Bring Graph Querying and Formal Language Theory Together 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. Bring Graph Querying and Formal Language Theory Together for Great Power. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 4 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,

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!!!.

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.
- 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

Third year bachelor students. Software engineering. To design query engines.

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

3.1 Prerequirements

Git/github/CI

Python programming (intermediate level), testing frameworks, tets creation.

Linear algebra basics: matrices, vectors, semirings, matrix-matrix operations such as Kronecker, elementwise operations, matrix-matrix multiplications, matrix-vector multiplication.

Basic graph theory: directed edge-labeled graphs, path problems, traversals (BFS, DFS), reachability problem and respective algorithms, transitive closure.

Programming languages theory: type systems, interpreters, so on.

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 course. Important aspects of formal language theory The following parts

(1) Introduction to formal languages that includes basic definitions such as *alphabet*, *word*, *language*, basic operations,
(2) Introduction to graphs and linear algebra. Path problems, reachability problems. Respective algorithms. Matrices, edge-labelled graphs, directed, undirected. Transitive closure in terms of linear algebra, BFS in terms of linear algebra. Kronecker product and graph product. With respect to initial level of students.
(3) Introduction to formal language constrained path querying. problem statement (FLPQ), different semantics (reachability, paths, all-pairs, multiple-sources). Problems (infinite number of paths, representation of such a set, desidability for different languages classes). String to graph generalization. History (Form Thomas Reps and Mihalis Yannakakis) and applications (examples, differences in static code analysis). Differences with languages processing (grammar not fixed, input not fixed).
(4) Regular languages. Closure properties, languages representations (regular expression, finite automata). Regular path queries
(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) New old problems. Parallel and distributed processing. Basic ideas from graphs and matrices to dynamic.
(7) Query language design and implementation. On the top of previously implemented algorithms. ANTLR. Parsing algorithms. Parsing techniques.
(8) Beyond Context-Free languages and Chomsky hierarchy. Multiple Context-Free languages (MCFL). Conjunctive and boolean languages. Static code analysis.

3.4 Exercises

Focused on reachability problem as on a simplest one. Different variations, including all-pairs and multiple source. Practical part is oriented to engineering, not theory. Algorithms implementation.: use libs. No basic concepts reimplementation. PyFormLang. This problem can be naturally described using boolean matrices. Applicable for real-world problems. Almost the all tasks are conceptually connected: starting from basics, through algorithms, to simple graph analysis system.

(1) All-pairs RPQ, tensors. Basic FA intersection algorithm. (commutativity of Kronecker product)
(2) 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) CFPQ Hellings. Pretty simple algorithm without linear algebra. Baseline for comparison with other algorithms.
(5) CFPQ matrices (associativity and commutativity of operations), normal form for grammar.
(6) CFPQ tensors (unification of RPQ and CFPQ), RSF introduction.
(7) CFPQ GLL.

(8) CFPQ evaluation and performance analysis. Different algorithms comparison. Advanced: GPU or GraphBLAS
(9) Query language design. Introduction of GQL and other real-world languages.
(10) Query language implementation. Graph query language introduction. GQL. Language design principles and problems. Interpreter. Other language implementation related tasks.

Exercises can be splitted in subtasks or equipped with additional introductory tasks, for example, with simple tasks aimed to investigate a new library.

3.5 Tests

engineers, so less theorems, more practice.

Short tests on the main topics to check basic knowledge. Each test before block.

Three blocks:

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

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

(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 was used to complete exercises from the block.

3.6 Environment

Applied course, so programming tasks, needs automatization.

Python programming language. PyFormLang[1] [3] is ued to provide basics 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 and so on. sciPy[2] is used for sparse boolean linear algebra. CFPQ-Data[3] as a collection of graphs and queries for algorithms evaluation. ANTLR with Python target for parsers creation.

Initial structure as a repository to fork is provided. Placeholders, functions signatures,supplementary code.

Automation is done using GitHub actions and includes tests execution, code style guide checking, !!!!

We provide an open tests 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: algorithms should return the same results for random input. Previously created algorithms to tests tne new one.

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

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. First is that practice with non-linear-algebra-based algorithm for FLPQ (and parsing, respectively) is 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.

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]

5 CONCLUSION AND FUTURE WORK

We propose a course on !!!

Possible directions

Extend testing system: performance testing. For now performance analysis only in respective tasks. No automatic control on performance of implemented algorithm. Nïve solutions with performance issues: no early exit, no analysis of sparse matrix format.

Move it to python-graphblas[4]

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

More optional tasks.

More non-matrix-based algorithms.

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

Decidability problems, so on.

To build an advanced course on data analysis. Materials (in Russian)[5]

[4]https://github.com/python-graphblas/python-graphblas.
[5]!!!

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] 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