

Санкт-Петербургский государственный университет

Кафедра Системного программирования

Свитков Сергей Андреевич

Реализация поиска путей с
КС-ограничениями в рамках библиотеки
УС.QuickGraph

Курсовая работа

Научный руководитель:
ст. преп, к. ф-м. н. Григорьев С. В.

Санкт-Петербург
2017

Оглавление

Введение	3
1. Постановка задачи	5
2. Обзор предметной области	6
2.1. Related works	6
2.2. YaccConstructor	7
2.3. YC.QuickGraph	8
3. Реализация	10
3.1. Архитектура решения	10
3.2. Реализация библиотеки	11
3.3. SPPF	12
4. Заключение	13
Список литературы	14

Введение

Разнообразные данные могут быть представлены в виде ориентированного графа с помеченными ребрами. Такая модель имеет широкую область применения и используется в биоинформатике, при реализации графовых баз данных, в социальных исследованиях, semantic web. Работу с такими данными, как правило, выполняют с помощью языков запросов.

Существует множество таких языков, применительно к графовым базам данных наиболее известными являются Gremlin[15], Cypher[10], используемые в графовых базах данных Titan[16] и Neo4J [9] соответственно. Однако, эти языки являются регулярными, а значит, не могут применяться, например, при разборе генеалогического дерева часто возникает задача поиска потомков общего предка — поиск последовательностей вида $parent^n child^n$. Но строки такого вида не выводимы в языках, задаваемых регулярными грамматиками, однако, могут быть выведены в языке, заданном контекстно-свободной (далее — КС) грамматикой.

Существуют работы, предлагающие различные подходы к реализации языков запросов и процессу исполнения запроса, например [14], [5], [2], [1], [6]. Большая часть перечисленных работ представляет теоретические сведения о реализации, а предлагающие практическую реализацию имеют ограниченный функционал или очень узкую специализацию. Однако класс задач, решение которых возможно с помощью КС-запросов, является обширным, поэтому хотелось бы реализовать инструмент, который имел бы широкую область применения на практике. Для достижения этого необходимо поддерживать различные форматы представления результата запроса, поэтому было принято решение реализовать библиотеку для выполнения КС-запросов к графам, позволяющую представить результат исполнения запроса в нескольких форматах.

Для реализации была выбрана технология .NET. Для неё существует ряд библиотек для работы с графами, например GraphSharp [8],

Automatic Graph Layout [7], но наиболее известной является QuickGraph [19], работа над которой прекращена в 2011 году. В лаборатории языковых инструментов JetBrains с 2015 года ведется разработка и поддержка библиотеки YC.QuickGraph [17], за основу которой была взята QuickGraph. Было принято решение использовать её при реализации для работы с графами. В качестве алгоритма для синтаксического анализа графов используется алгоритм синтаксического анализа регулярной аппроксимации на основе GLL[13], который в рамках работы [26] был реализован и интегрирован в YaccConstructor [18], [20], [23], [22], [23], [21], [24], [25] — исследовательский проект лаборатории языковых инструментов JetBrains, представляющий собой набор инструментов для решения различных задач синтаксического и лексического анализа, реализованный для платформы .NET.

1. Постановка задачи

Целью работы является реализация библиотеки для выполнения КС-запросов к графам с использованием .NET, YC.QuickGraph — в качестве библиотеки для представления графов и YaccConstructor — для решения задач синтаксического анализа. Результат работы позволяет выполнять КС-запросы к ориентированным графам с помеченными ребрами и представлять результат в виде подграфа, множества путей, кратчайшего пути, КС-отношения ($R = (N, n, m)$, где N — нетерминал, из которого выводим путь из вершины n в m).

Для достижения данной цели необходимо решить следующие задачи:

- разработать алгоритм исполнения запроса с КС-ограничениями;
- спроектировать удобный для пользователя интерфейс задания запросов;
- реализовать расширение библиотеки YC.QuickGraph;
- опубликовать результат в виде NuGet-пакета.

2. Обзор предметной области

2.1. Related works

В этой секции будут рассмотрены работы, посвященные реализации инструментов для исполнения запросов.

