Санкт-Петербургский государственный университет

Кафедра Системного программирования

Соловьев Александр Александрович

Разработка архитектуры для унификации синтаксических анализаторов в проекте YaccConstructor

Курсовая работа

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

Санкт-Петербург
2017

Оглавление

Введение

Основной задачей синтаксического анализа является задача проверки выводимости некоторой последовательности токенов в заданной грамматике и построением соответствующего дерева вывода. Данная задача может быть рассмотрена шире, так как в качестве объекта синтаксического анализа могут рассматриваться структуры данных, отличные от последовательностей. Так, в работах [6] и [8] синтаксическому анализу подвергается граф.

В 2010 году был представлен алгоритм обобщенного синтаксического анализа Generalized LL (GLL), в основе которого лежит алгоритм нисходящего синтаксического анализа [1]. Данный алгоритм и различные его модификации были реализованы в проекте YaccConstructor [5] в качестве отдельных не связанных между собой модулей (рис. 2)). Различаются они в структурах, представляющих входные данные, в представлении вспомогательных структур и в возможности построения деревьев вывода. В то время, как основная структура различных версий алгоритма неизменна, внутренняя их реализация различна в зависимости от используемых структур и необходимости построения деревьев вывода. Несмотря на то, что алгоритмы в большей части одинаковы, их поддержка и сопровождение затруднены в связи с независимостью реализаций. Данную проблему возможно решить их обобщением. Теоретически оно возможно, однако на практике может привести к возникновению различных трудностей, получившееся решение может оказаться настолько более сложным, что его сопровождение окажется еще более проблематичным, чем сопровождение всего набора независимо реализованных алгоритмов. Наиболее вероятным недостатком может оказаться значительное падение производительности. Так представление последовательности токенов в виде графа ведет к появлению циклов перебора всех исходящих из вершины ребер, что ведет к росту числа операций.

1. Постановка задачи

Целью данной работы является разработка архитектуры для унификации существующих синтаксических анализаторов в проекте YaccConstructor. Для достижения данной цели были поставлены следующие задачи:

- спроектировать архитектуру, позволяющую объединить различные модификации алгоритма;
- реализовать предложенную архитектуру;
- разработать тестовое покрытие;
- провести эксперименты для оценки производительности.

2. Обзор

2.1. YaccConstructor

YaccConstructor — проект, разрабатываемый в лаборатории языковых инструментов JetBrains, расположенной на кафедре системного программирования. В нем занимаются исследованиями и разработками в области лексического и синтаксического анализа. Большинство компонентов проекта реализованы на языке F#, исходный код проекта находится в открытом доступе [5]. Проект имеет модульную архитектуру (рис.1), что позволяет собирать требуемый инструмент из существующих модулей: можно выбрать фронтенд, задать требуемые преобразования грамматики и указать генератор. Фронтенды позволяют построить по спецификации грамматики ее внутреннее представление, к которому могут быть применены необходимые преобразования, генераторы предоставляют инструменты, позволяющие по внутреннему представлению грамматики получить нужный пользователю результат, например, таким результатом может быть синтаксический анализатор.

