Bring Graph Querying, Formal Language Theory, and Linear Algebra Together to Make It Better

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, Formal Language Theory, and Linear Algebra Together to Make It Better. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 3 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. 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,

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!!!. 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 !!! 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).

3 THE COURSE

Third year bachelor students. Software engineering.

3.1 Prerequirements

Git/github/CI

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

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

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

3.2 Learning Outcomes

(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) Experience with ANTLR, interpreters, etc.

3.3 Structure

The course is structured with respect to typical FL. Important aspects of formal language theory

(1) Introduction to formal languages, problem statement (FLPQ), different semantics (reachability, paths). Problems (infinite number of paths, representation of such a set, desidability for different languages classes). String to graph generalization. History (Reps, Yannacacis) and applications (examples, differences in static code analysis). Differences with languages processing (grammar not fixed, input not fixed).
(2) Introduction to graphs and linear algebra. Matrices, edge-labelled graphs, directed, undirected. With respect to initial level of students.
(3) Regular languages, queries. Closure properties, languages representations (regular expression, finite automata).
(4) Context-Free languages, queries. Closure properties. Language representations (grammars, RSMs). Bar-Hillel theorem. Complexity. Partial cases (Bradford, Pavlogianis, etc).
(5) New old problems. Basic ideas from graphs and matrices to dynamic.
(6) Query language design and implementation. On the top of previously implemented algorithms. ANTLR. Parsing algorithms. Parsing techniques.
(7) Beyond Context-free and Chomsky hierarchy. MCFL.

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. PyFormLang. This problem can be naturally described using boolean matrices. Applicable for real-world problems. Starting form basics, throu 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.

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

3.6 Environment

Automatization. Python programming language, PyFormLang[1] [1] for basis forma languages concepts: regular expressions, finite automata, context-free grammars, recursive automata, and operations over them (automata minimization, regular expression to finite

[1] !!!

automata conversion) , CFPQ-Data[2] as a collection of graphs and queries, sciPy[3] for sparse linear algebra, pyGrpahBlas[4] for advanced sparse linear algebra, pyCubool[5] [**?**], Google colab.

Testing (unit tests), code style guide checkers, automation using GitHub actions.

Open tests. Property-like: algorithms should return the same results for random input. Previously created algorithms to tests tne new one. The classical one: just unit tests to check corner cases.

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.

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.

5 CONCLUSION AND FUTURE WORK

We propose a course on !!!

Possible directions

Move it to python-graphblas[6]

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

Optional tasks.

Non matrix based algorithms.

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

Decidability problems, so on.

To build an advanced course on data analysis. Materials (in Russian)[7]

REFERENCES

[1] 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

[2] !!!

[3] !!!

[4] !!!

[5] !!!

[6] https://github.com/python-graphblas/python-graphblas.

[7] !!!