ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ИМЕНИ Н. Э. БАУМАНА

(МГТУ ИМ. Н. Э. БАУМАНА)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»*

*КАФЕДРА «ТЕОРЕТИЧЕСКАЯ ИНФОРМАТИКА*

*И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ»*

ОТЧЕТ ПО ПРЕДДИПЛОМНОЙ ПРАКТИКЕ

***«ГЕНЕРАЦИЯ ТЕСТОВ ДЛЯ СИНТАКСИЧЕСКОГО АНАЛИЗАТОРА МЕТОДАМИ СУПЕРКОМПИЛЯЦИИ»***

Студент группы ИУ9-72 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ **Головань С. М.**

Руководитель дипломного проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ **Коновалов А. В.**

Москва, 2018

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc517291348)

[1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 4](#_Toc517291349)

[1.1. СИНТАКСИЧЕСКИЙ АНАЛИЗ 4](#_Toc517291350)

[1.1.1. Постановка задачи синтаксического анализа 4](#_Toc517291351)

[1.1.2. LL(1)-грамматики 5](#_Toc517291352)

[1.1.3. Предсказывающий анализ 6](#_Toc517291353)

[1.1.4. Восстановление при ошибках 8](#_Toc517291354)

[1.1.5. Граф состояний анализатора 9](#_Toc517291355)

[1.2. СУПЕРКОМПИЛЯЦИЯ 11](#_Toc517291356)

[1.2.1. Основные понятия 11](#_Toc517291357)

[1.2.2. Граф конфигураций 11](#_Toc517291358)

[1.2.3. Свертка дерева конфигураций 12](#_Toc517291359)

[1.2.4. Отношение Турчина 13](#_Toc517291360)

[1.3. ПОСТАНОВКА ЗАДАЧИ ТЕСТИРОВАНИЯ 15](#_Toc517291361)

[2. РАЗРАБОТКА 16](#_Toc517291362)

[2.1. ПАРСЕР ГРАММАТИКИ 16](#_Toc517291363)

[2.2. ПОСТРОИТЕЛЬ ГРАФА КОНФИГУРАЦИЙ 16](#_Toc517291364)

[3. ТЕСТИРОВАНИЕ 19](#_Toc517291365)

[ЗАКЛЮЧЕНИЕ 21](#_Toc517291366)

[СПИСОК ЛИТЕРАТУРЫ 22](#_Toc517291367)

# ВВЕДЕНИЕ

С развитием языков высокого уровня значение компиляторов и интерпретаторов приобрело промышленные масштабы. При разработке ПО программист рассчитывает, что программа, написанная на исходном языке A будет корректно переведена на язык B.

Именно поэтому тестирование компиляторов занимает важное место при их разработке. Ожидается, что в процессе компиляции логика программы не будет искажена, а компилятор при любых допустимых входных данных -возможно, ошибочных с точки зрения грамматики входного языка - либо успешно переведет программу на язык B, либо выдаст сообщение об ошибке.

В данной работе рассматриваются методы генерации тестов для синтаксического анализатора, являющегося неотъемлемой частью любого компилятора. Рассматриваемые методы основаны на идее суперкомпиляции, предложенной советским ученым В. Ф. Турчиным в 1970-х годах.

В качестве входного языка часто рассматриваются -грамматики, преимуществом которых является возможность построения таблицы предсказывающего анализа для выполнения разбора входной цепочки детерминированным распознавателем – автоматом с магазинной памятью.

Целью данной работы является практическое изучение и применение методов суперкомпиляции для построения набора тестовых цепочек парсера -грамматик.

# ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## СИНТАКСИЧЕСКИЙ АНАЛИЗ

### Постановка задачи синтаксического анализа

**Синтаксический анализ** – фаза компиляции, группирующая лексемы, порождаемые на фазе лексического анализа, в синтаксические структуры для некоторой заданной порождающей грамматики.

**Контекстно свободная (КС) грамматика** – кортеж , где:

– множество нетерминальных символов

– множество терминальных символов

– набор правил вывода: , *где* ,

– стартовое правило вывода (аксиома)

Для такой грамматики всегда можно построить **дерево вывода –** упорядоченное дерево, каждая вершина которого помечена символом из множества .

