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»*

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

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

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

Научный руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ **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 в процессе разработки и применения рядовыми пользователями. От каждого компилятора ожидается, что в процессе его работы 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 одной из наиболее значимых частей любого компилятора. Теоретическая часть, в основном, опирается на идею суперкомпиляции, предложенную советским ученым В. Ф. Турчиным еще в 70-х годах прошлого века, однако широкого распространения суперкомпиляция не получила по сей день, но интересна в рамках решения определенного ряда задач.

К сожалению, в рамках данной работы было бы невозможно привести универсальный пример, не опирающийся на определенную узкую область, поскольку процесс разработки теории и реализации адекватного программного решения, учитывающего все многообразие различных входных данных, потребовал бы несопоставимых временных затрат, даже в контексте КС-грамматик. В связи с этим, данная работа ограничится рассмотрением -грамматик в качестве входного языка. Преимуществом таких грамматик является возможность построения таблицы предсказывающего анализа для выполнения разбора входной цепочки детерминированным распознавателем – автоматом с магазинной памятью.

Таким образом, целью данной работы является практическое изучение и применение методов суперкомпиляции для построения конечного набора тестов – цепочек языка – определенного вида (позитивные и негативные), проверяющих работу синтаксического анализатора, распознающего -грамматики.

# 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.

Как было сказано ранее, в данной работе рассматривается класс -грамматик, поэтому необходимо сказать несколько слов о КС-грамматиках, подмножеством которых и являются .

**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 . Символ является специальным и обозначает пустое правило раскрытия нетерминала.

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. Зачатую сам факт выполнения построения такого дерева не имеет значения, однако важным результатом является построение некоторых структур данных, с помощью которых построение дерева вывода будет являятся возможным.

Часто -грамматики (как и все контекстно-свободные языки) 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.

Данная работа не является исключением, поэтому все указанные грамматики для удобства задаются в РБНФ.

### 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 :

Говоря простым языком, множество есть множество терминалов, с которых начинаются цепочки, выводимые из .

В свою очередь – множество терминалов, следующих за выводом цепочки из . Данное 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) [1].

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

Таким образом, исходная грамматика обычно предварительно обрабатывается с целью оптимизации разбора и возможности быть поданной на вход некоторому синтаксическому анализатору.

### Предсказывающий анализ

Как уже было упомянуто ранее, -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. То есть, в обычной ячейке такой таблицы располагается некоторый упорядоченный набор (возможно пустой) терминальных и нетерминальных символов, а в ошибочной – индикатор ошибки.

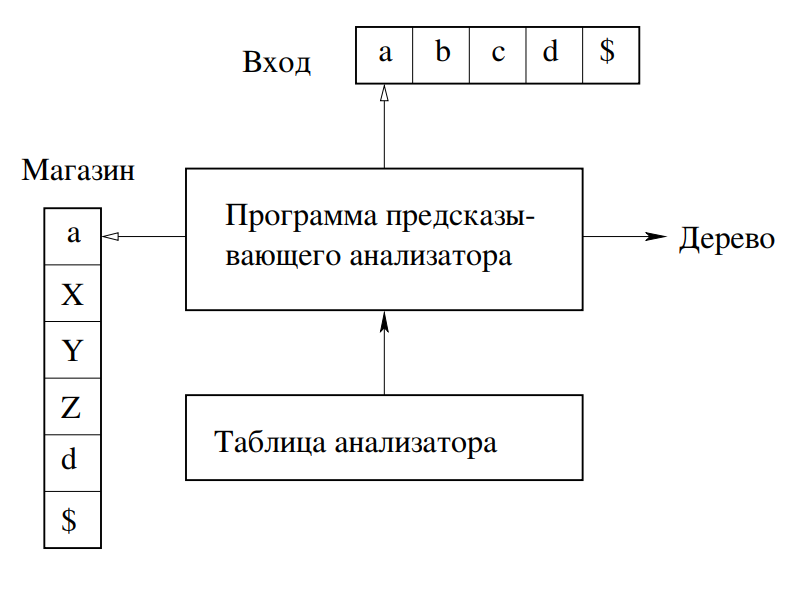


Рисунок 1. Структура предсказывающего анализатора (Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы: принципы, технологии и инструментарий).

