Tests used to tune tasks. Main idea is that is the student can not pass test, then highly possible that exercises, even be passed, done with cheats.

3.6 Environment

The course is applied and include a number of programming-related exercises that requires respective environment to unify settings for all students and to simplify work of tutor and mentors.

We choose Python programming language as one of the most popular language, particularly among students. Additionally, there are all required libraries implemented 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[1] [4] is ued 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[2] is used for sparse boolean linear algebra. It provides different formats for sparse matrices representations, thus allows us

[1] https://github.com/Aunsiels/pyformlang
[2] https://scipy.org/

to demonstrate correlation between matrix representation format and performance of matrix-based algorithms.

We use ANTLR[3] [3] as a parser generation tool. ANTLR is one of the modern tools for parsers 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[4] as a collection of graphs and queries for algorithms evaluation. This dataset allows us to provide real-world graph 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[5] 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[6] 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, parser generation related actions. 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 testing 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 form different assignments should return the same results for randomly generated input. This way we can simplify testing system (no private tests) and avoid algorithms fitting.

4 DISCUSSION

Motivation to study formal languages, refresh linear algebra, HPC.... Why matrices: pretty simple ideas and algorithms, rather than LR, hides implementation complexity, clear abstraction (it is easy to realize that one library of linear algebra operations can be replaced with another one). More over, students can do it yourself.

Simple formulation of engineering tasks, problems, challenges regarding performance 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" [2] is an improved results of experiments done as the course exercise.

Orachev. Muraviev.

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, respectively) is very limited. Different algorithms, based on GLR, GLL and other parsing algorithms. Only GLL, but reachability, not paths. These algorithms are powerful (can natively solve all-paths queries), but require special techniques that cannot be natively inferred from linear-algebra-based algorithms. Especially high-performance solution. Looks like advanced block. Last block requires special knowledge. Can de optional. Ore replaced with.

Proposed structure hides basics of some concepts. For example, sparse linear algebra. 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.

Manual control needed to check wether the requested algorithm implemented. Few algorithms for the same (or similar) problems: it is possible to resubmit single one implementation.

Last part requires programming languages theory. May be omitted. Other blocks can be used independently in other courses.

Other courses [1]

Advanced SQL, Graph databases, GQL,

5 CONCLUSION AND FUTURE WORK

We describe the course that aimed to graph query engine development !!! Formal languages anf graph theory. 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 nïve 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.

The next technical challenge is to replace sciPy with python-graphblas[7] 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.

More algorithms. Multiple sources versions of algorithms for CFL-r (linear-algebra based). But it should be simplified first. A bit more concepts required.

Custom semiring and path problems. Related to migration to python-graphnas.

More non-matrix-based algorithms. Comparison.

More Beyond context-free (MCFG). Static analysis. Matrices. Practical tasks.

More data for evaluation form wider range of applications. Biological data, data provenance, more code analysis.

All above leads to more optional tasks. But all of them consistent in terms tah allows student to create self-contained application for graph analysis.

Decidability problems, so on.

Flexibility. To create a set of reuseable modules for !!! cources for fromal languegaes, graph querying etc.

[3] ANTLR (ANother Tool for Language Recognition) home page: https://www.antlr.org/
[4] https://github.com/FormalLanguageConstrainedPathQuerying/CFPQ_Data
[5] In Russian https://github.com/FormalLanguageConstrainedPathQuerying/formal-lang-course
[6] https://rye.astral.sh
[7] Python wrapper for SuiteSparse:GraphBLAS: https://github.com/python-graphblas/python-graphblas.

To build an advanced course on data analysis. Materials (in Russian)[8]

REFERENCES

[1] 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

[2] 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

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

[4] 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

[8] !!!