Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Алефиров Алексей Андреевич

Непосредственная поддержка грамматик в расширенной форме Бэкуса-Наура в генераторах синтаксических анализаторов

Бакалаврская работа

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Терехов А.Н.

Научный руководитель:
ст. преп. Григорьев С.В.

Рецензент:
инженер-программист в ООО «ИнтеллиДжей Лабс» Беляков А. М.

Санкт-Петербург
2014

Saint-Petersburg State University

Mathematics & Mechanics Faculty

Department of Software Engineering

Aleksei Alefirov

Generator of parsers for context free grammars with built-in support for EBNF

Bachelor’s Thesis

Admitted for defence.
Head of the chair:
professor A.N. Terekhov

Scientific supervisor:
sen. lect. S.V. Grigoriev

Reviewer:
software developer at “IntelliJ Labs Co.Ltd” A.M. Belyakov

Saint-Petersburg
2014

Оглавление

1 Введение.. 4
2 Постановка задачи 6
3 Обзор 7
3.1 Алгоритмы.. 7
3.2 Технологии 11
4 Алгоритмическое решение.. 13
5 Детали реализации 16
6 Апробация.. 17
7 Заключение 24
Список литературы 25

1 Введение

В решении задачи синтаксического анализа преуспели генераторы синтаксических анализаторов - инструменты, принимающие на вход грамматику, описанную в определенной форме, и порождающие на et основе парсер, соответствующий ей.

Удобным способом определения грамматики языка программирования является расширенная форма Бэкуса-Наура(EBNF) [10] – правила в такой грамматике в правых частях содержат регулярные выражения. На практике, использование EBNF позволяет упростить и сократить описание языка, сделать его более понятным.

Однако многие современные инструменты не поддерживают EBNF-конструкции, либо же трансформируют исходные грамматики в классическую форму Бэкуса-Наура и, зачастую, еще и в форму, пригодную для дальнейшей генерации парсера (например, нормальную форму Грейбах [6]), изменяя правила исходной грамматики и добавляя в нее множество новых нетерминалов и новых продукций. Это означает, что пользователи таких инструментов получают выводы сгенерированного парсера, которые им трудно связать с исходной грамматикой и, как следствие, понять. Вывод в терминах исходной грамматики является естественным удобством для пользователя. Кроме того, увеличение множества символов грамматики увеличивает двумерные таблицы состояний и переходов, используемые в генерируемых парсерах, что влияет на производительность этих парсеров. Также увеличение количества продукций влияет на производительность как генератора парсеров, так и порождаемых парсеров.

Проблема восприятия вывода порожденных синтаксических анализаторов решается с помощью сохранения связи исходной грамматики и ее трансформированной формы и обратным восстановлением требуемого и удобного пользователю вывода из полученного с использованием этой связи, что добавляет задачу к работе порожденных парсеров и, таким образом, тоже влияет на их производительность. Проблема производительности ставит вопрос о решении генерации синтаксических анализаторов с сохранением EBNF исходных грамматик. Возникает гипотеза о том, что реализация такого решения будет не только иметь удобный для пользователей вывод порожденных парсеров, но и выигрывать в производительности перед классическими генераторами на множестве грамматик, устройство которых использует преимущества EBNF.

В рамках данной работы ставится цель разработки генератора синтаксических анализаторов, позволяющего работать с контекстно-свободными грамматиками, описанными в расширенной форме Бэкуса-Наура, и не производящего над ними трансформаций, сохраняя их исходную форму.

2 Постановка задачи

Целью данной работы является создание генератора синтаксических анализаторов, которые сохраняют исходную расширенную форму Бэкуса-Наура входных грамматик. Для достижения этой цели были поставлены следующие задачи:

- изучить алгоритмы и решения в области синтаксического анализа, выбрать и обосновать выбор алгоритмов, на основе которых можно разработать решение, связанное с поставленной целью;
- разработать алгоритмическое решение;
- по разработанному решению создать программу-генератор синтаксических анализаторов с сохранением EBNF и программу-интерпретатор порождаемых синтаксических анализаторов на базе проекта YaccConstructor;
- провести апробацию: протестировать созданные программы и сравнить результаты тестирования с показателями аналогичных решений, подразумевающих трансформации исходной формы входной грамматики, которые устраняют изначальную выразительность описания в EBNF.