Стоит отдельно рассмотреть процесс построения таблицы предсказывающего анализатора для -грамматики. Такая таблица однозначно определяет раскрытие нетерминального правила по текущему терминальному символу входной цепочки. В ячейках таблицы располагаются либо правые части правил грамматики, либо индикатор ошибки. В каждой ячейке такой таблицы может располагаться единственное возможное раскрытие правила по текущему входному символу или признак ошибки.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ) |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Рисунок 2. Таблица предсказывающего анализатора грамматики арифметических выражений

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

На Рисунке 2 можно увидеть построенную таблицу предсказывающего разбора для следующей грамматики, называемой грамматикой арифметических выражений:

;

;

;

Из такой грамматики, в частности, выводятся цепочки и для некоторого . При попытке перейти по ячейке с ошибочным правилом анализатор переходит в режим восстановления, который будет рассмотрен ниже.

### 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. В таком случае, интерпретатор может либо остановить разбор цепочки, либо войти в режим восстановления при ошибках, попытавшись заменить или пропустить некорректный терминал на входе, для того, чтобы иметь возможность выявить другие синтаксические 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 [1]. 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 2. Таблица предсказывающего разбора для грамматики арифметических выражений, дополненная обычными правилами восстановления

На Рисунке 3 значение в ячейке «synch» означает, что в ней 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 некоторый граф (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. Транзитные переходы из каждого состояния строятся согласно альтернативам для нетерминала на верхушке стека. Если на верхушке – терминал, то строится переход в следующее состояние по данному символу.

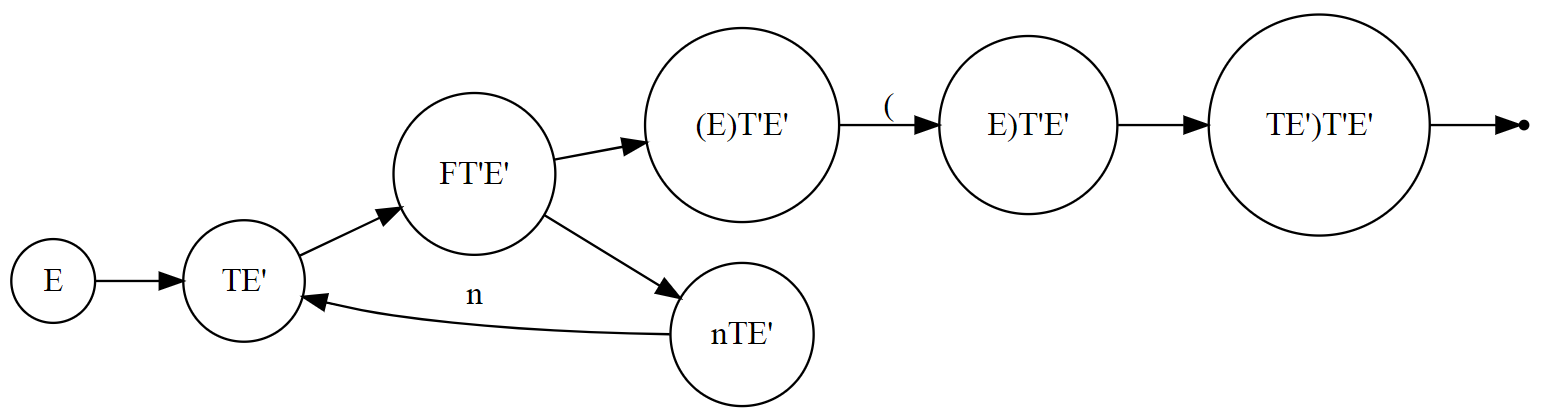


Рисунок 3. Граф переходов грамматики арифметических выражений

На Рисунке 4 продемонстрирован фрагмент бесконечного графа, который характеризует общий процесс разбора входных цепочек для входной грамматики арифметических выражений. Очевидно, что такой граф является бесконечным, поскольку для некоторых терминалах нетерминал на верхушке стека будет периодически повторяться, однако сам стек будет расти.