Conjunctive Context-Free Path Queries

В данной работе рассматривается построение обобщения существующего регулярного языка запросов к графам с помеченными ребрами CRPQ до КС-языка CCFPQ. Расширение позволяет использовать КС-грамматики вместо регулярных выражений для поиска путей в графе. Предлагаемый в статье алгоритм использует СΥК для синтаксического анализа графов. Результатом исполнения запроса является КС-отношение R . К минусам данной работы можно отнести отсутствие практической реализации и возможность представления результата запроса лишь в одном формате.

Subgraph Queries by Context-free Grammars

Данная работа рассматривает вопрос о применении КС-запросов в различных задачах биоинформатики. Предложенный в статье подход подразумевает поиск связного подграфа, порождаемого множеством путей, строки из меток на которых выводимы из задаваемой в качестве запроса КС-грамматики. Для синтаксического анализа используется Earley parser [4]. Авторами было проведено тестирование алгоритма, предлагаемого в статье, как на случайно сгенерированных, так и на реальных данных. Эксперименты, поставленные на реальных данных, показали, что для графа с максимальной длиной пути, равной 8 вершинам, время работы алгоритма может достигать 250 секунд. Минусами данной работы являются единственный формат представления результата и слишком наивный алгоритм, предлагаемый авторами.

Ослабленный синтаксический анализ динамически формируемых выражений на основе алгоритма GLL

В данной работе описыван алгоритм для синтаксического анализа регулярной синтаксической аппроксимации на основе алгоритма GLL, позволяющий проверять выводимость последовательности меток на ребрах графа, представленных токенами (целочисленными значениями), в задаваемой грамматике. Предложенный в работе алгоритм позволяет обрабатывать входные данные большого размера и может быть использован, например, при поиске подпоследовательностей в метагеномных сборках. Так же была доказана корректность и завершаемость. Следует отметить, что результатом работы алгоритма является лес разбора, представляемый в виде SPPF[12], который можно отобразить в нужный формат вывода. Учитывая наличие реализации алгоритма для платформы .NET в проекте YaccConstructor, было принято решение использовать результаты автора при реализации.

2.2. YaccConstructor

YaccConstructor — исследовательский проект лаборатории языковых инструментов JetBrains, применимый для исследования и решения различных задач синтаксического и лексического анализа. Проект имеет одноименный инструмент с открытыми исходниками, который включает в себя большое количество компонентов, таких, как язык спецификаций грамматик YARD, алгоритмы для преобразования грамматик, алгоритмы для синтаксического анализа графов и др. Большая часть компонент проекта YaccConstructor реализована для платформы .NET на языке F#. Поскольку проект имеет модульную архитектуру (рис.1), его компоненты могут быть использованы независимо.

Более подробно рассмотрим YARD, GLLAbstractParser и GLLGenerator. YARD — язык спецификаций грамматик, позволяющий задавать различные типы грамматик (атрибутные, в нормальной форме Бэкуса-Наура, КС и др.). Так как в рамках данной работы грамматика является запросом, то для его задания будем использовать YARD. GLL