3 Обзор

Алгоритмы генерации синтаксических анализаторов являются областью знаний, ориентированной на мощное прикладное применение в индустрии программной инженерии, и, вместе с тем, подобластью информатики, подкрепляемой научными исследованиями. Поэтому обзор состоит из двух частей – теоретические алгоритмы, созданные в области науки, связанной с темой данной работы, и технологии, использующие эти алгоритмы.

3.1 Алгоритмы

В области распознавания расширенных грамматик было разработано множество теоретических решений [9]. Наибольшее развитие получило направление LR-анализа, в котором отметилась статья [12], за которой последовали различные вариации [11][13][15], использующие оригинальную идею, описанную ниже.

Состояния LR-анализатора по расширенной грамматике создаются аналогично классической генерации LR-парсеров:

- регулярные правые части исходной грамматики преобразуются в эквивалентные конечные автоматы;
- пара, состоящая из номера продукции и номера состояния конечного автомата этой продукции, представляет собой в аналог LR-ситуации [5];
- тройка, состоящая из пары - LR ситуации, и множества символов предпросмотра является элементом, из которых

посредством операций Closure и Goto строятся состояния LR-анализатора.

Проблема, связанная с тем, что в правых частях расширенных грамматик находятся выражения, а не обычные цепочки символов, заключается в том, что такие правые части не обладают фиксированной длины, что используется в классических LR алгоритмах, например, для совершения свертки, когда со стека просто снимается количество символов, соответствующее длине продукции, согласно с которой происходит свертка. Авторы статьи в качестве решения проблемы предлагают складывать и выталкивать со стека номера правил, соответственно которым происходит порождение входной цепочки символов.

Основная идея алгоритма состоит в том, чтобы определять, когда начинается распознавание новой правой части грамматики с тем, чтобы положить правило на стек. Для этого элементы состояния парсера делятся на два множества - так называемое “ядро” состояние LR парсера (множество “главных” элементов) и элементы, “выведенные” из ядра посредством операции Closure. Если переход из одного состояния анализатора в другое связан с выведенным элементом первого состояния, это означает, что переход ассоциирован с началом распознавания новой правой части, и что следует положить правило на стек; когда же переход связан с главным элементом, то состояние стека не меняется. Свертка по правилу означает снятие этого правила с вершины стека и переход по его нетерминалу - его правой части.

Однако такой алгоритм позволяет построить анализаторы далеко не для всех расширенных LR грамматик. Возможны случаи, когда есть и главные, и выведенные элементы, связанные с одним переходом из состояния парсера. Такая ситуация называется *конфликтом стека.*

Авторы приводят грамматику Г1, для которой LR-автомат имеет конфликт стека, она описана ниже в нотации YARD [26]:

```
[<Start>]
s : A* B
s : A a C
a : A*
```

В оригинальной статье для разрешения таких ситуаций приводится алгоритм, который трансформирует исходную грамматику с конфликтом, добавляя новые правила и нетерминалы. Из всех последующих решений данной проблемы наиболее эффективным и интересным с точки зрения сохранения исходной формы грамматики являются алгоритмы, приведенные в статье [11]. Первый из этих алгоритмов, как пишут авторы, создает генератор, покрывающий почти полностью класс расширенных LR грамматик, второй алгоритм покрывает весь класс расширенных LR грамматик, но производит небольшую дополнительную работу.

В направлении LL-анализа языков, описанных грамматиками в расширенной форме Бэкуса-Наура описанные в [8] и [17]. В [17] авторы также складывают правила на стек и производят свертки согласно этим меткам. Авторы [8] предлагают использовать стек не явно, а представленным множеством поддеревьев частичного дерева синтаксического разбора.

Стоит отметить, что приведенные статьи имеют очевидные ограничения по классу входных грамматик - например, они должны быть однозначными. Однако в исходных задачах, таких как создание генератора парсеров по существующей документации языка, входные грамматики могут быть неоднозначными. В задачах автоматизированного реинжиниринга программ [2] возникает проблема устаревших языков, для которых трудно задать однозначные грамматики. Таким образом, особый

интерес представляет решение проблемы для неоднозначных расширенных контекстно-свободных грамматик. В области работы с неоднозначными грамматиками наиболее ярко отметился алгоритм Томиты обобщенного синтаксического анализа GLR [16]. Этот эффективный алгоритм позволяет разрешать конфликты LR-анализатора, производя “разветвление” стека. Стек алгоритма GLR представлен так называемым стеком, структурированным в виде графа (GSS). В действительности, это граф, вершины которого - состояния LR-автомата, достигаемые входе анализа строки и разделенные на “рубежы” - для n символов входной цепочки n+1 рубеж - начальный и по рубежу соответственно очередному входному символу. Вершины соединены дугами соответственно переходам по символам грамматики в обратном направлении; по данным дугам происходит возвращение входе произведения сверток.