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, который будет рассмотрен далее.

Если дополнить таблицу предсказывающего анализа расширенными правилами восстановления, полностью заполняющими её в ошибочных ячейках, то из каждой вершины графа состояний будут исходить ребра для каждого терминального символа. Под расширенными правилами подразумевается использование обычных правил восстановления с учетом принадлежности текущего входного символа множеству нетерминала на верхушке стека.

В случае отрицательного ответа, для данного символа будет применено несколько другое правило. Формирование таблицы предсказывающего анализатора с учетом расширенных правил восстановления детально описано в разделе «Реализация».

Как станет ясно дальше, именно возможность построения бесконечного графа, построенного с использованием таблицы предсказывающего анализатора и расширенных правил восстановления, а также применение техник суперкомпиляции для сверки такого графа позволяют построить конечный набор тестов, полностью покрывающих всю логику работы синтаксического анализатора.

## 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**. Вершинами такого дерева являются «конфигурации» – обобщенные состояния программы, включающие переменные и вызовы функций.

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 при заданных ограничениях на входные данные.

Говоря о суперкомпиляции -грамматик с целью порождения тестов для синтаксического анализатора, процесс построения остаточной программы – оптимизированной грамматики входного языка – не является приоритетным. Одной из главных задач здесь является построение и свертка бесконечного дерева в конечный граф. Вершинами такого графа будут являться состояния магазина предсказывающего анализатора, а ребрами – переходы по некоторому терминалу в следующее состояние. Далее для выполнения свертки вершины будут разделены на **конфигурации** и **-узлы**, речь о которых пойдет в пункте «Свертка дерева конфигураций».

Выполняя проход по графу конфигураций определенным образом и выписывая атрибуты ребер – терминальные символы – можно построить необходимые наборы тестов. Процесс построения и обхода 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 может являться бесконечным.

**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, как станет ясно позже, гарантируют конечность полученного графа, а значит гарантируют возможность его полного обхода с целью построения тестовых наборов цепочек.

### 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.

Вершина графа конфигураций, в которой произошла свертка, обозначается, так называемой, **-вершиной**. Такая вершина имеет только двух потомков: нижнего и верхнего, развитие которых определяется согласно наличию вложения либо обобщения. Также, такая вершина подразумевает независимость её потомков, что означает окончание в ней поиска наличия вложения или обобщения для всех вершин её подграфа. Таким образом, с помощью -вершин обеспечивается непосредственная свертка дерева в граф.

Вложение представляет собой выделение уже вычисленной ранее части конфигурации, с образованием цикла к найденной подконфигурации (общего собственного префикса двух стеков) и дальнейшим разбором уже оставшейся части конфигурации.

Говоря более формально, если имеется конфигурация со стеком , а среди её предков существует конфигурация , то исходная конфигурация замещается -вершиной, из которой верхняя дуга исходит к конфигурации конфигурации , а нижняя – к новой конфигурации , развивающейся, как уже было сказано ранее, независимо.

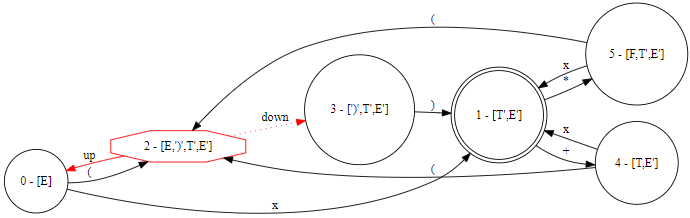


Рисунок 4. Граф конфигураций для грамматики арифметических выражений

На Рисунке 4 приведен граф конфигураций для грамматики арифметических выражений (для наглядности, здесь опущены ошибочные переходы). Красным цветом здесь обозначена -вершина, характеризующая наличие вложения, а также верхняя и нижняя дуги, одна из которых исходит в корневую конфигурацию. Однако не всегда возможно обойтись вложением для свертки дерева конфигураций. Как будет рассказано далее, в процессе построения могут возникать ситуации, когда стек начинает разрастаться, при том, что его префикс и суффикс (оба ненулевой длины) остаются неизменными. В связи с этим, далее будут описаны теоретические основы, на которые опирается инструмент обобщения, позволяющий свернуть граф в таких ситуациях.

