Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра Системного Программирования

Полубелова Марина Игоревна

Генератор абстрактных лексических анализаторов

Курсовая работа

Научный руководитель:
старший преподаватель Григорьев С. В.

Санкт-Петербург
2014

SAINT PETERSBURG STATE UNIVERSITY
Mathematics and Mechanics Faculty

Software Engineering Chair

Marina Polubelova

Abstract lexers generator

Graduation Thesis

Admitted for defence.
Head of the chair:

Scientific supervisor:
senior lecturer Semyon Grigorev

Reviewer:

Saint Petersburg
2014

Оглавление

Введение

Современные языки программирования позволяют во время выполнения программы формировать выражения на других языках и выполнять их. Языки, которые используются для написания таких динамически формируемых выражений, называются встроенными. Одной из трудностей применения разнообразных встроенных языков является отсутствие какой-либо информации о правильности составленного выражения до этапа выполнения основной программы. Как следствие, ошибки выявляются лишь во время выполнения программы, что существенно увеличивает затраты на разработку, отладку и сопровождение приложений, в которых используются встроенные языки.

В качестве примера рассмотрим SQL встроенный в C#:

```
class Example
   {
       private void Go(int cond)
       {
           string tableName;
           if (cond > 0)
           {
               tableName = "1";
           }
           else
           {
               tableName = "2";
           }
       Program.ExecuteImmediate("select x from y " + "where a  +  b > 2");
       Program.ExecuteImmediate("select x from name_" + tableName);
       }
   }
```

Первый динамически формируемый запрос получается с помощью конкатенации двух строк, а второй — с помощью условного оператора if.

Для обычных языков программирования существуют интегрированные среды разработки, которые предоставляют разработчику набор различной функциональности, упрощающей процесс разработки, например, автодополнение, рефакторинги и дополнительные статические проверки. Для встроенных языков такая функциональность была бы также полезной.

С другой стороны, при проведении реинжиниринга программного обеспечения необходимо сохранять связь преобразованного кода программы с исходным кодом.

Это важно уметь делать, потому что с программой работали люди и было бы странным, если бы они увидели в ней новые имена переменных, функций и другие конструкции языка.

Поэтому возникает необходимость в разработке инструмента, который умеет проводить обработку встроенных языков, а также использоваться в качестве инструмента для проведения реинжиниринга программного обеспечения. Одним из компонентов этого инструмента будет являться генератор абстрактных лексических анализаторов, который по заданной спецификации обрабатываемого языка автоматически генерит соответствующий анализатор. Главная особенность абстрактного анализа — не вычислять все возможные значения динамически формируемой строки, а работать с компактным представлением множества значений. В данном случае это будет граф, являющийся представлением динамически формируемого выражения. Таким образом каждому пути в графе будет соответствовать некоторое возможное значение динамического запроса. Результатом работы анализатора также является граф, ребро которого содержит токен. В ходе токенизации входного графа необходимо сохранить привязку частей динамически формируемого выражения к исходному коду и привязку лексических единиц внутри каждой части, которая понадобится при дальнейшем разборе полученного графа.

Таким образом, целью данной работы является сохранение привязки к исходному коду и доработка существующего генератора абстрактных лексических анализаторов, реализованного на базе инструмента FsLex [3] в рамках проекта YaccConstructor [7], [8]. В рамках этого же проекта разрабатывается платформа для поддержки встроенных языков. Демонстрация возможностей разрабатываемой платформы представлена в виде плагина к ReSharer, которому также нужны координаты токена в исходном коде.

В данной работе также представлены результаты замеров работы лексера инструментов YaccConstructor и Alvor (плагин к среде разработки Eclipse, предназначенный для статической валидации SQL-выражений встроенных в код на Java, [1]), а также инструмент, который позволяет получить эти результаты.

1. Постановка задачи

В рамках данной работы были поставлены следующие задачи:

- Доработать существующий генератор абстрактных лексических анализаторов так, чтобы он осуществлял привязку частей динамически формируемого выражения к исходному коду и привязку лексических единиц внутри каждой части.
- Реализовать обработку “рваных” токенов, то есть случаев, когда токен находился на двух и более ребрах входного графа, являющегося компактным представлением множества динамически формируемых выражений. Пример такого случая можно увидеть на рис. 1.

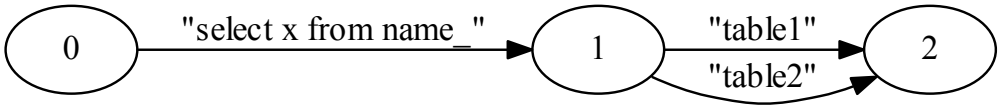


Рис. 1: Пример случая с ”рваными” токенами

- Осуществить корректную передачу координат токена в ReSharper.
- Сравнить полученный инструмент с его аналогами.