Надо отметить, что классический алгоритм GLR обладает недостатком класса входных грамматик - в различных модификациях он работает некорректно с грамматиками с левой или правой рекурсиями. Данную проблему решил алгоритм RNGLR [14] - он успешно работает со всеми неоднозначными грамматиками. Простая модификация классического алгоритма состоит в том, чтобы производить свертки, когда оставшаяся непрочитанная часть тела правила порождает пустую строку.

Кроме того, авторы RNGLR приводят способ создания вывода для GLR-алгоритма - это разделенный сжатый лес синтаксического анализа (SPPF). При работе GLR-анализа рассматривает несколько вариантов разбора входной строки, и SPPF - это подходящая компактная структура для хранения нескольких деревьев разбора.

3.2 Технологии

Среди генераторов синтаксических анализаторов наибольшее распространение достиг YACC [24] и его различные “потомки”, такие как Ocamlyacc [22], Bison [19] и прочие. Это семейство LALR генераторов парсеров, которые в основном принимают на вход грамматики, описанные в классической BNF форме. Версии YACC, принимающие грамматики в EBNF форме, производят преобразования над ними для генерации парсеров, не сохраняя, таким образом, исходную расширенную форму Бэкуса-Наура.

Также среди LL-решений задачи генерации синтаксических анализаторов преуспел и получил большое развитие инструмент ANTLR [18]. Помимо того, что он поддерживает EBNF входных грамматик, инструмент обладает всеми преимуществами LL-техники генерации парсеров: простое отслеживание ошибок вывода, простое осуществление “нагрузки” анализа различными семантическими конструкциями и функциями. Однако, у LL-анализаторов есть известные недостатки [3]: ограниченный класс грамматик (что, однако решено в ANTLR - создателями инструмента заявлена поддержка LL(*) грамматик) и возможное долгое время работы.

Как было замечено выше, целевой язык синтаксического анализа может быть удобно описать не просто грамматикой в EBNF-форме, но также неоднозначной грамматикой, и тут преуспели GLR-решения генерации синтаксичеких анализаторов. Инструменты, реализующие эти решения и получившие распространение, такие, как Bison [19], Jison [21], SDF/SGLR [23], Elkhound [20] и пр., не заявлены, как сохраняющие исходную форму входных грамматик без изменений.

В 2010 году на кафедре Системного Программирования СПБГУ Семеном Григорьевым была написана работа [4], целью которой в том числе являлось создание генератора синтаксических анализаторов, который решал бы задачи работы с неоднозначными контекстно-свободными грамматиками и работы с EBNF-грамматиками без изменения их формы. Работы была выполнена, но результат обладает явными недостатками:

- непростой для понимания алгоритм анализа;
- класс грамматик, поддерживаемый реализованным инструментом, довольно слабый - LR(0);
- код генерируемых инструментом парсеров имеет сложную структуру, что усложняет понимание их работы и отладку.

Работа является интересным экспериментом и была реализована как часть проекта YaccConstructor [25]. Данный проект занимается разработкой одноименного программного инструмента YaccConstructor, предназначенного для конструирования синтаксических анализаторов и обработки грамматик.

Позднее в рамках того же проекта был реализован алгоритм RNGLR [1], работающий с обычными контекстно-свободными грамматиками.

4 Алгоритмическое решение

Алгоритмическое решение проблемы генерации синтаксических анализаторов состоит из двух этапов:

- создание непосредственно генератора синтаксических анализаторов и
- создание интерпретатора генерируемых синтаксических анализаторов.

В качестве решения генерации синтаксических анализаторов за основу был взят классический алгоритм генерации LR-анализаторов [7]. Для генерации таблиц LR-анализа необходимо было создать промежуточное представление для грамматик с регулярной правой частью. Таким представлением для правой части каждого правила является эквивалентный недетерминированный конечный автомат, алгоритм создания которых описан у Хопкрофта [6]. Выше было описано, как по грамматикам с НКА в правых частях правил создаются таблицы LR-анализа.