Корнем дерева является аксиома грамматики, внутренние вершины – нетерминалы, листья – терминалы, либо . Построение дерева производится согласно правилам вывода грамматики.

Опишем теперь основную задачу синтаксического анализа. Пусть дана КС-грамматика а также – входная цепочка. Тогда **основной задачей синтаксического анализа** является определение принадлежности входной цепочки заданной грамматике и, в случае положительного ответа, формирование набора данных, по которому возможно построение дерева вывода.

Часто КС-грамматики задаются в расширенной форме Бэкуса-Наура (РБНФ), т.е. имеют вид дерево, задаваемое следующей грамматикой: , где

альтернатива, вхождение 0 и более раз, вхождение 1 и более раз, вхождение 0 или 1 раз.

Здесь и далее рассматриваются только контекстно-свободные грамматики в форме РБНФ как наиболее часто используемые и легко формализуемые.

### LL(1)-грамматики

КС-грамматики представляют собой обширный класс языков. Их наиболее распространенным подклассом являются грамматики вида , для которых можно построить синтаксический анализатор, работающий за линейное время. Для определения раскрытия нетерминального правила такому анализатору требуется информация о следующих входных символах.

Наибольший интерес здесь представляет класс -грамматик, для распознавания которых достаточно иметь информацию только о текущем входном символе. Такие грамматики обладают рядом преимуществ, которые будут рассмотрены в данной работе. Но прежде опишем указанный класс более формально.

Введем вспомогательные множества и , определяемые для любой КС-грамматики :

Таким образом, множество есть множество терминалов, с которых начинаются цепочки, выводимые из . В свою очередь – множество терминалов, следующих за выводом цепочки из . Указанное множество может также содержать терминальный символ $ - концевой маркер, если может быть самым правым символом некоторой сентенциальной формы.

Как упоминалось выше, грамматика имеет вид , если для раскрытия любого из её правил достаточно знать только текущий символ входного потока. В терминах множеств и данное определение можно переформулировать следующим образом:

КС-грамматика является , если для любого правила вывода

выполняется:

2. *Если то*

Как правило, большинство грамматик языков программирования задаются в общем виде, то есть содержат левую рекурсию и неоднозначности.

Поскольку наличие левой рекурсии и неоднозначностей выбора альтернативы, очевидно, не позволяют исходной грамматике быть , предварительно применяются соответствующие процедуры, устраняющие эти недостатки (устранение левой рекурсии, левая факторизация).

Также, грамматика КС-грамматика может быть преобразована к эквивалентной, в которой множество правил вывода не содержит переходов по , а также цепных правил и недостижимых символов.

Таким образом, исходная грамматика обычно предварительно обрабатывается с целью оптимизации разбора и возможности применения к ним общеизвестных подходов построения синтаксических анализаторов.

### Предсказывающий анализ

Как уже было упомянуто выше, -грамматики обладают важным свойством: они могут быть распознаны предсказывающим анализатором – автоматом с магазинной памятью, работа которого основана на предварительном построении таблицы предсказывающего разбора.

**Детерминированный автомат с магазинной памятью** – это набор

, где

– входной алфавит,

– стековый алфавит,

– множество состояний автомата,

– множество заключительных состояний автомата,

– начальное состояние,

– маркер дна стека (обычно обозначается как «$»),

– функция переходов.

В контексте синтаксического анализа описанной ранее -грамматики , входной алфавит совпадает со множеством терминальных символов, дополненных маркером дна стека, . Cтековый алфавит – , а функция переходов задается таблицей предсказывающего анализатора, , где – индикатор ошибки, – концевой маркер.