2. Обзор

Для работы с динамически формируемыми строками существует ряд инструментов, основные особенности которых представлены ниже. Последние три инструмента интересны тем, что они имеют абстрактный анализ — на вход подается компактное представление множества динамически формируемого выражения. Это может быть data-flow уравнение, регулярное выражение или граф. Первые два инструмента отвечают на вопрос — является ли обрабатываемый язык подмножеством какого-нибудь языка.

- **PHP string analyzer** [6] — это статический анализатор для строк, порождаемых программами на PHP. Он аппроксимирует значения таких строк некоторой контексно-свободной грамматикой.

- **Java String Analyzer** [4] — это инструмент для анализа формирования строк и строковых операций в программах на Java. Для каждого строкового выражения он строит конечный автомат, представляющий приближенное значение всех значений этого выражения, которые могут быть получены во время выполнения.

- **Alvor** [1] — это плагин к среде разработки Eclipse, предназначенный для статической валидации SQL-выражений встроенных в код на Java. Найденные в коде SQL-запросы проверяются на основе SQL-грамматики. Компактным представлением множества динамически формируемых выражений для лексера является регулярное выражение.

- **статья Kyung-Goo Doh, Hyunha Kim, David A. Schmidt** [2] — предложили алгоритм абстрактного анализа на примере статической валидации динамически генерируемого HTML в PHP. Абстрактный парсер на вход принимает data-flow уравнение и LALR(1) – таблицу, в качестве результата выдает абстрактное синтаксическое дерево, пригодное для дальнейшего анализа. Парсер работает на символах, не на токенах.

- **инструмент YaccConstructor** [7], [8] — модульный инструмент для проведения лексического анализа и синтаксического разбора. Также является платформой для исследования и разработки генераторов абстрактных лексических и синтаксических анализаторов. Структура инструмента представлена на рис.2

  В рамках проекта YaccConstructor разрабатывается платформа для поддержки встроенных языков. Демонстрация возможностей разрабатываемой платформы представлена в виде плагина к ReSharper. Например, с помощью него можно подсвечивать ошибки (рис.3) в Microsoft Visual Studio IDE.

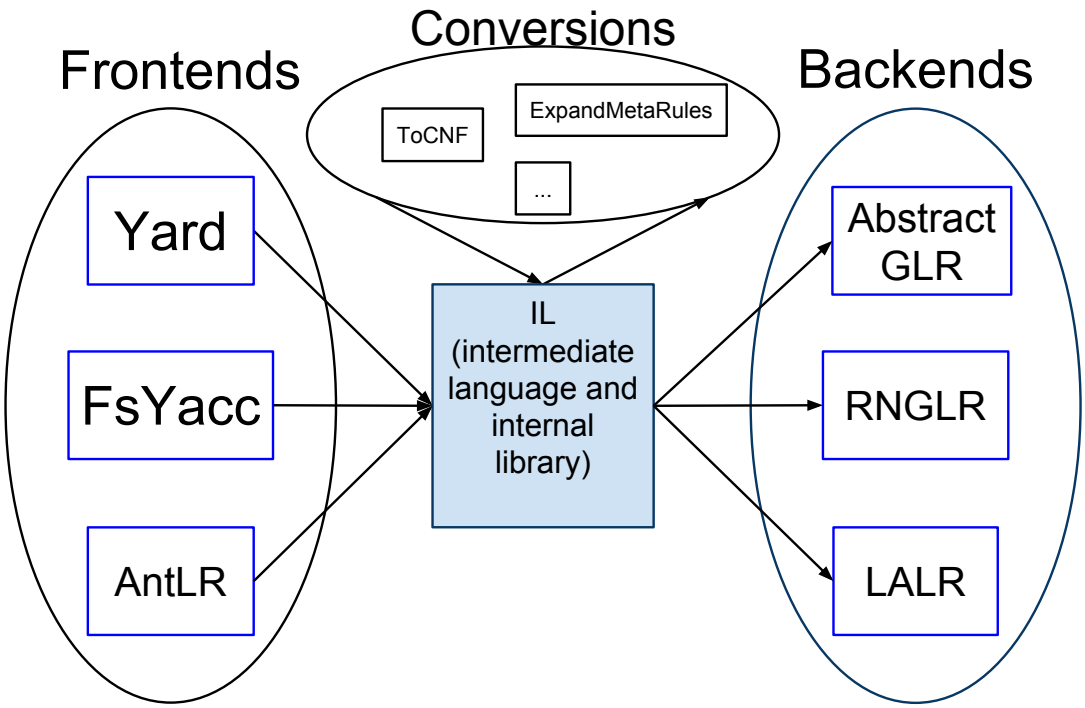


Рис. 2: Структура инструмента YaccConstructor

Для того чтобы это сделать, необходимы координаты токена в исходном коде, которые можно получить, сохранив привязку частей динамически формируемого выражения к исходному коду и привязку лексических единиц внутри каждой части при проведении абстрактного лексического анализа.

```
private void Go(int cond)
{
    string tableName;
    if (cond > 0)
    {
        tableName = "1";
    }
    else
    {
        tableName = " 2";
    }
    Program.ExecuteImmediate("select x from y " + "where a ++ b > 2");
    Program.ExecuteImmediate("select x from name" + tableName);
}
```