Были применены две модификации таблиц состояний и переходов:

- добавление сверток для LR-ситуаций, когда после точки находится цепочка символов грамматики, порождающая пустую строку; эта модификация используется алгоритмом RNGLR;
- переходы между состояниями помечены двумя множествами правил, согласно которым происходит переход, и согласно которым во время распознавания строки будет происходить свертка; первое множество состоит из правил, возможное порождение правых частей которых уже началось и которые не надо складывать на стек, второе множество, наоборот, состоит из правил,

возможное порождение правых частей которых начинается согласно с переходом, и которые следует положить на стек.

В качестве решения интерпретации сгенерированных парсеров за основу был взят алгоритм RNGLR. Была изменена работа стека, структурированного в виде графа, который строится при выполнении алгоритма RNGLR. Теперь дуги этого графа метятся парами множеств соответственно переходам из LR-таблиц, порожденных программой-генератором. Это производится с целью выполнения сверток - в классическом GLR и RNGLR алгоритмах, работающих с обычными контекстно-свободными грамматиками, а значит, с правыми частями фиксированной длины, свертки производятся возвращением по всем путям, идущим из текущей вершины, длины, равной длине правым частям правила, согласно которым они производятся. В случае работы с правилами с регулярной правой частью такой возможности нет, поэтому для сверток используются метки дуг GSS. Это происходит следующим образом: допустим, есть вершина *v* стека, соответствующая состоянию LR-анализатора, в котором по предпросмотру следующего символа происходит свертка по правилу *j*. Выполнение свертки производится согласно следующему алгоритму:

В начале работы алгоритма создается пустое множество вершин S, обойденных во время алгоритма.

- Для каждой дуги *e*, выходящей из *v* и вершина-конец *u* которой не принадлежит S, рассмотреть множества правил, которыми помечена *e*; положить *v* в S;
- Если j принадлежит множеству правил, которые были сложены на стек, то следует произвести свертку к вершине *u*, затем

произвести переход из *u* согласно символу-нетерминалу, находящемуся в левой части правила *j*; если *j* принадлежит множеству правил, несложенных на стек, то следует повторить алгоритм, начиная с шага 1 для вершины *u*.

Таким образом, модифицированная работа со стеком позволяет производить свертки аналогично тому, как это происходит в классическом GLR-алгоритме. Идея такой работы со стеком состоит в том, чтобы разрешать конфликты стека, которые выражаются в ситуациях нахождения правила как в множестве правил, сложенных на стек, так и в множестве правил, не сложенных на стек одной дуги.

5 Детали реализации

Генератор синтаксических анализаторов и интерпретатор генерируемых синтаксических анализаторов были реализованы как часть проекта YaccConstructor. Проект YaccConstructor занимается разработкой одноименного инструмента для конструирования синтаксических анализаторов и обработки грамматик. Проект является площадкой для исследований и разработок в области генераторов парсеров, компиляторов компиляторов и прочего программного обеспечения, связанного с грамматиками формальных языков. Инструмент проекта разрабатывается на языке программирования F#, поэтому это также стал и язык реализации алгоритмического решения данной работы, кроме того, это целевой язык разработанного генератора синтаксических анализаторов.

В инструменте YaccConstructor в 2012 году был реализован алгоритм RNGLR [1]. Реализованное решение данной работы является модификацией RNGLR в YaccConstructor. Модификации подверглись построение LR-анализаторов в генераторе парсеров и работа со стеком в интерпретаторе создаваемых парсеров.

6 Апробация

Разработанные программы были протестированы на ряде простых грамматик. Тесты показали, что программы работают корректно.

Сравнение генераторов синтаксических анализаторов имеет два аспекта:

- сравнение генерации парсеров по фиксированным грамматикам, параметром сравнения является время генерации
- сравнение работы генерируемых парсеров, параметром сравнения является время анализа фиксированных строк исходной грамматики.

Для сравнения показателей работы был взято RNGLR решение из проекта YaccConstructor, далее обозначаемое RNGLR. Решение, описанное, в данное работе, будет обозначаться как RNGLR EBNF. Генератор парсеров RNGLR принимает на вход грамматики в EBNF форме, но трансформирует их перед процессом создания LR-автомата, устраняя расширяющие конструкции, такие как альтернативы и повторения.

Поскольку оба решения имеют в основе концепцию LR-анализатора и используют GSS в интерпретации парсеров, интерес для сравнения также представили:

- для сравнения генерации - количество состояний LR-анализаторов, генерируемых по фиксированных грамматикам;
- для сравнения работы генерируемых парсеров - количество вершин GSS, соответствующего полному анализу входных строк, количество переходов соответственно goto LR-таблицам и количество возвращений по дугам GSS в ходе сверток.