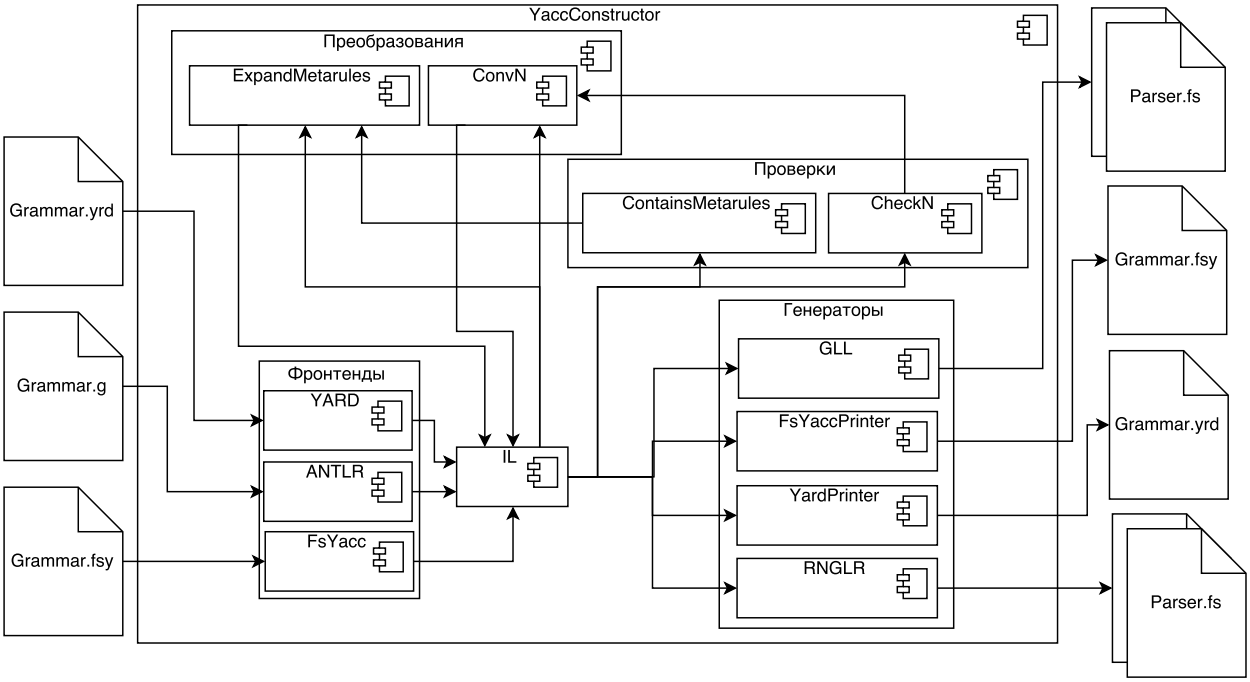

Рис. 1: Архитектура YaccConstructor, заимствована из [7]

2.2. Синтаксические анализаторы на основе GLL

Как говорилось выше, в проекте было реализовано несколько модификаций GLL, различавшихся в том, подавались ли на вход алгоритму грамматика или же рекурсивный автомат [4], в объекте синтаксического анализа, а также в возможности построения дерева.

2.2.1. Generalized LL

GLL имеет некоторые преимущества перед прочими алгоритмами синтаксического анализа:

- разбор любых контекстно-свободных грамматик, в том числе и неоднозначных;
- время исполнения в худшем случае кубически зависит от размера входных данных;
- алгоритм обладает свойством "рекурсивного спуска": анализатор может быть легко построен напрямую по грамматике.

В 2013 году была представлена статья [2], в которой описывалось конструирование структуры SPPF (Shared Packed Parse Forest) в ходе работы GLL. SPPF представляет все возможные деревья вывода строки в заданной грамматике.

2.2.2. Синтаксический анализ графов

При решении различных практических задач может возникнуть необходимость проверки выводимости элементов некоторого регулярного множества в заданной грамматике. Подобное множество может быть и бесконечным, в таком случае проверка всех его элементов на выводимость не представляется возможной. Однако, поскольку регулярные множества описываются при помощи конечных автоматов, задачу можно свести к проверке выводимости элементов, заданных конечным автоматом в заданной грамматике. Примером подобной задачи может стать проверка корректности динамически формируемых SQL-запросов.

2.2.3. Использование рекурсивного автомата

Процесс конструирования синтаксических анализаторов может быть автоматизирован с помощью генераторов при наличии требуемой спецификации. Предствленая она может быть в расширенной форме Бэкуса-Наура, справиться с которой способны немногие решения, однако они не умеют работать с неоднозначными грамматиками. В работе [3] представлена модификация GLL, позволяющая работать с грамматиками в форме, близкой к РФБН. Также показано, что данное решение имеет лучшую проивзодительность, чем при трансформации грамматики.