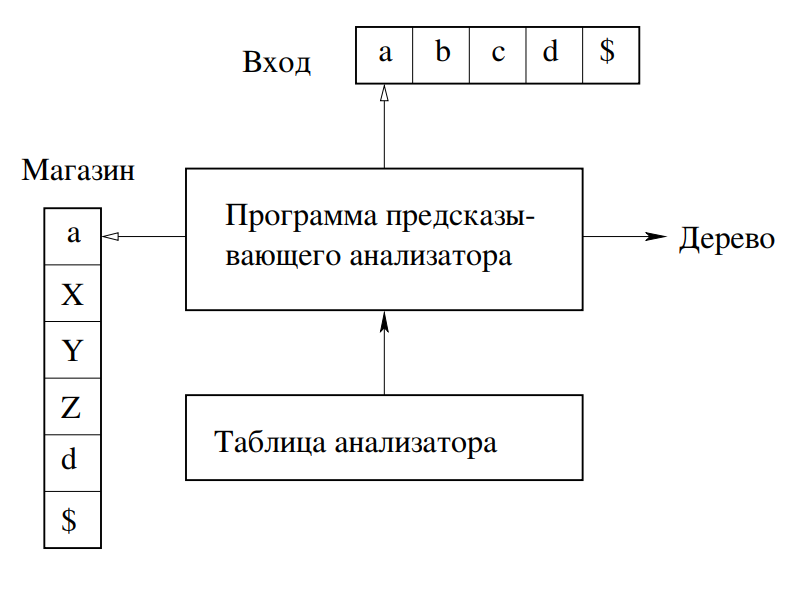


Рисунок . Схема работы анализатора

Рассмотрим процесс построения таблицы предсказывающего анализатора. Такая таблица однозначно определяет раскрытие нетерминального правила по текущему символу во входном потоке. В ячейках таблицы располагаются либо правые части правил грамматики, либо индикатор ошибки.

Далее рассматривается детерминированный случай, где в ячейке таблицы может располагаться единственное возможное раскрытие правила по текущему входному символу.

Пусть в начале . Затем, для всех правил перехода грамматики выполняем следующее:

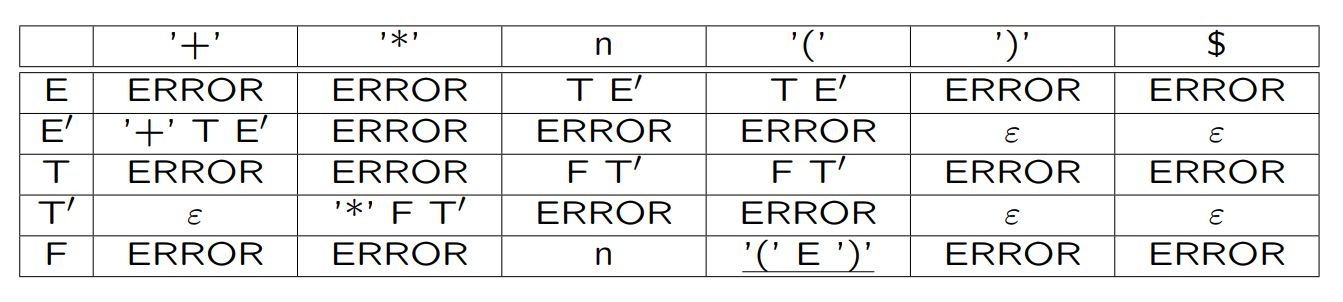


Рисунок . Таблица предсказывающего анализатора грамматики арифметических выражений

При попытке перейти по ячейке с ошибочным правилом анализатор переходит в режим восстановления, который будет рассмотрен далее.

### Восстановление при ошибках

Очевидно, что на вход анализатору не всегда будут подаваться корректные цепочки. В процессе работы предсказывающего анализатора может возникнуть ситуация, когда терминал на верхушке стека не соответствует входному символу или, когда на вершине стека находится нетерминал, а ячейка в таблице, соответствующая данному нетерминалу и текущему входному символу, содержит признак ошибки.

Рассмотрим восстановление при ошибках «в режиме паники». Он основан на пропуске символов входного потока до тех пор, пока не будет обнаружен токен из синхронизирующего множества. Такие множества должны выбираться так, чтобы анализатор мог быстро восстанавливаться после часто встречающихся на практике ошибок.

