Санкт-Петербургский государственный университет

Кафедра Системного программирования

Свитков Сергей Андреевич

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

Курсовая работа

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

Санкт-Петербург
2017

Оглавление

Введение

Модель представления данных в виде ориентированных графов с метками на ребрах имеет широкую область применения и используется в биоинформатике, социальных исследованиях (например, при представлении социальных графов), semantic web, при реализации графовых баз данных.

При наличии данных, представленных в виде такой структуры, возникает задача их обработки. Наиболее часто требуется выбрать из всего набора данных какую-либо конкретную часть. Такие задачи в базах данных решаются, как правило, с помощью задания запросов на некоем специализированном языке. Существует множество таких языков, применительно к графовым базам данных наиболее известными являются Gremlin[14], Cypher[9], используемые в графовых базах данных Titan и Neo4J соответственно. Однако, эти языки являются регулярными, а значит, не могут применяться для решения ряда задач. Например, при разборе генеалогического дерева часто возникает задача поиска потомков общего предка. Она решается поиском строк вида $parent^n child^n$. Но такие строки не могут содержаться в регулярных языках, однако, существуют в языке, задаваемом контекстно-свободной (в дальнейшем — КС) грамматикой с правилами вывода $N \to parent\, child$, $N \to parent N\, child$. Таким образом, актуальной задачей является применение КС-языков запросов для расширения класса задач, решаемых с помощью регулярных языков.

Существуют работы, предлагающие различные подходы к реализации языков запросов, например [13], [5], [2], [1], [6]. Большая часть из них представляет только теоретические сведения о реализации, а те, что реализованы на практике, имеют довольно ограниченный функционал или же очень узкую специализацию. Однако, как уже было отмечено ранее, класс задач, решение которых возможно с помощью КС-запросов, является обширным. Исходя из этого, реализация средства для выполнения запросов должна быть применима для решения различных задач, связанных с данной тематикой. Логично, что для со-

ответствия этим требованиям необходим широкий функционал и различные форматы представления результата запроса. Из этого можно сделать вывод о том, что для возможности применить реализацию к различным задачам обработки графовых структур данных, её удобнее всего оформить как библиотеку. Исходя из этого было принято решение реализовать библиотеку для выполнения КС-запросов к графам, позволяющую представить результат исполнения запроса в нескольких форматах.

В качестве платформы для реализации была выбрана .NET. Для неё существует ряд библиотек для работы с графами, например GraphSharp [8], Automatic Graph Layout [7], но наиболее известной является QuickGraph [17], работа над которой прекращена в 2011 году. В лаборатории языковых инструментов JetBrains с 2015 года ведется разработка и поддержка библиотеки YC.QuickGraph [15], за основу которой была взята QuickGraph. В данной библиотеке имеется множество инструментов для работы с графами, поэтому было принято решение использовать её для реализации. В качестве алгоритма для синтаксического анализа графов используется GLL[12], поскольку он имеет хорошую асимптотику и может обрабатывать все КС-грамматики, в том числе и леворекурсивные, а так же в рамках работы [24] был реализован и интегрирован в YaccConstructor [16], [18], [21], [20], [21], [19], [22], [23] — 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

Резюмируя сказанное во введении, была поставлена цель работы: реализовать библиотеку для выполнения КС-запросов к графам, используя .NET как основную платформу, YC.QuickGraph — в качестве библиотеки для представления графов и YaccConstructor в качестве набора инструментов 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;
- 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 CYK 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 проводились на компьютере с 1GB оперативной памяти, в качестве ОС использовался Linux. Графы, на которых проводилось тестирование алгоритма, генерировались со следующими ограничениями: фиксированный размер в 10000 вершин, изменяемое ограничение на максимальную длину пути, а так же регулируемая вероятность существования ребра между двумя вершинами. Например, запрос к сгенерированному графу с максимальной длиной

пути в 9 вершин выполняется около 200 секунд. Эксперименты, поставленные на реальных данных, показали, что для графа с максимальной длиной пути, равной 8 вершинам, время работы алгоритма может достигать 250 секунд. Единственный формат представления результата, а также большое время исполнения запроса являются основными причинами, по которым результаты данной работы едва ли применимы на практике.