Рис. 3: Подсветка ошибок в Microsoft Visual Studio IDE

3. Реализация

3.1. Генератор абстрактных лексических анализаторов

В рамках проекта YaccConstructor был реализован генератор абстрактных лексических анализаторов, который по спецификации геннерирует лексический анализатор для обрабатываемого языка. В качестве основы для него был взят генератор лексических анализаторов для F# — FsLex [3]. Структура генератора представлена на рис.4

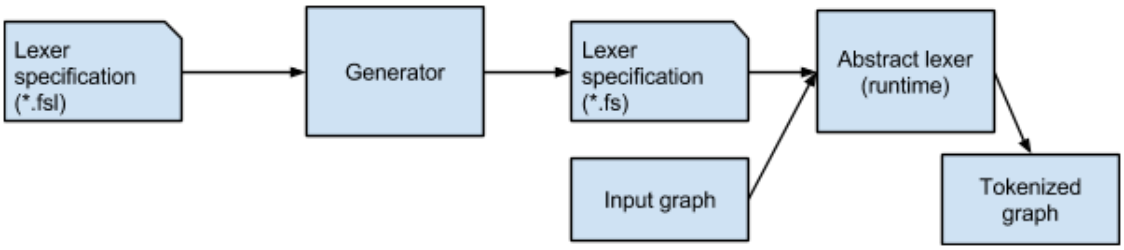


Рис. 4: Структура генератора в YaccConstructor

Входной структурой для анализатора является граф — компактное представление множества динамически формируемых выражений, которые могут быть получены посредством конкатенации строковых констант и выражений встроенного языка в цикле, условных выражениях и другими способами. Ограничения на граф — граф является DAG-ом (направленный ациклический граф), у которого одна стартовая и одна конечная вершина. Для конечной вершины верно, что из нее не выходит ни одна дуга. Это достигается явным добавлением дуги с токеном, обозначающим конец ввода. Абстрактный лексический анализатор основан на конечном преобразователе. Это конечный автомат, который может выводить конечное число символов для каждого входного символа. На основе него лексер переводит входной граф в граф, содержащий соответствующие спецификации токены.

Рассмотрим пример получения входных данных для абстрактного лексического анализатора из динамически формируемого выражения, представленного на рис.3:

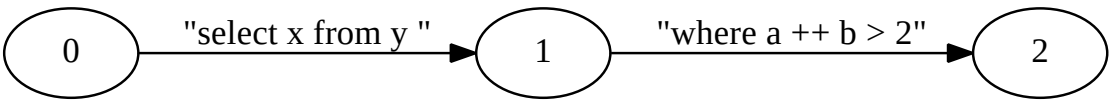


Рис. 5: Граф для первого запроса

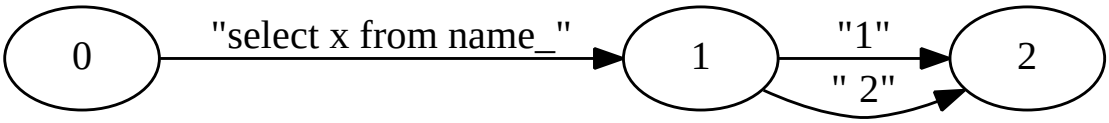


Рис. 6: Граф для второго запроса

Поскольку динамически формируемые выражения могут быть получены с помощью конкатенации строковых констант и выражений встроенного языка в цикле, условных выражениях и другими способами, то возникают ситуации с “рваными” токенами. В реализованном инструменте был предложен следующий алгоритм для обработки таких случаев:

(a) На вход подаётся граф, составленный на основе дерева разбора исходного кода программы: на ребрах — фрагменты строк кода на встроенном языке, вершинам соответствуют случаи конкатенации строк.

(b) Входной граф преобразовывается к виду, удобному для лексического анализа: каждое ребро входного графа разбивается на последовательность новых ребер, метки которых содержат не более одного символа из строки исходного ребра.

(c) Запускается процедура токенизации на преобразованном графе, заключающаяся в вычислении множества возможных состояний лексического анализатора в каждой вершине входного графа. В случае, если в какой-либо вершине получено финальное состояние, то из накопленной строки формируется токен согласно соответствующему пользовательскому коду, и в множество состояний для данной вершины заносится стартовое состояние. В результате данной процедуры получается новый граф, на ребрах которого находятся накопленные в результате работы токены, соответствующие грамматике анализируемого встроенного языка

Рассмотрим пример входного графа с “разрывными” токенами:

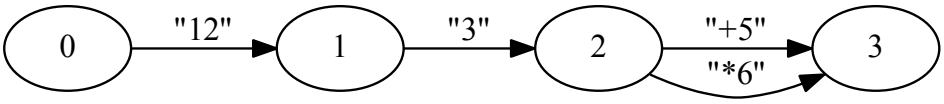


Рис. 7: Пример графа с “разрывными” токенами

Результат работы абстрактного лексического анализатора:

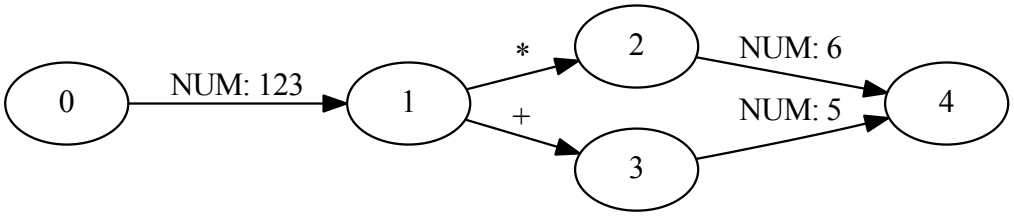


Рис. 8: Результат работы лексера в случаи “разрывных” токенов

Существующий генератор абстрактных лексических анализаторов был модифицирован таким образом, чтобы он умел корректно обрабатывать случаи, когда на дугах входного графа находятся пробелы и комментарии. В связи с этим добавился этап построения eps-замыкания токенизированного графа. Пример такого графа представлен на рис. 9. Результат работы лексера представлен на рис.10

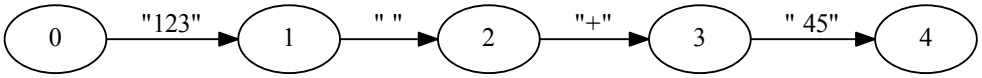


Рис. 9: Пример графа, на ребрах которого находятся пробелы

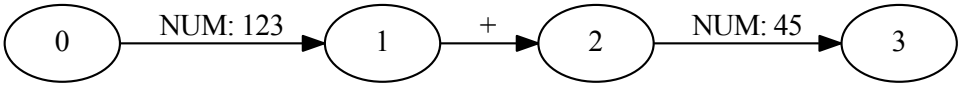


Рис. 10: Результат работы лексера, когда ребра графа содержали пробелы

Также в ходе разбора входных данных абстрактный лексический анализатор сохраняет привязку частей динамически формируемого выражения к исходному коду и привязку лексических единиц внутри каждой части. Для этого к существующей структуре, которая хранила информацию о токене, было добавлено новое поле Positions. Оно хранит для каждого символа токена строку из которой

он получен и позицию этого символа внутри этой строки. Для корректного вычисления позиции токена в случае “рваных” токенов была добавлена глобальная структура LexBuffer, которая хранит состояние графа после каждого прочитанного символа.

Результат работы абстрактного лексического анализатора с сохранением привязки к исходному коду на примере рис.8 будет иметь вид:

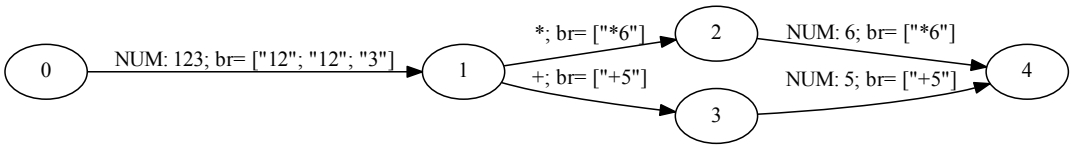


Рис. 11: Результат работы лексера с сохранением привязки к исходному коду

Одним из примеров использования данной привязки является позиционирование токена в исходном коде, поскольку она позволяет однозначным образом определить в какой строке и на какой позиции внутри этой строки находится токен. Это нужно, например, для подсветки синтаксиса.

```
    {
        static int Calculate(bool cond)
        {
            var expr = "1+(2";
            var baseMult1 = "4";
            var baseMult2 = "3";
            if (cond)
            {
                expr = expr + "*" + baseMult1 + ")";
            }
            else
            {
                expr = expr + "/" + baseMult2 + ")";
            }
            return Program.Eval(expr);
        }
```

Рис. 12: Подсветка синтаксиса в Microsoft Visual Studio IDE

3.2. Инструмент для тестирования лексера инструментов YaccConstructor и Alvor

В обзоре был представлен ряд инструментов, которые работают с динамически формируемыми выражениями. Поскольку только инструмент Alvor решает схожую задачу с инструментом YaccConstructor и имеет открытые исходники, то сравнение происходило только с ним. В инструменте Alvor, так же как и в YaccConstructor, отдельно выделены этапы лексического анализа и синтаксического разбора. Входными данными для лексера Alvor является регулярное выражение — компактное представление множества значений динамически формируемого выражения. Для тестирования лексера инструментов YaccConstructor и Alvor был реализован инструмент, который предоставляет пользователю следующий набор функциональности:

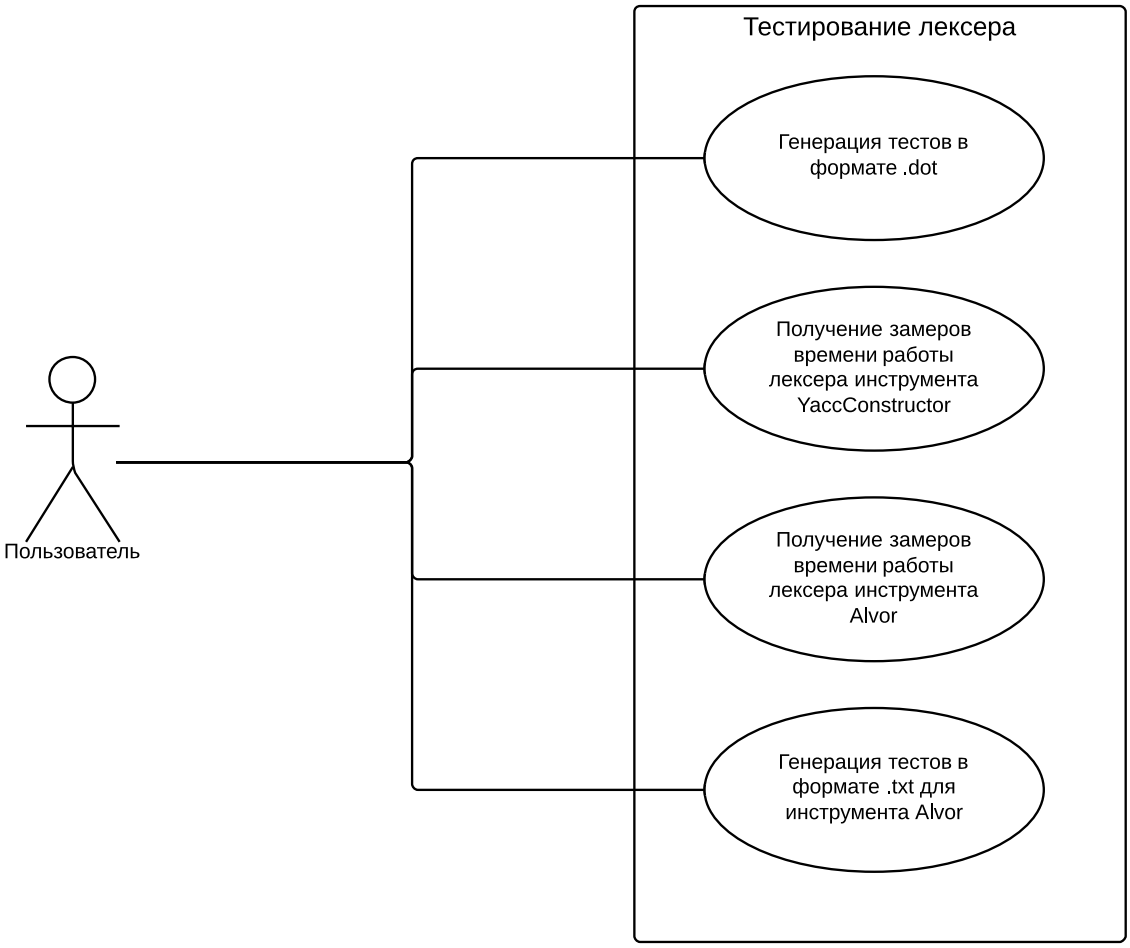


Рис. 13: Диаграмма прецедентов

Диаграмма классов реализованного инструмента, представленная на рис. 14, описывает генератор тестов, который по двум заданным параметрам m(кратность ребер) и n(длина цепочки) выдает граф в формате .dot, генера-

тор замеров времени работы лексера YaccConstructor и Alvor, генератор тестов, который переводит данные, представленные в формате .dot, в регулярное выражение, записанного потом в текстовый файл.