В качестве синхронизирующих множеств могут быть использованы множества нетерминалов грамматики. Также, если нетерминал может порождать пустую строку, то по умолчанию может быть использована пустая продукция. Если терминал на вершине стека не может быть сопоставлен со входным символом, то терминал снимается со стека и синтаксический анализ продолжается. Тогда, синхронизирующее множество состоит из всех остальных токенов.

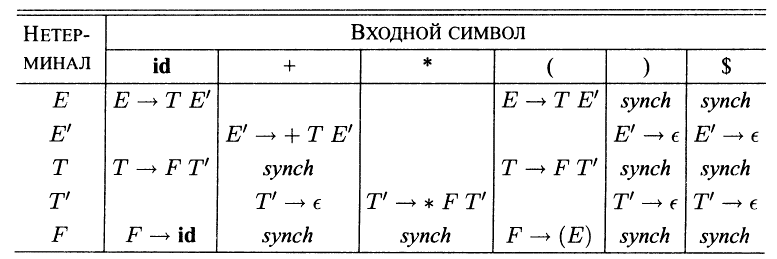


Рисунок . Таблица предсказывающего разбора, дополненная правилами восстановления

На Рисунке 3 пустыми ячейками обозначены ошибочные переходы. Значение в ячейке «synch» означает, что в данной ячейке содержится правило восстановления, заданное согласно синхронизирующему множеству для данного нетерминала.

Забегая вперед, следует сказать, что в дальнейшем потребуется полностью заполнить таблицу предсказывающего анализатора, даже для тех терминалов, которые не входят в синхронизирующее множество. При этом необходимо отличать ячейки, в которых находятся правила восстановления от обычных ячеек таблицы. Описание механизмов заполнения таблицы правилами восстановления описывается в разделе «Реализация».

### Граф состояний анализатора

Поскольку предсказывающий анализатор есть автомат с магазинной памятью, для него можно построить граф процессов (возможно бесконечный). В этом графе вершины являются состояниями магазина, а ребра могут быть двух типов – помеченные символом входной цепочки, если при переходе был потреблен символ, иначе – транзитные, не содержащие атрибутов.

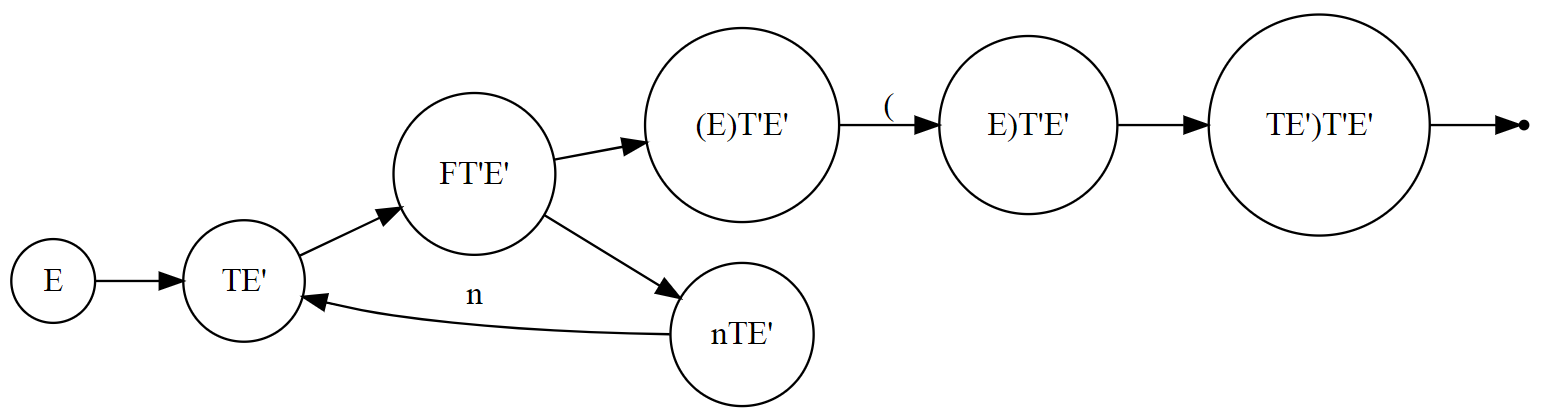


Рисунок . Граф состояний грамматики арифметических выражений

Выполняя удаление транзитных узлов в графе, а также выделяя циклы к эквивалентным вершинам, можно получить **граф состояний** анализатора.