Ослабленный синтаксический анализ динамически формируемых выражений на основе алгоритма GLL

Работа Анастасии Рагозиной, написанная на кафедре СП Математико-Механического факультета СПбГУ предлагает алгоритм для синтаксического анализа динамически формируемого кода на основе алгоритма GLL, позволяющий проверять выводимость последовательсти меток на ребрах графа 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 алгоритма. Следует отметить, что результатом работы алгоритма является лес разбора, представляемый в виде SPPF[11], который можно отобразить в нужный формат вывода. Учитывая наличие реализации алгоритма для платформы .NET в проекте YaccConstructor, было принято решение использовать результаты этой работы 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,

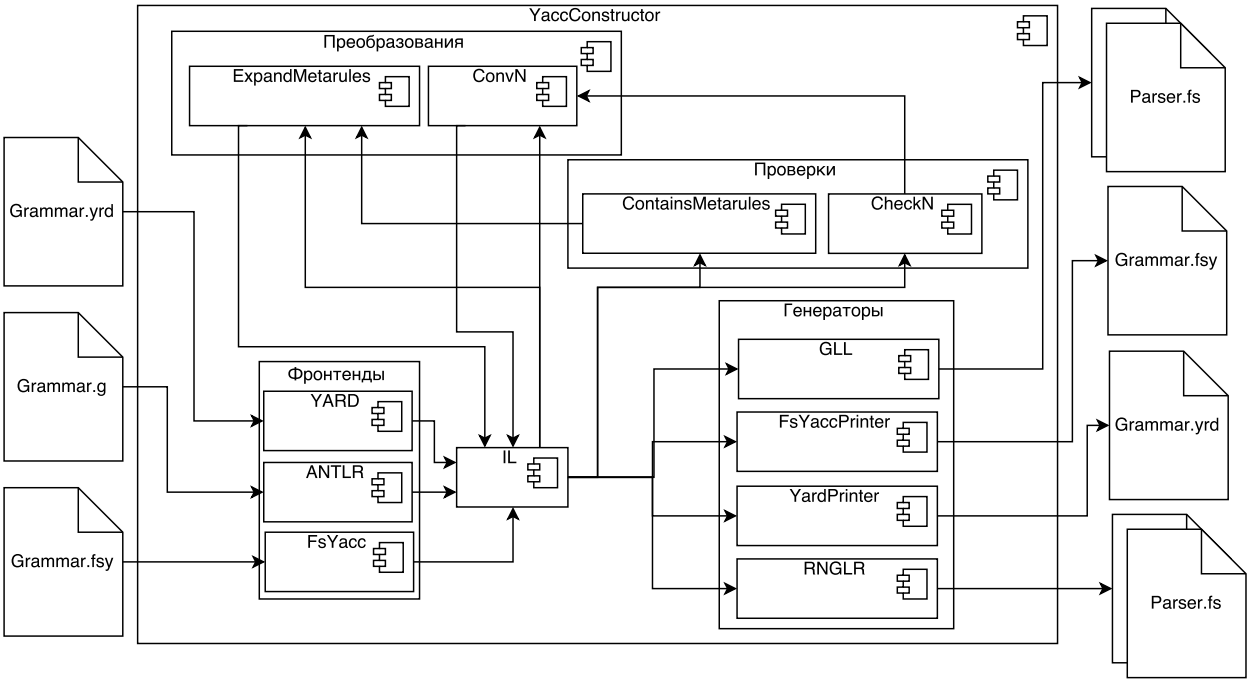

Рис. 1: Архитектура проекта YaccConstructor, заимствована из работы [20]