2.3. Изначальная архитектура

Как уже говорилось, основной проблемой данной архитектуры (рис. 2) являлась проблематичность ее сопровождения, что следовало из целого ряда ее недостатков. Одним из самых серьезных недостатков является то, что используемые алгоритмами структуры данных, основными из которых являются GSS и SPPF не были выделены в отдельные сущности и реализовывались для каждого алгоритма отдельно с незначительными изменениями. Некоторые структуры данных все же использовались какой-либо группой алгоритмов, но при этом, как правило, имелись структуры данных, испольнявшие аналогичные роли, но в других группах алгоритмов. Последнее приводило к тому, что при изменении таких структур данных изменению также должны были подвергнуться и использовавшие ее алгоритмы. Многое из сказанного выше было справедливо и для функций.

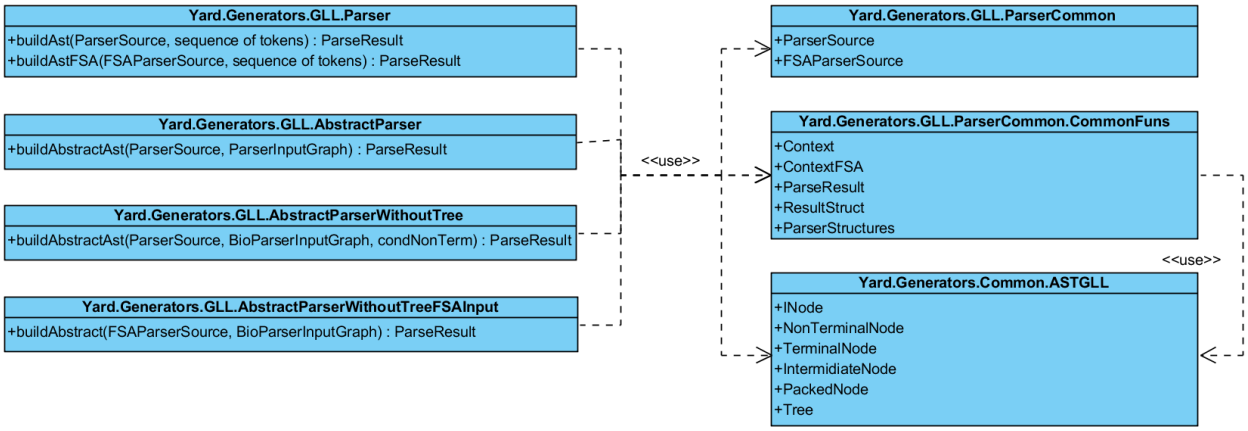

Рис. 2: Взаимодействие основных модулей при изначальной архитектуре

3. Новая архитектура

Реализованная архитектура позволяет избежать упоямнутых ранее проблем. Структуры GSS (Graph Structured Stack) и SPPF были выделены в отдельные сущности, что позволило снять с алгоритма ответственность за работу с ними. Для различных входных данных было решено использовать абстракцию, конкретные реализации которой ответственны за различные действия над входными данными, что позволило обобщить алгоритм для различных входных данных. Подобное решение обусловленно тем, что различия в действиях над входными данными зависят исключительно от их внутреннего представления. Из-за описанных в разделе 2.2.3 достоинств рекурсивного автомата было решено использовать его в качестве единственного представления грамматики. Был добавлен флаг, определяющий, будет ли в процессе работы алгоритма строиться дерево вывода. Таким образом осталась только одна версия алгоритма, обобщенная для различных входных данных и умеющая строить дерево при необходимости.

