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

В области синтаксического анализа существует множество различных задач. Наиболее известной из них является задача анализа некоторой последовательности токенов с целью проверки ее выводимости в заданной грамматике и построением соответствующего дерева вывода. Однако данная задача может быть рассмотрена более широко, поскольку подвергаться синтаксическому анализу могут не только строки, но и другие структуры данных. Например, в работах [6] и [8] в качестве объекта синтаксического анализа рассматривается граф.

В 2010 году был предложен 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] независимо друг от друга (рис. 2)). Различия их заключаются в первую очередь в структурах данных, которые подаются на вход, а также в возможности построения с помощью этих алгоритмов деревьев вывода. Конечно, отличается также и их внутренняя реализация, но общая структура различных версий при этом не меняется. Из сказанного выше следует, что поддерживать и сопровождать приходится все реализованные алгоритмы. Эту проблему могло бы решить их обобщение. Теоретически оно возможно, однако на практике возможно возникновение различных трудностей, получившееся решение может обладать рядом недостатков по сравнению с набором отдельно реализованных алгоритмов. Решение может быть крайне громоздким, что может еще больше усложнить его поддержку, одним же из наиболее вероятных недостатков, которые могут возникнуть, является значительное падение производительности. Например, при попытке представить линейный вход в виде графа, появляются циклы для перебора всех исходящих из вершины ребер, что приводит к увеличению 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. Генераторы предоставляют инструменты, позволяющие по внутреннему представлению грамматики получить полезный для конечного пользователя результат. Примером такого результата являются синтаксические анализаторы.

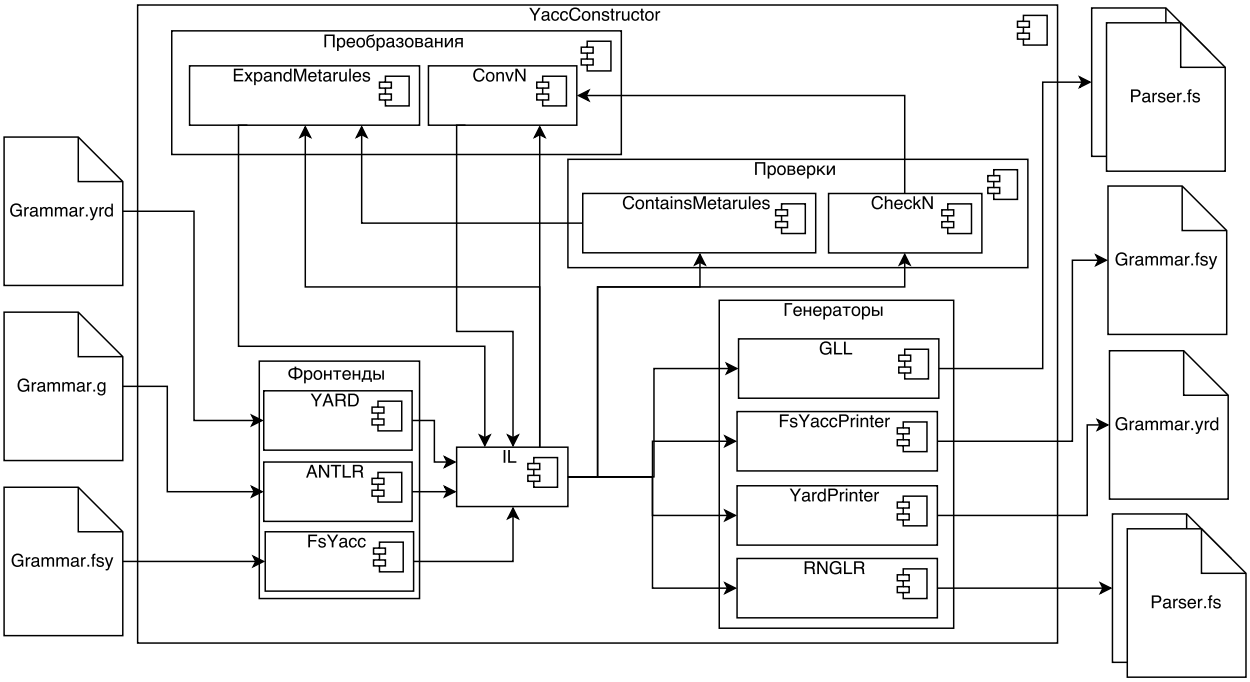

Рис. 1: Архитектура YaccConstructor, заимствована из [7]