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 такие средства YaccConstructor, как YARD, GLLParser 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 — алгоритм синтаксического анализа, поддерживающий все типы КС-грамматик (в том числе и леворекурсивные), кроме того, имеющий асимптотику $O(n)$ для однозначных граматик и $O(n^3)$ в худшем случае. В данной работе GLL будет использоваться для синтаксического анализа графов. YaccConstructor имеет 2 модуля, использующих GLL — GLLGenerator и GLLParser. Первый отвечает за генерацию парсеров для заданных грамматик, а второй, соответственно, за синтаксический анализ. Результатом работы GLLParser является лес разбора — SPPF, из которого с помощью

различных функций, которые планируется реализовать в рамках данной работы, можно 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 [17], работа над которой была прекращена в 2011 году. YC.QuickGraph имеет средства для представления графов, различные алгоритмы для них (DFS, BFS, поиск кратчайшего пути и др.). Функионал данной библиотеки планируется использовать для представления графа и построения нужного формата вывода.

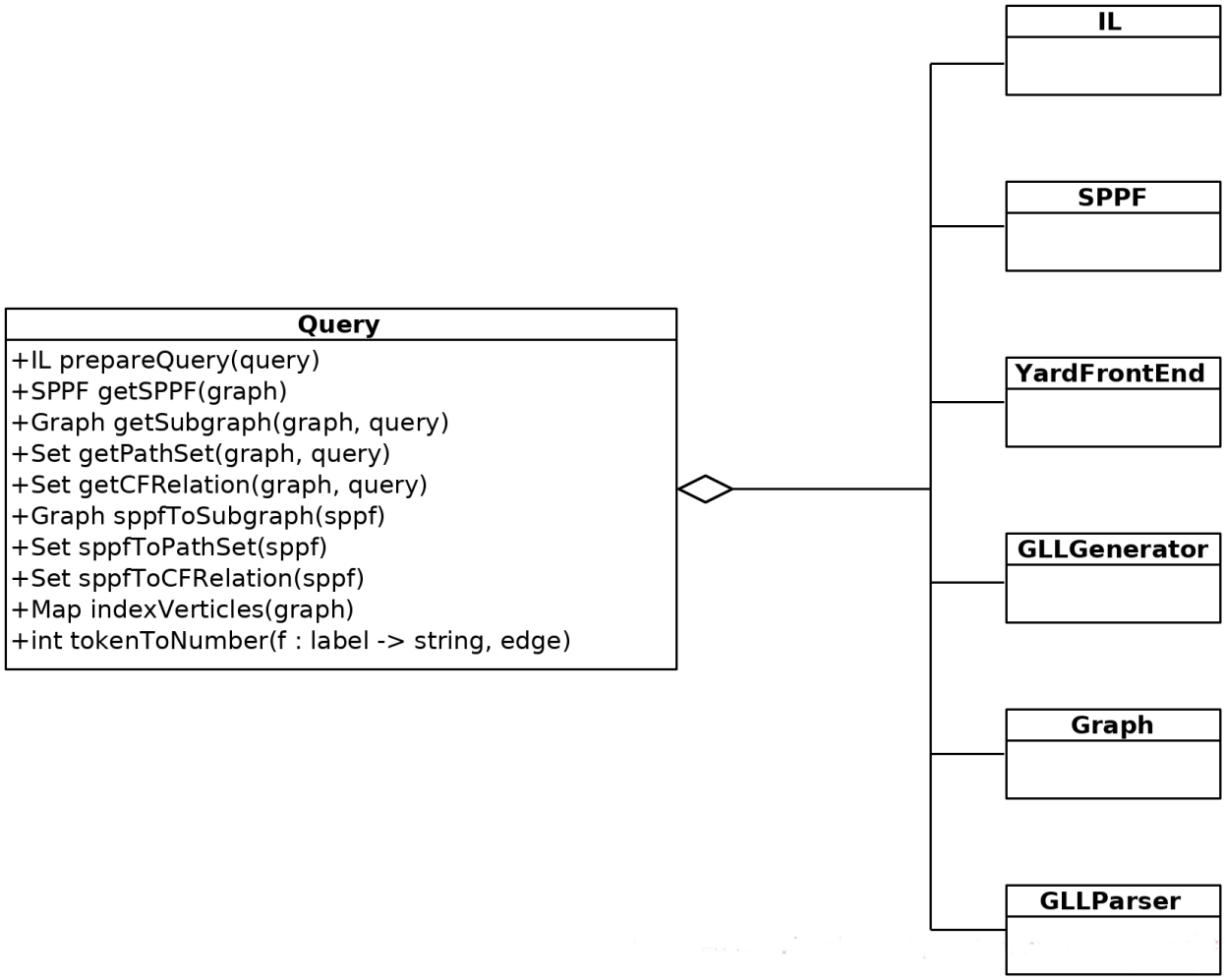

Рис. 2: Архитектура библиотеки

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 [10], [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-

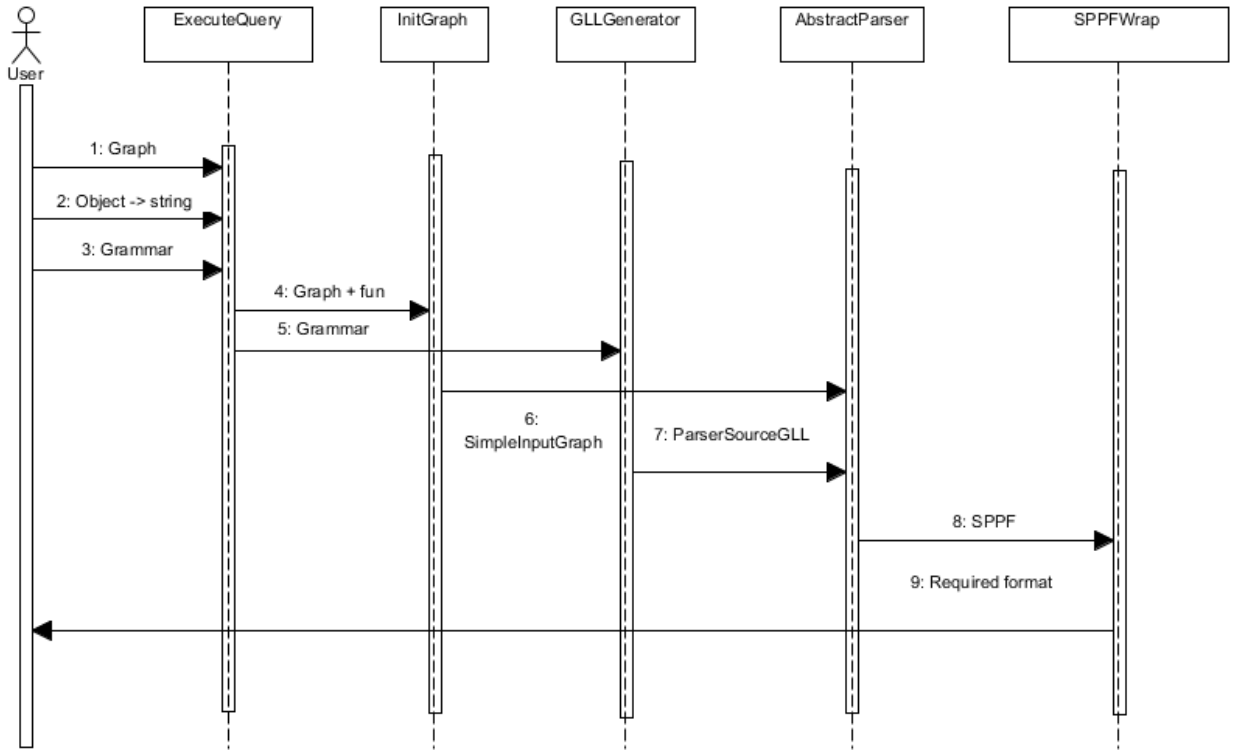

Рис. 3: Последовательность исполнения запроса

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

Для реализации библиотеки необходимо было решить ряд инженерных задач задач:

- реализовать класс представления графа, позволяющий преобразовывать пользовательские объекты на рёбрах в токены;
- провести рефакторинг модуля, отвечающего за генерацию ParserSourceG
- 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 данном графе, поскольку алгоритм может разбирать только графы с токенами — численным значениями — на рёбрах. Сам процесс преобразования объекта 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 $f : EdgeObject- > token$ получается путём комбинации заданной пользователем функции $g : EdgeObject- > string$ и функции $u : string- > token$, получаемой из ParserSourceGLL после разбора грамматики с помощью GLLGenerator. После этого на полученном графе и полученном из грамматики ParserSourceGLL можно запустить алгоритм синтаксического анализа.

Модуль, в котором было реализовано преобразование грамматики к ParserSourceGLL, был подвергнут рефакторингу, поскольку ... 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 строками.

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: 18.12.2016).

[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/Agl` (online; accessed: 06.12.2016).

[8] NDepend. Graph Sharp // CodePlex. — URL: `http://graphsharp.codeplex.com/` (online; accessed: 06.12.2016).

[9] Neo4j. Cypher // Neo4j official page. — URL: `https://neo4j.com/developer/cypher/` (online; accessed: 22.11.2016).

[10] Petricek Tomas. Library patterns // tomasp.net. — URL: http://tomasp.net/blog/2015/library-frameworks/ (online; accessed: 18.12.2016).

[11] Rekers Joan Gerard. Parser generation for interactive environments : Ph. D. thesis / Joan Gerard Rekers ; Citeseer. — 1992.

[12] Scott Elizabeth, Johnstone Adrian. GLL parsing // Electronic Notes in Theoretical Computer Science. — 2010. — Vol. 253, no. 7. — P. 177–189.

[13] Sevon Petteri, Eronen Lauri. Subgraph queries by context-free grammars // Journal of Integrative Bioinformatics. — 2008. — Vol. 5, no. 2. — P. 100.

[14] Titan. Gremlin // Titan official page. — URL: https://github.com/tinkerpop/gremlin/wiki (online; accessed: 29.11.2016).

[15] YaccConstructor. YC.QuickGraph // YaccConstructor official page. — URL: http://yaccconstructor.github.io/QuickGraph/ (online; accessed: 22.11.2016).

[16] YaccConstructor. YaccConstructor // YaccConstructor official page. — URL: http://yaccconstructor.github.io (online; accessed: 29.11.2016).

[17] de Halleux Jonathan Peli. QuickGraph // CodePlex. — URL: http://quickgraph.codeplex.com/ (online; accessed: 06.12.2016).

[18] Авдюхин ДА. Создание генератора GLR трансляторов для .NET // URL: http://se. math. spbu. ru/SE/YearlyProjects/2012/YearlyProjects/2012/445/445_Avdyukhin_repo pdf. — 2012.

[19] Азимов Рустам Шухратуллович, Rustam Azimov. Syntax error detection in dynamically generated code. — 2016.

[20] Григорьев Семён Вячеславович. Синтаксический анализ динамически формируемых программ.

[21] 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.

[22] Ковалев Дмитрий. Реализация и оценка эффективности алгоритма обобщенного синтаксического анализа с уменьшенной активностью стека. — 2016.

[23] Полубелова Марина Игоревна. Генератор абстрактных лексических анализаторов // Курсовая работа кафедры системного программирования СПбГУ.—-2014.—-URL: h ttp://se. math. spbu. ru/SE/YearlyProjects/2014/YearlyProjects/2014/344/344-Polubelova-report. pdf.

[24] Рагозина Анастасия Константиновна, Шкредов СД. Ослабленный синтаксический анализ динамически формируемых программ на основе алгоритма GLL. — 2016.