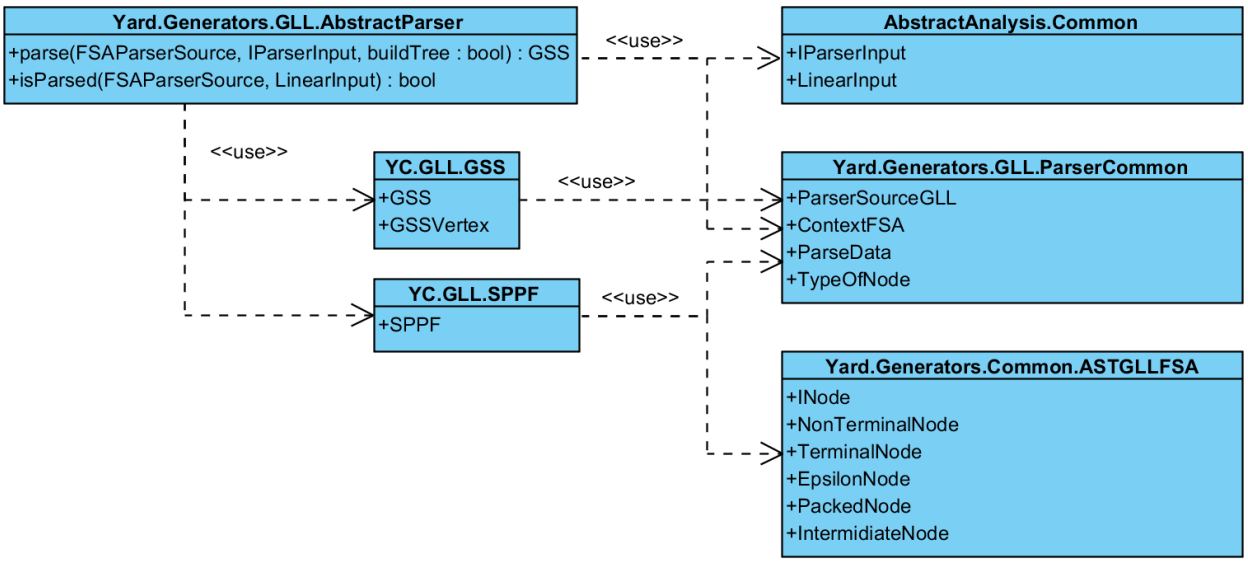

Рис. 3: Архитектура решения после унификации

4. Эксперименты

В рамках данной работы были произведены эксперемнты по сравнению производительности алгоритмов до их обобщения и после. Замеры производились на компьютере со следующими характеристиками:

- Операционная система: Microsoft Windows 10 Home
- Тип системы: x64-based PC
- Процессор: Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz, 2.40GHz, 4 Core(s), 8 Logical Processor(s)
- Объем оперативной памяти: 8.0 GB