Такой граф также может быть бесконечным, однако отсутствие транзитных переходов и отсутствие различных вершин с одинаковым состояние магазина существенно упростит процесс свертки.

Если дополнить таблицу предсказывающего анализа правилами восстановления, то из каждой вершины графа состояний будут исходить ребра для каждого терминального символа.

Как станет ясно дальше, именно возможность построения и свертки бесконечного графа, построенного по схожим правилам, а также использование идей суперкомпиляции, позволяют взглянуть на процесс генерации тестовых цепочек для синтаксического анализатора под другим углом.

## СУПЕРКОМПИЛЯЦИЯ

### Основные понятия

**Суперкомпиляция** (supercompilation) – процесс моделирования выполнения некоторой программы в общем виде с целью её преобразования или анализа. Под словосочетанием «в общем виде» подразумевается построение семантического дерева выполнения исходной программы, учитывающего все возможные входные данные – **дерева конфигураций**.

В общем случае дерево является бесконечным и предпринимается попытка свернуть данное его в конечный **граф конфигураций**. По полученному графу возможно построить «**остаточную**» **программу**, которая в некотором смысле будет являться оптимальной для заданных ограничений входных данных.

Говоря о суперкомпиляции -грамматик с целью порождения тестов для синтаксического анализатора, построение остаточной программы не требуется. Основной задачей является построение и свертка бесконечного дерева в конечный граф. Выполняя проход по графу определенным образом и выписывая атрибуты ребер – терминальные символы, можно построить необходимые наборы тестов. Процесс построения графа конфигураций для входной грамматики описан в разделе «Реализация».

### Граф конфигураций

Как было сказано ранее, дерево конфигураций задается при выполнении некоторой программы «в общем виде». Поскольку в общем случае программа может содержать циклы и/или рекурсию, дерево является бесконечным.

Имеется возможность построить данное дерево частично, остановив построение поддерева при некоторой заданной глубине, однако такое дерево не представляет практической ценности.

**Граф конфигураций** – это семантическое дерево программы, над которым произведены следующие преобразования:

1. Удаление транзитных переходов – удаление вершин дерева, являющихся промежуточными, не зависящими от конфигурационных переменных.
2. Выделение эквивалентных вершин – процесс поиска вершины с эквивалентным состоянием, и, в случае нахождения таковой, добавление ребра к найденной конфигурации вместо создания новой.
3. Свертка – процесс выделения отношений между вершинами дерева с целью превращения исходного бесконечного дерева в конечный граф.

Указанные здесь преобразования гарантируют конечность полученного графа, а значит его возможно обойти с целью анализа или преобразования исходной программы.

### Свертка дерева конфигураций

Рассмотрим методы свертки бесконечного дерева конфигураций. Свертка естественным образом разделяется на два инструмента: **вложение** и **обобщение**. Оба эти инструмента выявляют «сходство» некоторой конфигурации с конфигурациями-предками, разделяя её на независимые составные части.

При свертке дерева конфигураций факт наличия вложения (или обобщения) задается, так называемой, **-вершиной** – фиктивной вершиной, добавляемой в граф конфигураций. Такая вершина имеет только двух потомков: нижнего и верхнего, развитие которых определяется согласно наличию вложения либо обобщения. Таким образом, с помощью -вершин обеспечивается непосредственная свертка графа.

Вложение представляет собой выделение уже вычисленной ранее части конфигурации, с образованием цикла к найденной подконфигурации и дальнейшим разбором остаточной части конфигурации.

Говоря более формально, если имеется исходная конфигурация , а среди её предков существует конфигурация , то исходная конфигурация замещается -вершиной, из которой верхняя дуга исходит к конфигурации , а нижняя к новой конфигурации , развивающейся независимо. Независимость в данном случае означает, что при последующий поиск вложения или обобщения конфигураций из данного подграфа будет производиться только в его пределах. При наличии вложения, говорят, что конфигурации находятся в отношении Хигмана (вкладываются по Хигману).