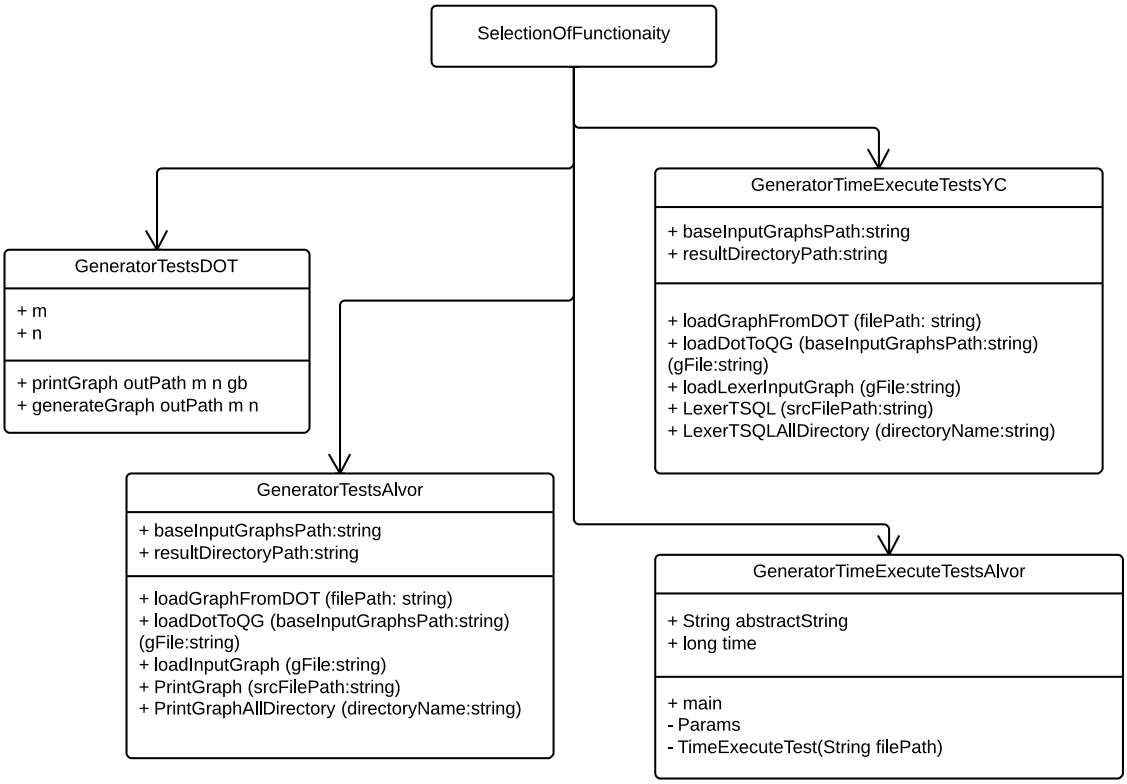


Рис. 14: Диаграмма классов

Пример графа, который может быть получен с помощью генератора тестов в формате.dot с параметрами m = 2 и n = 3, представлен на рис. 15 (a и b используются в качестве ограничителя). Для лексера Alvor данный граф запишется в следующее регулярное выражение, которое можно увидеть на рис. 16.

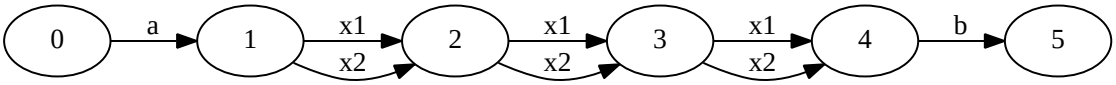


Рис. 15: Пример графа с параметрами m = 2 и n = 3

{ "x1", "x2" } { "x1", "x2" } { "x1", "x2" }

Рис. 16: Пример регулярного выражения с параметрами m = 2 и n = 3

Компоненты реализованного инструмента можно использовать независимо друг от друга. Например, для лексера в качестве входных данных можно использовать тесты, которые были получены другим генератором тестов. Полученные

результаты замеров работы лексера можно визуализировать с помощью R-studio с целью получения графика, например, зависимости времени от сложности лексического разбора входных данных. Для визуализации файлов с расширением .dot можно использовать Graphviz.

4. Апробация

Большинство программ написанные SQL содержат в себе вложенные конструкции языка, которые формируются с помощью условных выражений, конкатенаций и прочими способами. Для анализа программы необходимо рассмотреть все эти случаи. Поэтому для тестирования работы абстрактного лексического анализатора были выбраны тесты, которые уже были описаны в пункте "Инструмент для тестирования лексера инструментов YaccConstructor и Alvor".

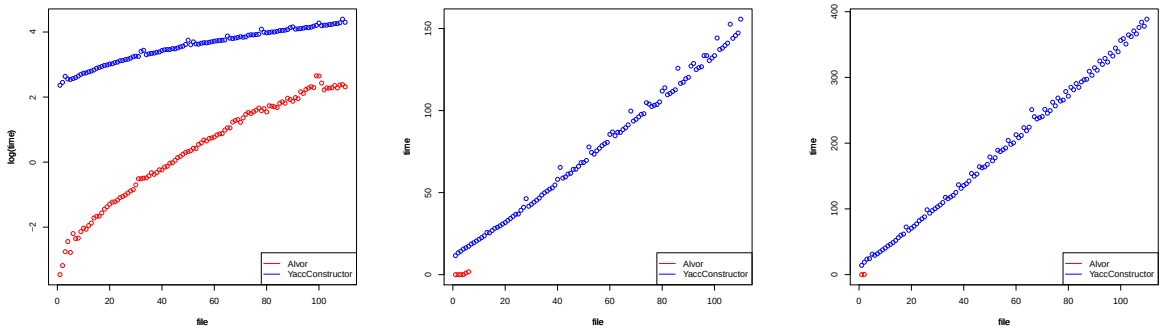
Так как во встроенных языках возможны случаи с "разрывными" токенами — когда токен находился на двух и более ребрах исходного графа, то были подобраны тесты и с "разрывными" токенами. Разумным считается, что длина цепочки не может быть большой — на практике редко встретишь, чтобы идентификатор был составлен при помощи большого числа конкатенаций. Структура теста такая же как в случаи безразрывных токенов.