Дополнительные параметры сравнения частично характеризуют количество используемой оперативной памяти.

Для сравнения RNGLR и RNGLR EBNF тестировались на грамматике Calc, задающейся в нотации YARD:

```
binExpr<operand binOp>:
operand (binOp operand)*

[<Start>]
expr: binExpr<term termOp> {}
termOp: PLUS { (+) } | MINUS { (-) }
term: binExpr<factor factorOp> {}
factorOp: MULT { ( * ) } | DIV { (/) }
factor: binExpr<powExpr powOp> {}
powOp: POW { ( ** ) }
```

Это грамматика арифметических выражений, содержащая операции сложения, вычитания, умнажения, деления и возведения в степень. Как можно видеть, ее описание использует средства расширенной формы Бэкуса-Наура.

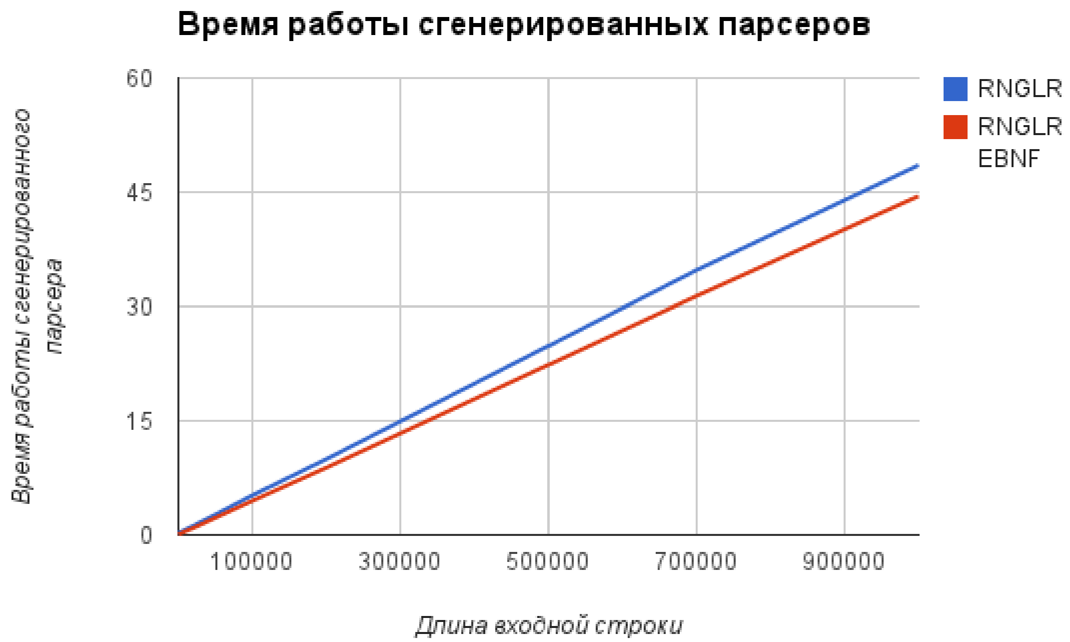
Ниже приведены показатели работы генераторов синтаксических анализаторов по входной грамматике Calc:

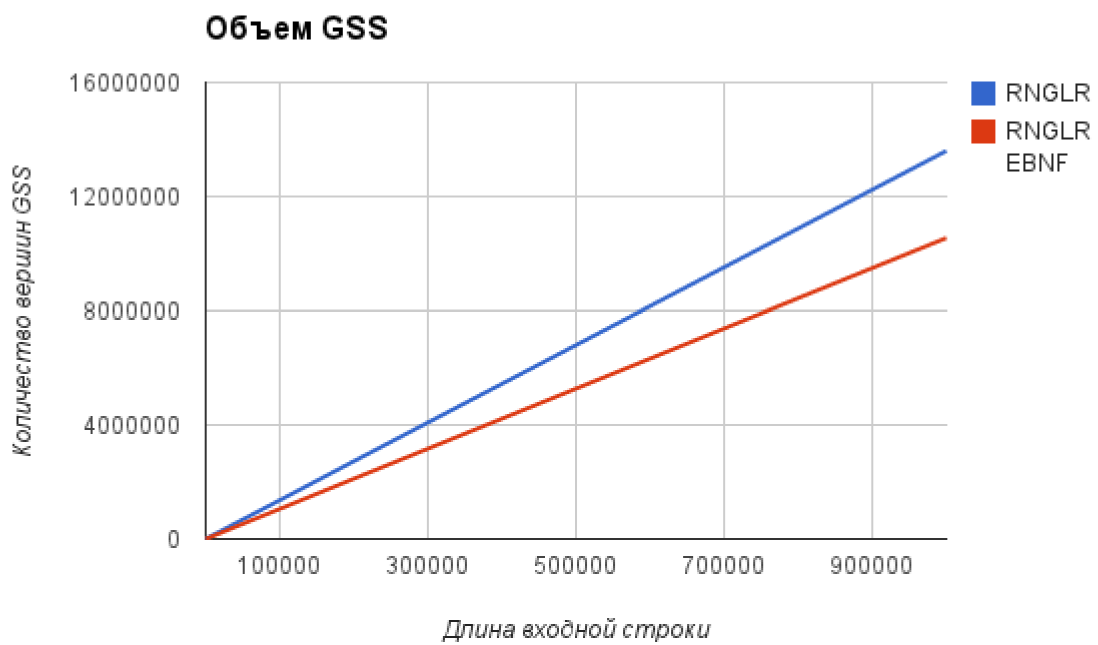
	RNGLR	RNGLR EBNF
Время генерации парсера, с.	0.76	0.70
Кол-во состояний LR-автомата	51	29

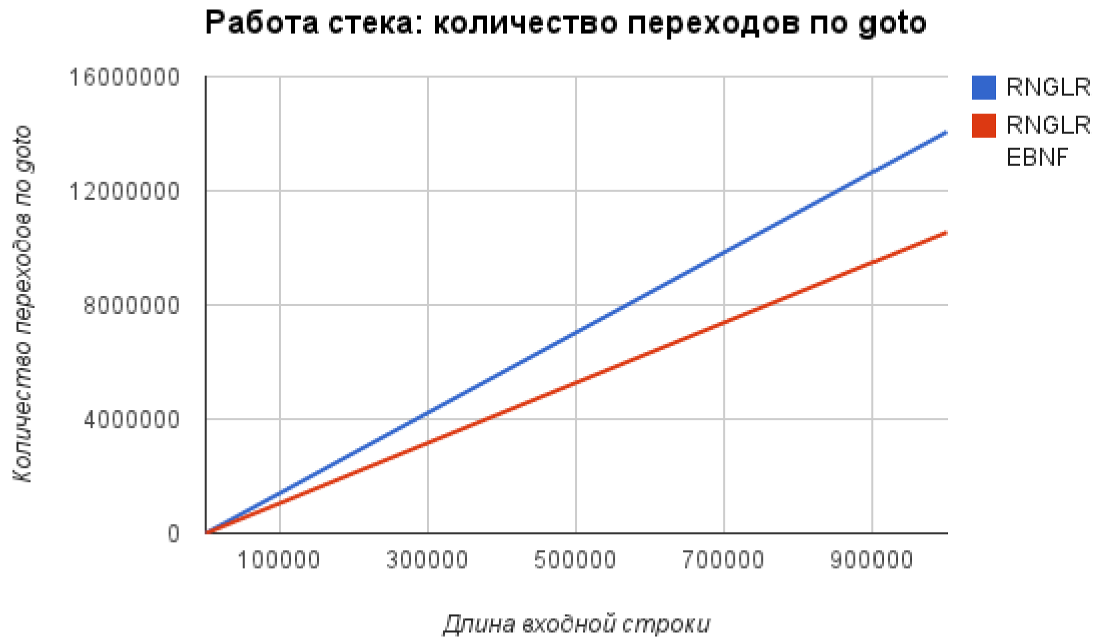
Таб. 1: Результаты тестирования генераторов синтаксических анализаторов на грамматике Calc.

Для тестирования порожденных парсеров было случайным образом сгенерировано множество примеров выражений приведенной грамматики, содержащих до 1000000 чисел. Ниже приведены графики результатов

тестирования:







Результаты тестирований и сравнений показывают, что разработанное решение может показывать лучшую производительность, чем существующие решения, которые не сохраняют EBNF.

В то же время, анализаторы, генерируемые RNGLR EBNF, дают вывод (дерево синтаксического разбора) ипользуя термины исходных грамматик. Эти деревья имеют меньшую высоту и меньшее количество узлов, нежели решения, которые трансформируют исходные грамматики, увеличивая множества нетерминалов и правил.