Далее будут описаны теоретические основы, на которые опирается инструмент обобщения.

### Отношение Турчина

При решении задачи поиска семантических циклов в программе В. Ф. Турчин рассматривал преобразование стека функций, описываемое **префиксной грамматикой**, в которой также описывается изменение состояний автомата с магазинной памятью.

Тройка , где – алфавит, – начальное слово (возможно пустое), – конечный набор правил переписывания, в котором правила вывода применимы только к словам вида , где – суффикс, – префикс, называется **префиксной грамматикой**.

Обобщение сводится к поиску семантических циклов в программе с целью исключить ситуации, когда вершина стека проходит один и тот же путь развития неограниченное количество раз, в конце пути повторяя сама себя.

Схематично можно изобразить такую ситуацию следующим образом:

Или, говоря в контексте состояний анализатора:

*.*

Здесь и – непустые подмножества в стеке.

Рассматривая цепочки, порожденные префиксной грамматикой, проведем **временное индексирование** – припишем ко всем буквам слов их временный индекс. То есть просматривая цепочку и выписывая посимвольно каждое слово справа налево, пометим каждый символ тем моментом времени, в который он появился в цепочке.

Пусть – слова, буквы в которых размечены временными индексами. Можно определить отношение эквивалентности слов с точностью до временных индексов: .

Определим теперь **отношение Турчина** :

Пары , находящиеся в отношении Турчина называются **турчинскими**.

Конструкция обобщения неотделима от определения отношения Турчина. Канонически оно рассматривается для префиксных грамматик с учетом временных индексов. Здесь и далее будет рассмотрена «безвременная» версия данного отношения. Такое возможно в силу того, что «безвременное» отношение Турчина включает в себя каноническое в качестве подмножества, а в силу того, что турчинские пары встречаются на каждом пути, порождаемом грамматикой, на этом пути встретятся пары, находящиеся в каноническом отношении Турчина.

## ПОСТАНОВКА ЗАДАЧИ ТЕСТИРОВАНИЯ

Следует обратить внимание на сам процесс тестирования грамматики. Как было упомянуто ранее, оно осуществляется путем подачи некоторого набора входных цепочек и исследование работы на них синтаксического анализатора.

Ожидается, что такой набор тестов будет минимальным, то есть будет покрывать всю возможную логику работы парсера. При этом тестовые наборы стоит разделять на **позитивные** и **негативные.**

Позитивными являются тесты, проверяющие работу анализатора, подавая на вход цепочки, являющиеся предложениями языка. В свою очередь, негативными являются тесты, содержащие в себе синтаксическую ошибку.

Ожидается, что набор позитивных тестов будет покрывать все корректные состояния анализатора. Набор негативных тестов составляют такие тесты, которые выявляют единственную синтаксическую ошибку, если это возможно.

Важным понятием здесь является **критерий полноты тестирования** – критерий, по которому проводится классификация всех возможных вариантов выполнения проверки правильности выполнения программы.

Говоря о тестировании -грамматики, можно естественно сформулировать такой критерий в первом приближении:

*Тестирование синтаксического анализатора является полным тогда и только тогда, когда в процессе вывода слов из тестового набора окажутся посещены все ячейки таблицы предсказывающего разбора.*

Однако, как будет установлено в разделе «Реализация», рассмотренные ранее методы восстановления при ошибках позволят построить критерий полноты тестирования в терминах графа конфигураций.

Таким образом, задача тестирования состоит в построении минимального набора тестов, удовлетворяющего критерию полноты тестирования.

# РАЗРАБОТКА

## ПАРСЕР ГРАММАТИКИ

Основной задачей данной работы является построение графа конфигураций для входной грамматики и генерация на его основе набора тестовых цепочек. Для этого необходимо иметь инструмент, получающий на вход описание произвольной -грамматики, упрощающий её в процессе разбора и строящий для неё таблицу предсказывающего разбора, на основе которой и будет построен граф.