В обзоре был представлен ряд инструментов, которые работают с динамически формируемыми выражениями. Сравнение времени работы лексера было проведено для инструментов YaccConstructor и Alvor. Замеры времени работы лексера указанных инструментов проводились на машине со следующими техническими характеристиками: Intel(R) Core(TM) i7-3537U CPU @ 2.00GHz 2.50 GHz, 4,00 Гб оперативной памяти, процессор x64.

Были рассмотрены случаи с безразрывными токенами для m = 1, 2, 5 и n = 8..228 с шагом 2. Результаты представлены на рис.17 соответственно. Лексер инструмента Alvor разбирает данные быстрее, чем лексер инструмента YaccConstructr, но Alvor не обрабатывает случаи, когда на вход ему было подано регулярное выражение, содержащее большое количество альтернатив, в то время как YC аналогичные данные, только представленные в виде графа, обработал.

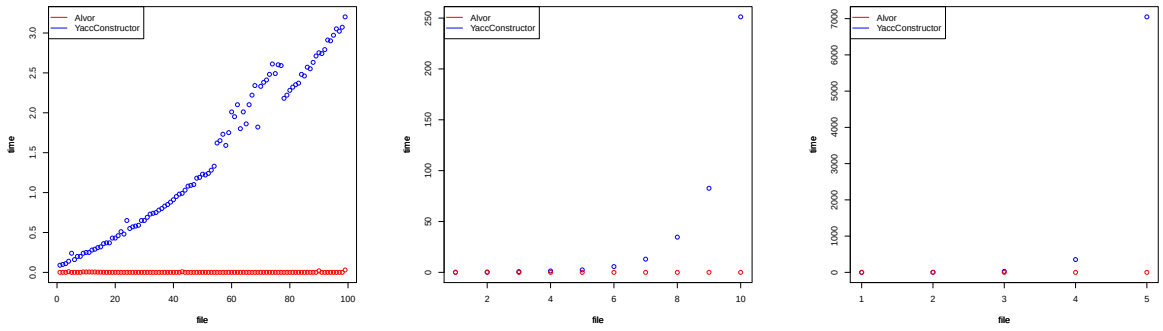
Также были рассмотрены случаи с "разрывными" токенами для m = 1, 2, 5 и n = 1..100. Результаты представлены на рис.18. Также лексер инструмента Alvor обработал входные данные быстрее, чем лексер инструмента YaccConstructor. Длина цепочки входного графа, которую разбирают лексеры этих инструментов, примерно одинаковая. На больших значениях параметоров m и n графа, лексеры инструментов завершают работу с ошибкой о переполнении стека.

На рис.19 и 20 можно увидеть график, на котором показаны результаты замеров работы лексера инструментов YaccConstructor и Alvor соответственно. Видно, что время разбора лексера увеличивается от сложности входных данных. В случае обычных литералов для инструмента YC почти линейно, а в случае "разрывных" токенов экспоненциально. Для инструмента Alvor для обычных токенов экспоненциально, поскольку они перебирают все варианты и возвращают только уникальные результаты лексического разбора.



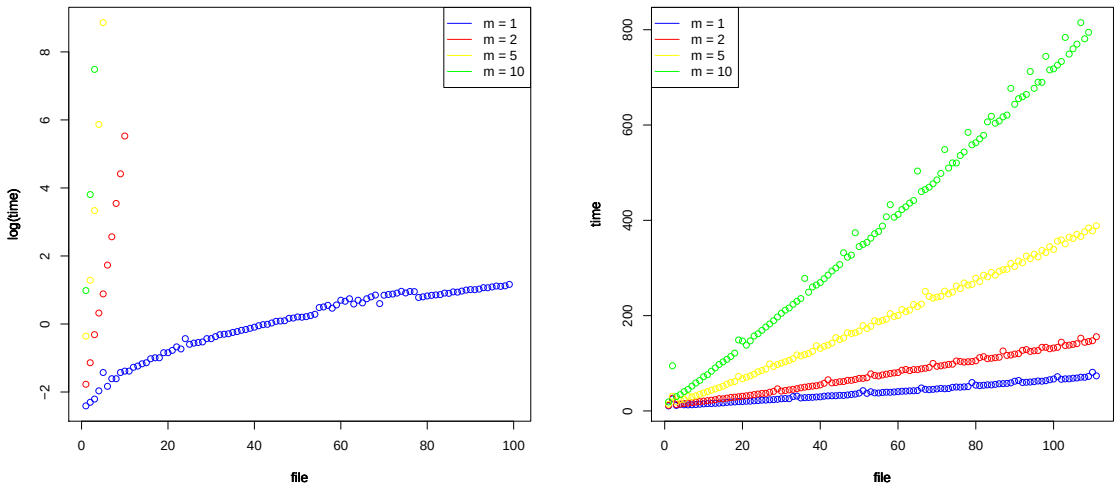
(a) График при m = 1 (b) График при m = 2 (c) График при m = 5