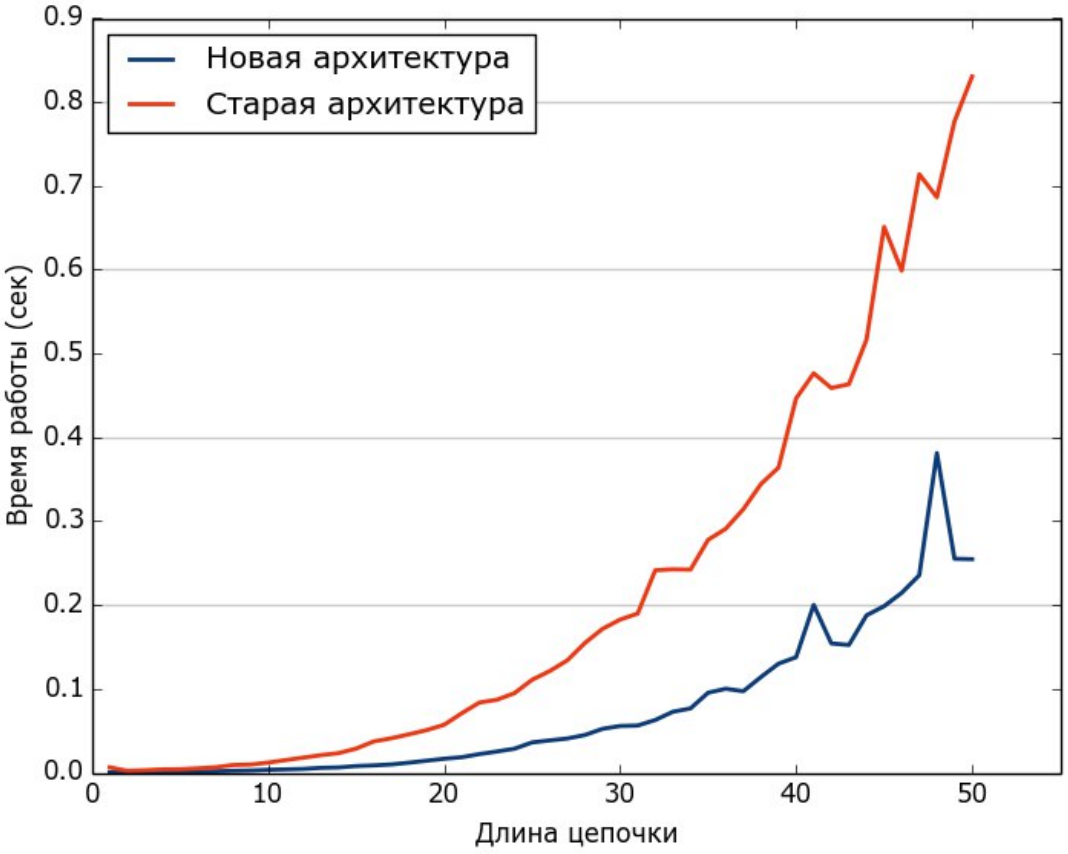

Рис. 4: Сравнение времени работы старого и нового решения для грамматики G

Для сравнения были выбраны выбраны две грамматики, первая из которых – сильно неоднозначная грамматика G, реализующая худший случай для анализатора. Результаты измерений представлены на рис. 4.

$$s \to s\ s\ s \mid s\ s \mid B$$

Listing 1: Грамматика G

Следующий эксперимент проводился на грамматике подмножества языка T-SQL [9]. Пример входного графа приведен на рис. 6. Результаты измерений представлены на рис. 5