В качестве такого инструмента используется приложение из курсового проекта студента кафедры ИУ9 Михаила Макарова, в которой реализован front-end, позволяющий получить упрощенное представление грамматики в нужном виде.

Поскольку суть той работы заключалась в построении тестов грамматики на основе вычисление кратчайших цепочек, выводимых из нетерминалов, исходная программа была модернизирована так, что она возвращает представление грамматики в некотором формате, принимаемом построителем графа конфигураций.

## ПОСТРОИТЕЛЬ ГРАФА КОНФИГУРАЦИЙ

Одной из основных задач работы является построение модели выполнения разбора цепочки грамматики «в общем виде», то есть построение графа конфигураций.

Как было описано в разделах «Синтаксический анализ» и «Суперкомпиляция», построение такого графа производится с использованием таблицы предсказывающего анализа для данной грамматики. Дополнив таблицы правилами восстановления, можно получить граф, содержащий ошибочные переходы, по которым в дальнейшем возможно также построение негативных тестов.

В связи с этим, введем следующее правило для ошибочных ячеек. Пусть имеется ошибочная ячейка . Тогда, если то добавляем в ячейку правило , иначе добавляем правило . Другими словами, если символ на вершине стека принадлежит синхронизирующему множеству, то нетерминал снимается со стека в попытке продолжить анализ, иначе токен из входного потока пропускается.

Также, потребуется модификация исходной -грамматики, заключающаяся в добавлении фиктивных нетерминалов , отвечающих каждому терминалу :

Также введем фиктивный нетерминал для маркера конца ввода. Определяя такие нетерминалы следует также заменить исходные терминалы в грамматике на фиктивные правила.

Смысл данного приема заключается в необходимости построения полного набора ошибочных ребер в графе конфигураций, речь о котором пойдет в следующих пунктах. Для этого удобно иметь маркеры ошибки в таблице перехода и при этом продолжать разбор в случае попадания в ошибочную транзитную вершину.

Заметим, что введение таких нетерминалов сохраняет -вид грамматики. Данное утверждение основывается на процессе построения множеств и для обычных нетерминалов: при замене терминалов на фиктивные нетерминалы, данные множества не изменяются, а при их конструировании лишь добавляется еще один транзитный шаг (аналогично и для ).

Таким образом все ячейки таблицы оказываются полностью заполнены и при построении графа конфигураций из каждой конфигурационной вершины будут исходить ребра по всем возможным терминальным символам.

Далее приведен алгоритм построения графа конфигураций. В начале построим корневую конфигурацию, соответствующую начальному состоянию магазина анализатора. Затем, перебирая каждый терминал, исследуем развитие стека при переходе по соответствующему терминалу входному символу. Если в процессе перехода была посещена ошибочная ячейка таблицы, то фиксируем данный факт. Если в процессе изменения стека не был потреблен терминальный символ, значит, перехода по данному терминалу нет, и он отбрасывается. Иначе выполняется попытка свернуть граф, выполняя поиск конфигурации с эквивалентным состоянием, либо выделяя отношения вложения или обобщения, описанные ранее. Стоит заметить, что некоторые конфигурации будут являться «финальными». Такими являются конфигурации, в которых стек может быть раскрыт в пустоту, либо является пустым. Финальные конфигурации представляют собой заключительные состояния, анализируя которые возможно выполнить обход полученного графа с выделением тестового набора цепочек.

В данный момент ведется разработка алгоритма обхода получившегося графа, с учетом семантики -вершин и финальных вершин, поскольку наличие таких вершин требует определенной разметки посещенных вершин, наличие стека для обработки перемещения по -вершинам и критерия окончания обхода.

# ТЕСТИРОВАНИЕ

В процессе выполнения работы было реализовано приложение для построения графа конфигураций входной -грамматики, а также его представление в графическом виде.