Ниже в качестве примера приведены деревья анализа, созданные RNGLR и RNGLR EBNF, в качестве исходной грамматики взята грамматика Г1, описанная выше, входная строка - *A A A B*.

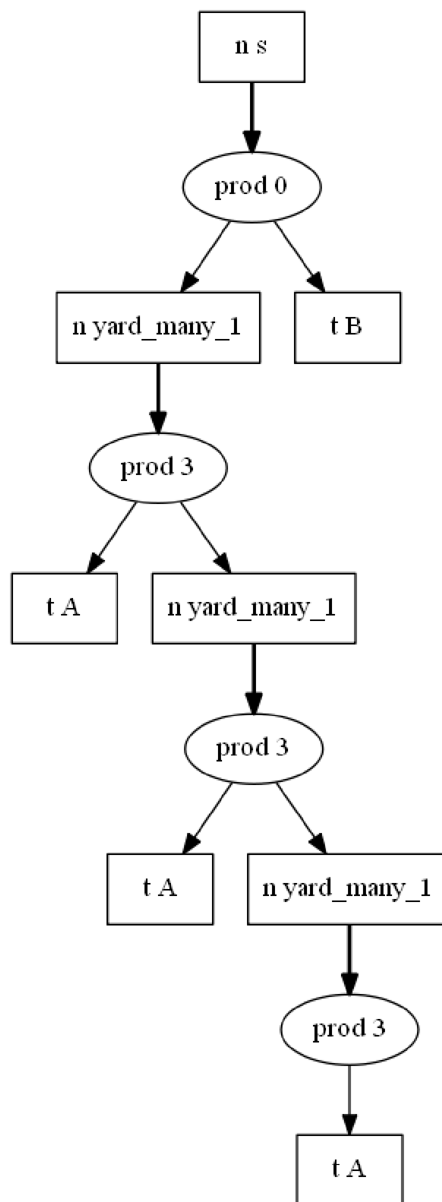


Рис. 1: Пример дерева вывода синтаксического анализатора, порожденного решением RNGLR

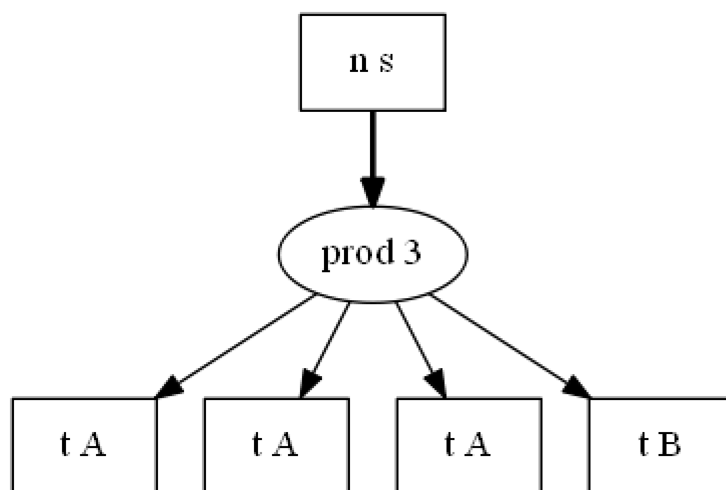


Рис. 2: Пример дерева вывода синтаксического анализатора, порожденного решением RNGLR

На рис.1 видно, как была изменена исходная грамматика - добавился сгенерированный нетерминал yard_many_1, чтобы удалить конструкцию повторения.

Следует заметить, что входная строка, взятая для примера, приводит LR-анализатор, создаваемый по исходной грамматике согласно алгоритму, описанному в статье [12], в состояние с конфликтом стека, однако RNGLR EBNF разрешает эту проблему благодаря GLR алгоритму.

7 Заключение

В ходе выполнения данной работы были получены перечисленные ниже результаты:

- Изучена предметная область - алгоритмы и решения в области синтаксического анализа.
- Разработано алгоритмическое решение, связанное с поставленной целью данной работы: за основу был взят алгоритм RNGLR, были проведены модификации его работы со стеком; алгоритм позволяет разрешать конфликты стека; реализация алгоритмического решения была произведена в рамках проекта YaccConstructor.
- Созданы по разработанному решению программа-генератор синтаксических анализаторов с сохранением EBNF и программу-интерпретатор порождаемых синтаксических анализаторов.
- Проведена апробация; тестирование разработанных программ и сравнение замеров тестирования с показателями работы аналогичных решений, подразумевающих трансформации исходной формы входной грамматики, показало, что сохранение EBNF может увеличить производительность синтаксического анализа.