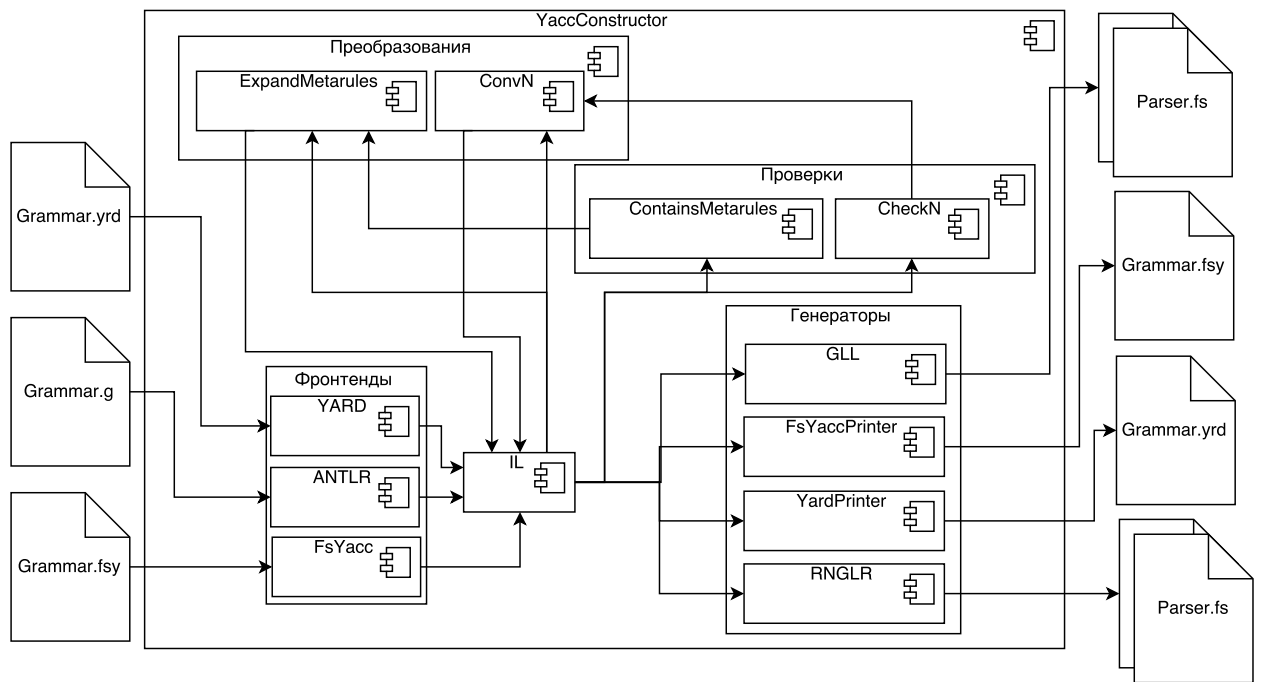


Рис. 1: Архитектура проекта YaccConstructor, заимствована из работы [22]

— алгоритм синтаксического анализа, поддерживающий все типы КС-грамматик (в том числе и лево-рекурсивные), кроме того, имеющий асимптотику $O(n)$ для однозначных грамматик и $O(n^3)$ в худшем случае. GLLGenerator позволяет генерировать парсеры для заданных грамматик (GLLParserSource). GLLAbstractParser позволяет получить из заданного графа и GLLParserSource сжатый лес разбора — SPPF. SPPF содержит информацию о всех деревьях вывода, что позволяет с помощью различных функций получить результат разбора в нужном формате: в виде КС-отношения, подграфа, множества путей или кратчайшего пути.

2.3. YC.QuickGraph

YC.QuickGraph — проект лаборатории языковых инструментов JetBrains, представляющий собой библиотеку для работы с графами на платформе .NET. YC.QuickGraph не является разработанным с нуля проектом, за его основу взята библиотека QuickGraph [19], работа над которой была прекращена в 2011 году. YC.QuickGraph имеет средства для пред-

ставления графов, различные алгоритмы для них (DFS, BFS, поиск кратчайшего пути и др.). Функционал данной библиотеки планируется использовать для задания пользователем графа и построения вывода в нужном формате.