2.2. Синтаксические анализаторы на основе GLL

Как говорилось выше, на данный момент в проекте реализованы GLL и несколько его модификаций. Различаются они в рассматриваемой структуре данных, в возможности построения дерева вывода, а также в том, подается ли на вход алгоритму грамматика или же рекурсивный автомат [4].

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 длины входной строки;
- алгоритм обладает свойством "рекурсивного спуска": анализатор может быть легко построен напрямую по грамматике.

Также, в 2013 году была представлена статья [2], в которой описывалось построение дерева вывода при помощи GLL. В качестве дерева вывода в статье конструировалась структура SPPF (Shared Packed Parse Forest), представляющая все возможные выводы 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. Если такое множество бесконечно, проверка всех его элементов на выводимость попросту невозможна. Однако, зачастую такие множества описываются при помощи конечных автоматов, таким образом задача сводится к проверке выводимости элементов, заданных конечным автоматом в заданной КС-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 при наличии требуемой спецификации с помощью генераторов. Подобным описание при этом может предоставляться в расширенной форме Бэкуса-Наура, справиться с которой способны немногие решения, при этом те из них, что способны, не умеют работать с неоднозначными грамматиками. Так в работе [3] предлагается 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 2 является проблематичность ее сопровждения, что является результатом целого ряда ее недостатков. Одним из самых серьезных недостатков является то, что используемые алгоритмами структуры данных, основными из которых являются GSS и SPPF, практически полностью реализуются отдельно для каждого алгоритма, причем различичя этих реализаций, как правило, незначительны. Некоторые структуры данных все же вынесены в прочие модули и используются несколькими реализациями сразу, но, как правило, для прочих реализаций существует практически такая же структура данных, исполняющая ту же роль в алгоритмах. Последнее приводит также и к тому, что при изменении такой общей структуры данных изменению также должны подвергнуться и использующие ее алгоритмы. Многое из сказанного выше распространяется также и на функции. Также проблематично расширение подобной архитектуры, ведь для некоторого нового алгоритма потребуется снова реализовывать многие структуры данных и функции, возможно, будет иметься необходимость в реализации сразу нескольких версий

алгоритма, например, для разных представлений грамматики.