Исходный код разработанных программ выложен на сайте YaccConsctructor - https://code.google.com/p/recursive-ascent/wiki/YaccConstructor. Работа велась под учетной записью alefirov93aa.

Список литературы

[1] *Авдюхин Д.А.* Создание генератора GLR трансляторов для .NET. Курсовая работа. СПБГУ. 2012.

[2] Автоматизированный реинжиниринг программ / Под ред. проф. А.Н. Терехова и А.А. Терехова. - СПб.: Издательство С.-Петербургского университета, 2000. 332 с.

[3] *Ахо А., Сети Р., Ульман Дж.* Компиляторы: принципы, технологии, инструменты. М. : Издательский дом "Вильямс" 2008. 281 с.

[4] *Григорьев С.В.* Генератор синтаксических анализаторов для неоднозначных контекстно-свободных грамматик. Дипломная работа. СПБГУ. 2010.

[5] *Мартыненко, Б.К.* Языки и трансляции. 2-е издание. Издательство Санкт-Петербургского Университета. 2012. 206 с.

[6] *Хопкрофт, Дж., Э., Мотвани, Р., Ульман, Дж., Д.* Введение в теорию автоматов, языков и вычислений, 2-е изд.. : Пер. с англ. — М. : Издательский дом "Вильямс". 2002. 120 с. 284 с.

[7] *Aho A.V., Johnson S.C.* LR Parsing // ACM Computing Surveys. Vol.6. Issue 2. ACM. 1974. P 99-124.

[8] *Brüggemann-Klein, A., Wood, D*. On predictive parsing and extended context-free grammars // CIAA'02 Proceedings of the 7th international conference on Implementation and application of automata. Springer. 2003. P 239-247.

[9] *Hemerik, K.* Towards a Taxonomy for ECFG and RRPG Parsing // LNCS. Vol. 5457. Springer. 2009. P. 410-421.

[10] ISO/IEC 14977 : 1996(E), "Information technology — Syntactic metalanguage — Extended BNF"

[11] *Morimoto, S.I., Sassa, M*. Yet another generation of LALR parsers for regular right part grammars // Acta Informatica. Vol. 37. Issue 9. 2001. P. 671–697.

[12] *Purdom, P.W.Jr., Brown C. A.*. Parsing extended *LR(k)* grammars // Acta Informatica. Vol. 15. Issue 2. Springer. 1981. P. 115-127.

[13] *Sassa, M., Nakata, I.*. A Simple Realization of LR-Parsers for Regular Right-Part Grammars // Information Processing Letters. Vol. 24. Issue 2. Elsevier North-Holland. 1987. P. 113 - 120.

[14] *Scott, E., Johnstone, A*. Right Nulled GLR Parsers. Royal Holloway, University of London. 2006.

[15] *Shin, H.-Ch., Choe, Kw.-M.*. An Improved LALR(k) parser generation for regular right part grammars // Information Processing Letters. Vol. 47. Issue 3. Elsevier North-Holland. 1993. P. 123-129.

[16] *Tomita, M*. Efficient Parsing for Natural Language. Kluwer Academic, Boston. 1986.

[17] *Wilhelm, R., Maurer, D* . Compiler design. Addison-Wesley. 1995.

[18] Сайт разработчиков ANTLR: http://www.antlr.org/

[19] Сайт разработчиков Bison: http://www.gnu.org/software/bison/

[20] Сайт разработчиков Elkhound: http://scottmcpeak.com/elkhound/

[21] Сайт разработчиков Jison: http://zaach.github.io/jison/

[22] Страница в сети интернет, посвященная Ocamlyacc: http://caml.inria.fr/pub/docs/manual-ocaml-400/manual026.html

[23] Официальная страница SGLR в сети интернет: http://www.meta-environment.org/

[24] Yacc - генератор синтаксических анализаторов в Unix-системах. Сайт разработчиков Yacc: http://dinosaur.compilertools.net/yacc/

[25] YaccConstructor - это инструмент, предназначенный для конструирования синтаксических анализаторов и обработки грамматик. Более подробную информацию можно найти на сайте проекта: https://code.google.com/p/recursive-ascent/wiki/YaccConstructor

[26] Страница YARD на сайте YaccConstructor https://code.google.com/p/recursive-ascent/wiki/YARD_frontend