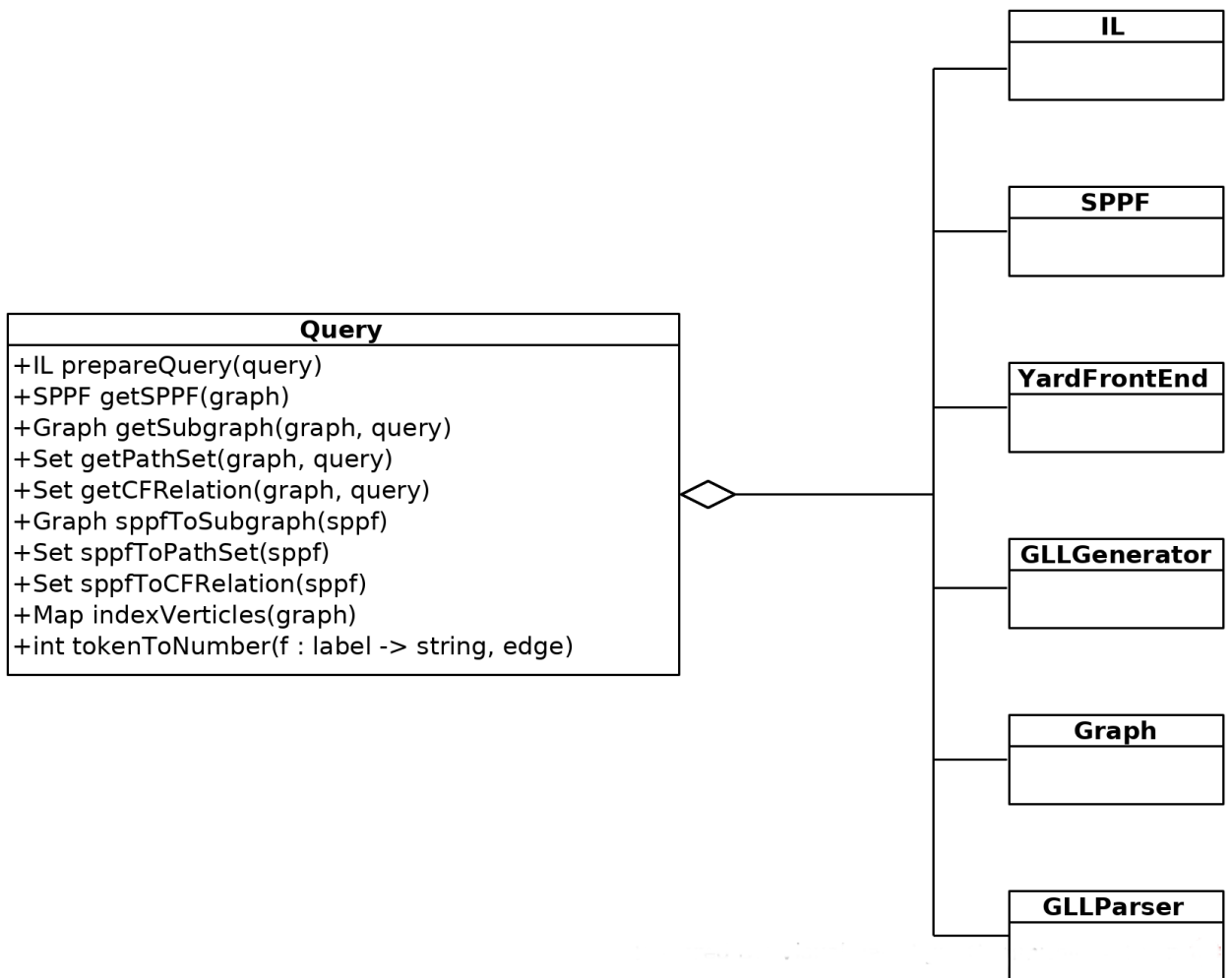


Рис. 2: Архитектура библиотеки

3. Реализация

3.1. Архитектура решения

Для предоставления конечному пользователю возможности написания запросов к графу G и представления результата в одной из нескольких форм была спроектирована архитектура библиотеки (рис.2). При разработке архитектуры были изучены статьи [11], [3], описывающие основные принципы дизайна функциональных библиотек. Пользователь получает возможность написания КС-запросов к задаваемым графам, но само исполнение запроса от него инкапсулировано и производится средствами YaccConstructor и YC.QuickGraph. Существование функций с низким уровнем абстракции позволяет, например, подгото-