На вход программе подавались различные -грамматики.

non-terminal E, E1, F;

terminal '+', 'n';

E ::= F E1;

E1 ::= '+' F E1 | eps;

F ::= 'n';

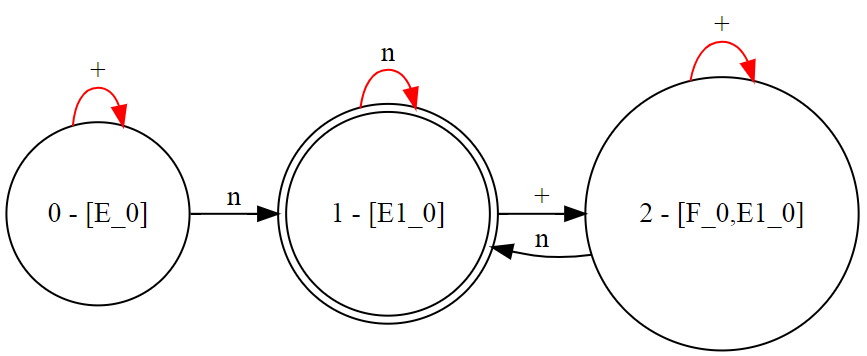


Рисунок . Граф конфигураций для грамматики сложений

non-terminal E, E1, T, T1, F, n;  
terminal '+', '\*', '(', ')', a, b, c;  
  
E ::= T E1;  
E1 ::= '+' T E1 | eps;  
T ::= F T1;  
T1 ::= '\*' F T1 | eps;  
F ::= n | '(' E ')';  
n ::= (a|b|c) n\*;

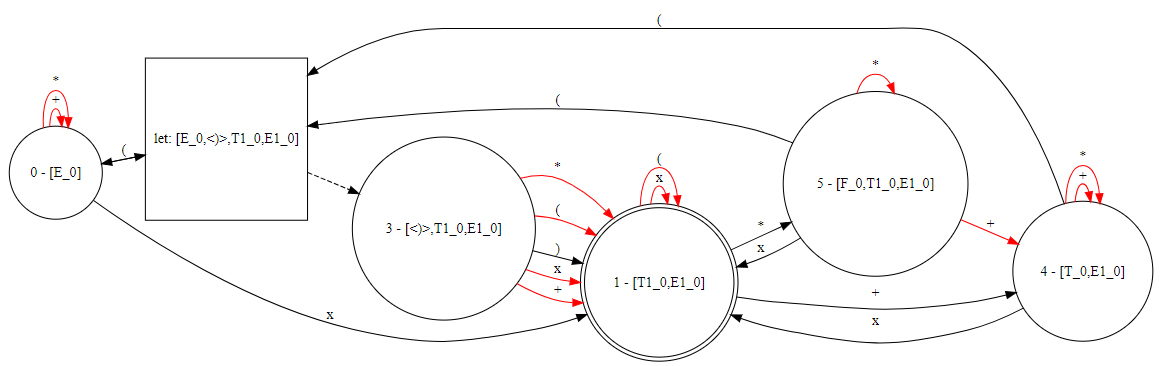


Рисунок . Граф конфигураций для грамматики арифметических выражений

Также тестирование проводилось на грамматике, описывающей json, арифметические операторы. Также был произведен тест с построением собственной грамматики парсера.

# ЗАКЛЮЧЕНИЕ

В ходе прохождения преддипломной практики были изучены основные методы суперкомпиляции, в частности – выполнена суперкомпиляция -грамматики, разработано приложение, позволяющее строить граф конфигураций -грамматики, заданной в РБНФ.

Поскольку основной целью работы является генерация положительных и отрицательных тестов для синтаксического анализатора на основе построенного графа конфигураций, итоговое приложение активно дорабатывается с целью удовлетворения критерию полноты тестирования. Также изучаются возможности модернизации методик обхода полученного в результате работы программы графа.

# СПИСОК ЛИТЕРАТУРЫ

1. Ахо, Ульман, Сети – Компиляторы. Принципы, технологии, инструментарий, 2010.
2. А.П. Немытых, А.Н. Непейвода – Cборник трудов по функциональному языку программирования Рефал 1, 2014.
3. И. Ключников – Суперкомпиляция: идеи и методы. Практика функционального программирования №7, 2011.
4. C. Романенко – Заметки о суперкомпиляции. Web-ресурс. <https://sergei-romanenko.github.io/scp-notes-ru/>.