Рис. 17: Графики для YC и Alvor



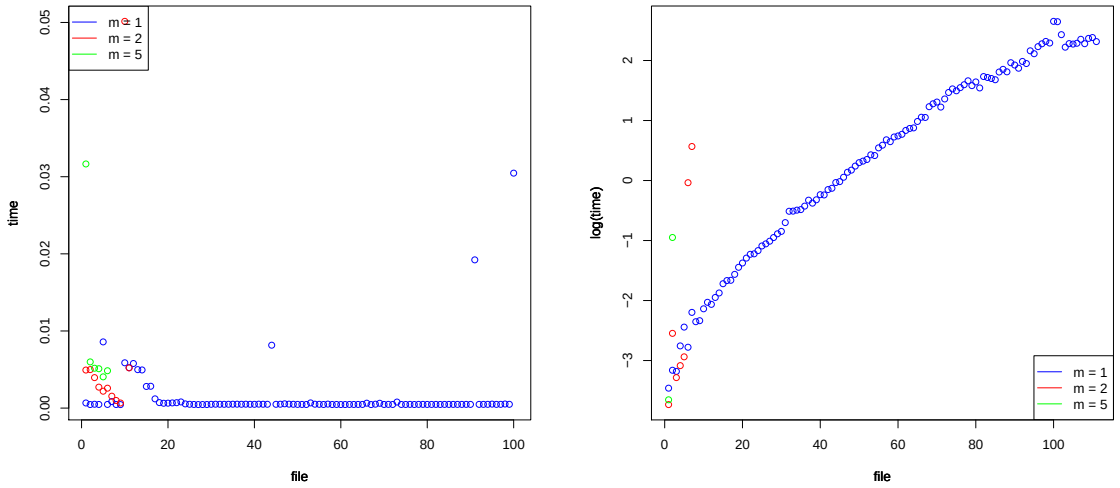
(a) График при m = 1 (b) График при m = 2 (c) График при m = 5

Рис. 18: Графики для YC и Alvor в случае с ”разрывными” токенами



(a) с ”разрывными” токенами (b) с неразрывными токенами

Рис. 19: Графики для YC



(a) с ”разрывными” токенами (b) с неразрывными токенами

Рис. 20: Графики для Alvor

Заключение

В ходе выполнения данной работы были получены следующие результаты:

- Реализовано сохранение привязки частей динамически формируемого выражения к исходному коду и привязки лексических единиц внутри каждой части
- Осуществлена корректная передача координат токена в ReSharper
- Доработан генератор абстрактных лексических анализаторов так, чтобы он корректно обрабатывал комментарии и пробелы.
- Изучены аналоги лексических анализаторов
- Реализован инструмент для тестирования лексеров YaccConstructor и Alvor
- Проведен анализ времени работы генераторов YaccConstructor и Alvor на одинаковых входных данных
- Результаты работы представлены на конференции «Технологии Microsoft в теории и практике программирования. Новые подходы к разработке программного обеспечения по технологиям Microsoft и EMC.» в секции ”Программная инженерия: инструментальные средства и технологии проектирования и разработки”. Доклад был награжден дипломом второй степени. Тезисы данной работы были опубликованы в сборнике материалов конференции.
- Принята статья “Инструментальная поддержка встроенных языков в интегрированных средах разрботки” на Workshop on Science Intensive Applied Software.

Дальнейшее направление работы

На практике не каждый случай использования встроенных языков может быть представлен в виде направленного ациклического графа. В дальнейшем хочется научиться поддерживать графы, содержащие циклы. Также планируется в инструменте YaccConstructor реализовать конечный преобразователь [5], который будет поддерживать строковые операции, например, replace.

Список литературы

[1] Alvor. — URL: http://code.google.com/p/alvor/.

[2] Doh Kyung-Goo, Kim Hyunha, Schmidt David A. Abstract LR-Parsing // Formal Modeling: Actors, Open Systems, Biological Systems 2011:. — P. 90 – 109. — URL: http://santos.cis.ksu.edu/schmidt/talcottfest.pdf.

[3] FsLex. — URL: http://fsharppowerpack.codeplex.com/wikipage?title=FsLex%20Documentation.

[4] Java String Analyzer. — URL: http://www.brics.dk/JSA/.

[5] Mehryar Mohri. Finite-State Transducers in Language and Speech Processing. — 1997. — URL: http://www.cs.nyu.edu/.

[6] PHP String Analyzer. — URL: http://www.score.cs.tsukuba.ac.jp/~minamide/phpsa/.

[7] YaccConstructor. — URL: https://code.google.com/p/recursive-ascent/.

[8] Кириленко Я.А, Григорьев С.В., Д.А. Авдюхин. Разработка синтаксических анализаторов в проектах по автоматизированному реинжинирингу информационных систем // Научно-технические ведомости Санкт-Петербургского государственного политехнического университета. Информатика. Телекоммуникации. Управление. — 2013. — no. 174. — P. 94 – 98. — URL: http://ntv.spbstu.ru/telecom/article/T3.174.2013_11/.