### 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 временной индекс. Иными словами, 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, а также для обогащенных префиксных грамматик.

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. Однако отсутствие учета временных индексов в общем случае для некоторых грамматик может привести к слишком ранней свертке:

Для такой грамматики, при развертке нетерминала возникнут стеки и , которые будут свернуты по «безвременному отношению» Турчина, но при этом не соответствуют каноническому.

## 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 минимально возможной длине цепочки. При этом тестовые наборы стоит разделять на **позитивные** и **негативные.**

Позитивными называются 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 синтаксического 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 интерес в данной работе в конечном итоге представляет построение графа конфигураций для заданной входной грамматики и генерация набора тестовых цепочек при обходе такого графа. Для возможности построения графа 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 конфигураций.

В качестве отправной точки использовались материалы курсового проекта студента кафедры ИУ-9, в котором был реализован front-end компилятора на языке Python 2.7, получающий на вход описание КС-грамматики и, в случае, если она имеет вид , строящий для неё таблицу предсказывающего анализатора [3].

Исходная программа была переработана с целью передачи результата её работы, в случае успеха, в формате JSON на вход программе построения графа конфигураций. В противном случае на вход подается сообщение об ошибке.

## 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.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ) |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Рисунок 5. Таблица предсказывающего анализатора для грамматики арифметических выражений, дополненная расширенными правилами восстановления.

На рисунке 5 красным цветом помечены расширенные правила восстановления, а черным – обычные правила грамматики.

Однако, для корректной работы распознавателя с такой таблицей 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 для обычных нетерминалов: при замене терминалов на фиктивные нетерминалы, данные множества не изменяются, а при их конструировании лишь добавляется еще один транзитный шаг .

Множества для всех фиктивных нетерминалов состоят из единственного терминала, их породившего, а значит они не пересекаются.

Аналогично, множества останутся неизменными (за исключением фиктивных терминалов) и в таком случае не будут пересекаться с для правил вида , что в итоге позволяет судить о том, что расширенная правилами восстановления грамматика останется .

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.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ) |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  | ) |  |
|  |  |  |  |  | ) |  |
|  |  |  |  |  | ) |  |
|  |  |  |  |  | ) |  |
|  |  |  |  |  | **)** |  |
|  |  |  |  |  | ) |  |

Рисунок 6. Таблица предсказывающего анализатора для расширенной грамматики арифметических выражений.

Теперь настало время обсудить алгоритм построения графа конфигураций. Данный алгоритм на вход принимает сведения о грамматике, а именно:

1. Список всех терминалов грамматики.
2. Описание множеств для всех обычных нетерминалов.
3. Таблицу предсказывающего анализатора, дополненную правилами восстановления при ошибках и фиктивными нетерминалами.

Список терминалов потребуется во время построения всех возможных путей из конфигурационной вершины по каждому терминалу, а описание множеств в будущем упростит определение финальных вершин.

Говоря о конфигурационных вершинах, можно разделить их на три группы:

* Транзитные – не финальные и не псевдо-финальные вершины.
* Финальные – вершины с пустым стеком, либо со стеком из нетерминалов, каждый из которых может раскрываться в пустоту.
* Псевдо-финальные – вершины, не являющиеся финальными, из которых не имеется перехода по какому-либо из терминальных символов.

Итак, опишем формально алгоритм построение графа конфигураций:

Если обобщение или вложение не были обнаружены:

Здесь осознанно не определен процесс проверки на наличие вложения или обобщения, поскольку требует особого внимания.