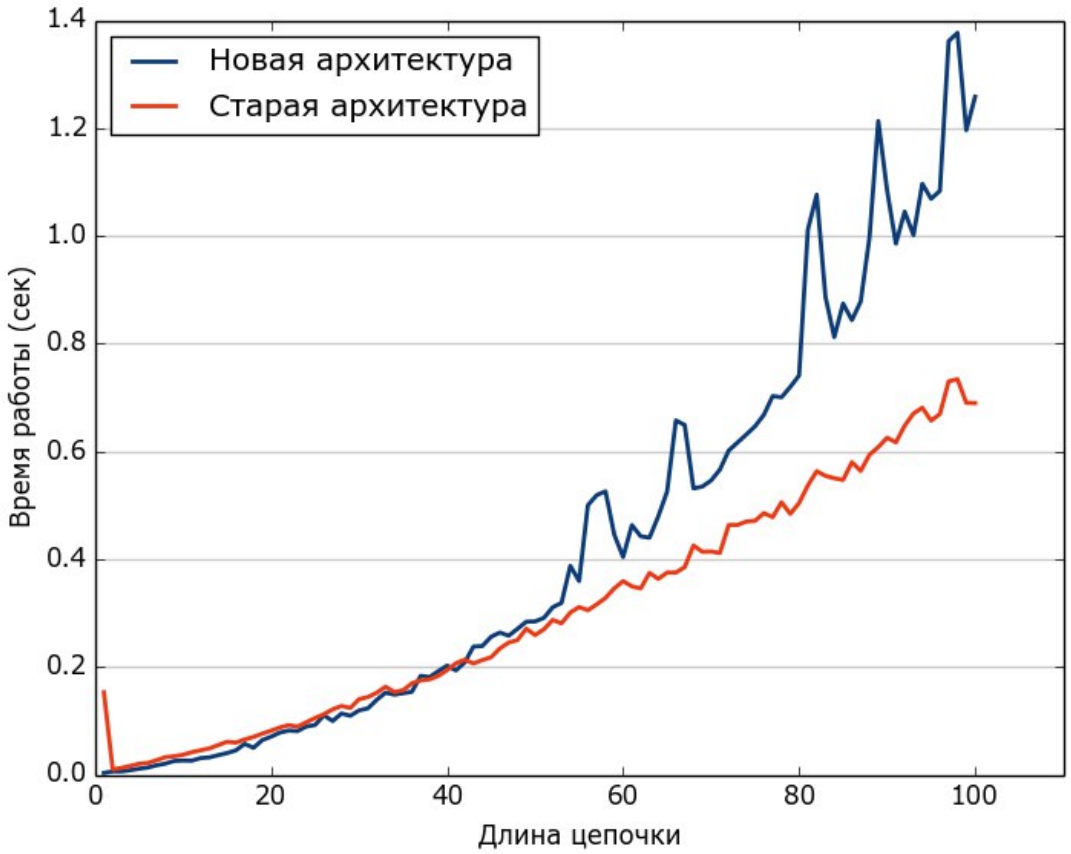

Рис. 5: Сравнение времени работы старого и нового решения для грамматики T-SQL

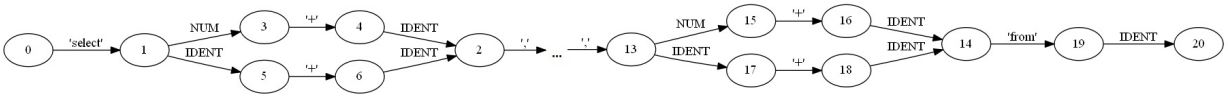

Рис. 6: Структура графа для эксперимента на T-SQL, заимствована из [8]

Результаты экспериментов выше наглядно показывают, что производительность алгоритма осталась на том же уровне, о ее изменении можно судить лишь для конкретных входных данных.

5. Заключение

Достигнуты следующие результаты:

- произведен обзор статей, связанных с предметной областью;
- написан обзор предметной области;
- спроектирована и реализована архитектура (рис. 3);
- разработано тестовое покрытие;
- проведены эксперименты для оценки производительности.

Список литературы

[1] E. Scott, A. Johnstone. GLL Parsing // Electron. Notes Theor. Comput. Sci. — 2010. — Vol. 253, no. 7. — P. 177–189.

[2] E. Scott, A. Johnstone. GLL parse-tree generation // Science of Computer Programming. — 2013. — Vol. 78, no. 10. — P. 1828–1844.

[3] Gorokhov Artem, Grigorev Semyon. Extended Context-Free Grammars with Generalized LL, неопуб. — 2016.

[4] Tellier Isabelle. Learning recursive automata from positive examples // Revue des Sciences et Technologies de l'Information-Série RIA: Revue d'Intelligence Artificielle. — 2006. — Vol. 20, no. 6. — P. 775–804.

[5] YaccConstructor. YaccConstructor // YaccConstructor official page. — URL: `http://yaccconstructor.github.io` (online; accessed: 18.12.2016).

[6] Вербицкая Екатерина Андреевна. Синтаксический анализ регулярных множеств. — 2015.

[7] Григорьев Семен Вячеславович. Синтаксический анализ динамически формируемых программ. — 2016.

[8] Рагозина Анастасия Константиновна. Ослабленный синтаксический анализ динамически формируемых выражений на основе алгоритма GLL. — 2016.

[9] Репозиторий, содержащий различные грамматики. — URL: `https://github.com/YaccConstructor/YC.GrammarZOO` (online; accessed: 10.10.2017).