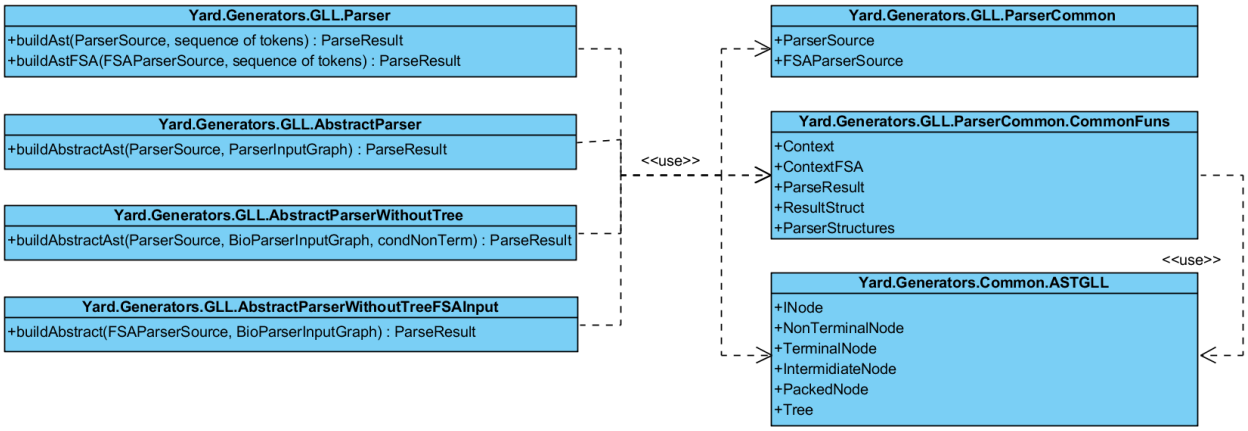

Рис. 2: Взаимодействие основных модулей при изначальной архитектуре

3. Предложенная архитекура

Предложенное решение позволяет избежать упоямнутых ранее проблем. В связи с тем, что различия в действии над входными данными зависят исключительно от их внутреннего представления, было решено использовать некоторую абстракцию, конкретные реализации которой ответственны за эти различия. Подобное решение позволяет обобщить алгоритм для различных входных данных. Также было решено в качестве представления грамматики использовать рекурсивный автомат из-за его достоинств, которые были описаны в разделе 2.2.3. Структуры GSS (Graph Structured Stack) и SPPF в свою очередь были выделены в отдельные сущности, что приводит к снятию с алгоритма ответственности за работу с ними. Различие в построении дерева релазиуется через передаваемый алгоритму флаг, что позволяет строить дерево лишь, когда оно нужно пользователю. Таким образом остается только одна версия алгоритма, обобщенная для входной грамматики и представления входных данных и умеющая строить дерево при необходимости.

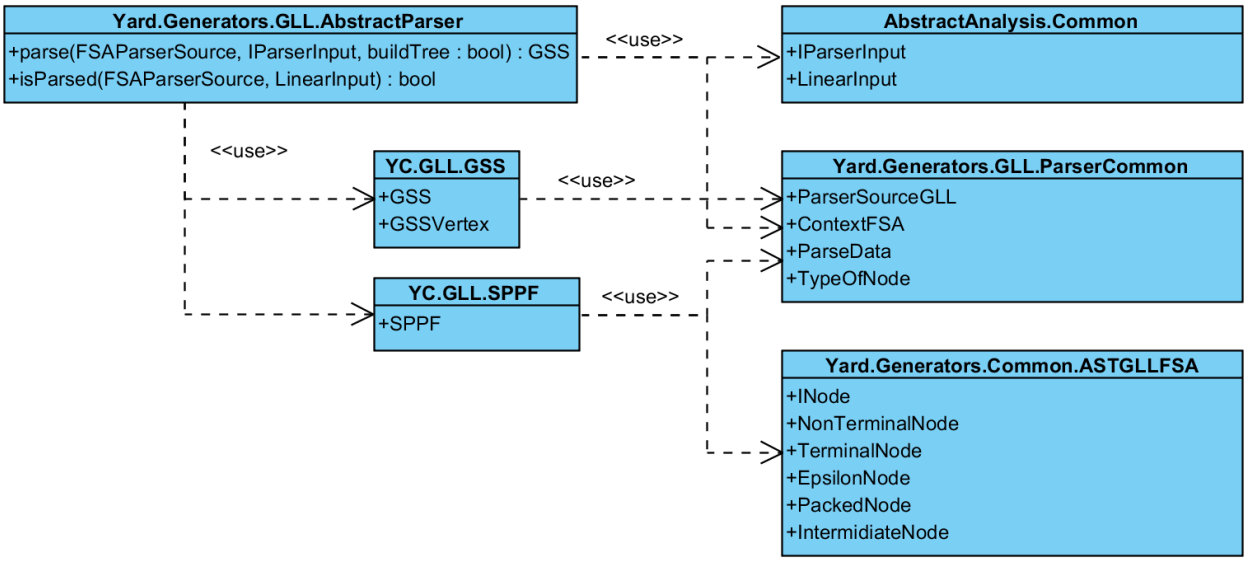

Рис. 3: Архитектура решения после унификации

4. Заключение

Достигнуты следующие результаты:

- произведен обзор статей, связанных с предметной областью;
- написан обзор предметной области;
- спроектирована и реализована архитектура (рис. 3).

В дальнейшем планируется:

- написать докумментацию;
- разработать тестовое покрытие;
- провести 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.