Рассмотрим способ проверки на наличие вложения. Для этого необходимо существить поиск среди предков данной вершины до первого let-узла или корня дерева, в случае отсутствия таковых. Поиск считается успешным, если была найдена конфигурация, удовлетворяющая определению вложения для стеков. Проверка заключается в попытке выделения непустого подмножества из текущего стека, являющегося эквивалентов для стека родителя.

В случае, если вложение было зафиксировано, создается новая let-вершина, верхним потомком которой является найденная родительская конфигурация, а нижним – конфигурация с не совпавшим остатком стека.

В процессе поиска вложения также возможно искать и обобщение. Поскольку эти два механизма являются взаимоисключающими (в случае непустоты подмножеств стека), при наличии общего собственного префикса двух стеков имеет смысл сравнить их суффиксы, и, если максимальные общие суффикс и префикс в итоге образуют родительский стек –обобщение обнаружено. В таком случае, аналогично вложению, создается новая let-вершина…

* Удаление поддерева
* Эквивалентные конфигурации при вложении и обобщении
* Построение позитивных тестов
* Построение негативных тестов

# 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';

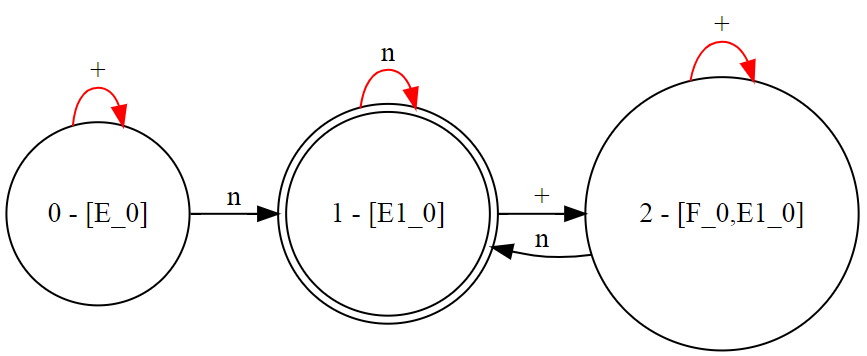


Рисунок 7. Граф конфигураций для грамматики сложений

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\*;

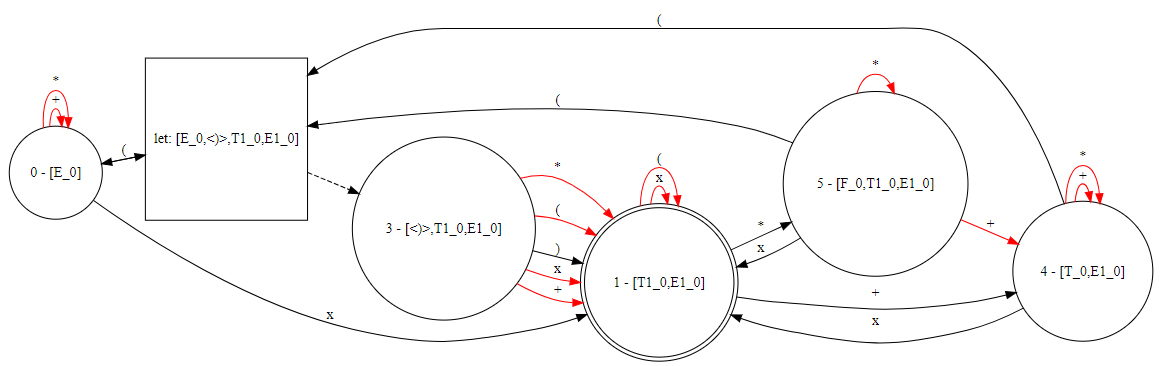


Рисунок 8. Граф конфигураций для грамматики арифметических выражений

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. Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы: принципы, технологии и инструментарий — 2 изд. — М.: Вильямс, 2008.
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/>.