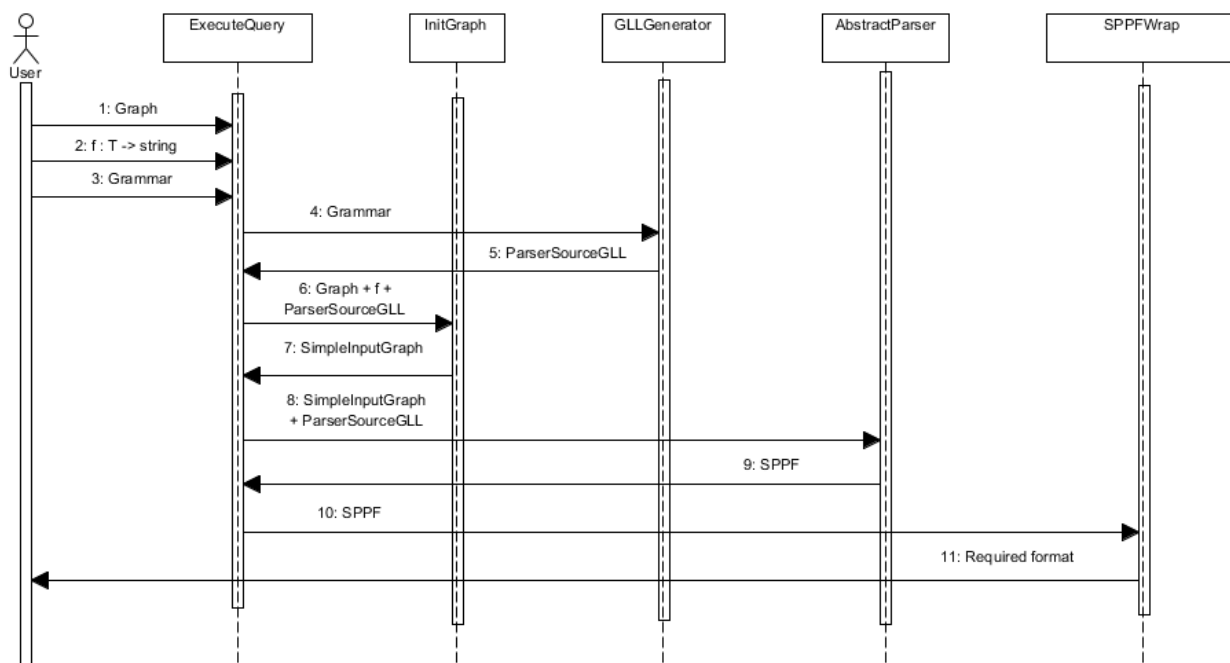


Рис. 3: Последовательность исполнения запроса

вить грамматику для исполнения запроса к нескольким графам. Существование же функций с высоким уровнем абстракции позволит пользователю просто задать запрос к графу и получить результат в желаемом формате. Таким образом, пользователь может комбинировать функции библиотеки, получая инструмент, применимый для решения множества задач.

3.2. Реализация библиотеки

В ходе реализации был решён ряд инженерных задач задач:

- реализовать класс представления графа, поддерживающий преобразование пользовательских объектов в токены;
- провести рефакторинг API проекта YaccConstructor;
- добавить поддержку генерации ParserSourceGLL для грамматик, заданных в виде строк;
- реализовать преобразования SPPF к подграфу, множеству путей, кратчайшему пути и КС-отношению.

Реализация поддержки преобразования пользовательских объектов на рёбрах графа нужна для запуска алгоритма синтаксического анализа на графе, поскольку алгоритм может быть применен только к графам с токенами на рёбрах. Сам процесс преобразования объекта к токenu инкапсулирован от пользователя; ему достаточно задать граф в виде одной из реализаций, существующих в библиотеке `YC.QuickGraph` и функцию, преобразующую объект на ребре в строку. После этого заданный граф преобразуется к нужному формату, а функция $f : EdgeObject \rightarrow token$ получается путём комбинации заданной пользователем функции $g : EdgeObject \rightarrow string$ и функции $u : string \rightarrow token$, получаемой после генерации `ParserSourceGLL` из задаваемой грамматики с помощью `GLLGenerator`. После данного преобразования на полученном графе и `ParserSourceGLL` можно запустить алгоритм синтаксического анализа.

Модуль, в котором было реализовано преобразование грамматики к `ParserSourceGLL`, был подвергнут рефакторингу. В связи с этим основная функция, выполняющая преобразование грамматики, была разбита на набор функций с более низким уровнем абстракции, что улучшило читаемость и понятность кода и позволило сохранить изначальную функциональность за счёт комбинации полученных функций. Кроме того, была добавлена поддержка генерации `ParserSourceGLL` из грамматик, заданных строкам.

3.3. SPPE

Для преобразования сжатого леса разбора к желаемым форматам был реализован итератор, позволяющий осуществить обход в ширину. Такой подход используется из-за того, что количество путей в графе, последовательность меток на которых выводима в задаваемом грамматикой языке, может быть бесконечным. Кроме того, обход в ширину позволяет избежать проблем при разборе графов, имеющих циклы. Итератор был написан в `lazy-evaluation` стиле, то есть результат запроса вычисляется только по требованию пользователя.

4. Заключение

В ходе работы достигнуты следующие результаты:

- изучена предметная область;
- проведен обзор статей, связанных с темой работы;
- разработана архитектура (рис.2) предлагаемого решения;
- реализована библиотека для выполнения КС-запросов к графам;
- написаны тесты и документация.

Список литературы

- [1] Abiteboul Serge, Vianu Victor. Regular path queries with constraints // Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems / ACM. — 1997. — P. 122–133.
- [2] Context-free path queries on RDF graphs / Xiaowang Zhang, Zhiyong Feng, Xin Wang et al. // International Semantic Web Conference / Springer. — 2016. — P. 632–648.
- [3] Fsharp.org. F# Component Design Guidelines // Fsharp.org. — URL: <http://fsharp.org/specs/component-design-guidelines/fsharp-design-guidelines-v14.pdf> (online; accessed: 15.05.2017).
- [4] Hale John. A probabilistic Earley parser as a psycholinguistic model // Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies / Association for Computational Linguistics. — 2001. — P. 1–8.
- [5] Hellings Jelle. Conjunctive context-free path queries. — 2014.
- [6] Koschmieder André, Leser Ulf. Regular path queries on large graphs // International Conference on Scientific and Statistical Database Management / Springer. — 2012. — P. 177–194.
- [7] Microsoft. AGL // Microsoft Research Site. — URL: <http://rise4fun.com/Ag1> (online; accessed: "15.05.2017").
- [8] NDepend. Graph Sharp // CodePlex. — URL: <http://graphsharp.codeplex.com/> (online; accessed: 06.12.2016).
- [9] Neo4J. Neo4J // Neo4J official page. — URL: <https://neo4j.com/> (online; accessed: 15.05.2017).
- [10] Neo4j. Cypher // Neo4j official page. — URL: <https://neo4j.com/developer/cypher/> (online; accessed: 15.05.2017).

- [11] Petricek Tomas. Library patterns // tomasp.net. — URL: <http://tomasp.net/blog/2015/library-frameworks/> (online; accessed: 15.05.2017).
- [12] Rekers Joan Gerard. Parser generation for interactive environments : Ph.D. thesis / Joan Gerard Rekers ; Citeseer. — 1992.
- [13] Scott Elizabeth, Johnstone Adrian. GLL parsing // Electronic Notes in Theoretical Computer Science. — 2010. — Vol. 253, no. 7. — P. 177–189.
- [14] Sevon Petteri, Eronen Lauri. Subgraph queries by context-free grammars // Journal of Integrative Bioinformatics. — 2008. — Vol. 5, no. 2. — P. 100.
- [15] Titan. Gremlin // Titan official page. — URL: <https://github.com/tinkerpop/gremlin/wiki> (online; accessed: 15.05.2017).
- [16] Titan. Titan // Titan official page. — URL: <http://titan.thinkaurelius.com/> (online; accessed: 15.05.2017).
- [17] YaccConstructor. YC.QuickGraph // YaccConstructor official page. — URL: <http://yaccconstructor.github.io/QuickGraph/> (online; accessed: 15.05.2017).
- [18] YaccConstructor. YaccConstructor // YaccConstructor official page. — URL: <http://yaccconstructor.github.io> (online; accessed: 15.05.2017).
- [19] de Halleux Jonathan Peli. QuickGraph // CodePlex. — URL: <http://quickgraph.codeplex.com/> (online; accessed: 15.05.2017).
- [20] Авдюхин ДА. Создание генератора GLR трансляторов для .NET // URL: http://se.math.spbu.ru/SE/YearlyProjects/2012/YearlyProjects/2012/445/445_Avdyukhin_report.pdf. — 2012.
- [21] Азимов Рустам Шухратуллович, Rustam Azimov. Syntax error detection in dynamically generated code. — 2016.

- [22] Григорьев Семён Вячеславович. Синтаксический анализ динамически формируемых программ.
- [23] Кириленко ЯА, Григорьев СВ, Авдюхин ДА. Разработка синтаксических анализаторов в проектах по автоматизированному реинжинирингу информационных систем // Научно-технические ведомости СПбГПУ: информатика, телекоммуникации, управление. — 2013. — no. 174. — P. 94–98.
- [24] Ковалев Дмитрий. Реализация и оценка эффективности алгоритма обобщенного синтаксического анализа с уменьшенной активностью стека. — 2016.
- [25] Полубелова Марина Игоревна. Генератор абстрактных лексических анализаторов // Курсовая работа кафедры системного программирования СПбГУ.—2014.—URL: <http://se.math.spbu.ru/SE/YearlyProjects/2014/YearlyProjects/2014/344/344-Polubelova-report.pdf>.
- [26] Рагозина Анастасия Константиновна, Шкредов СД. Ослабленный синтаксический анализ динамически формируемых программ на основе алгоритма GLL. — 2016.