МИНОБРНАУКИ РФ

ФГБОУ ВО Тверской государственный технический университет

Кафедра «Программного обеспечения».

Курсовая работа по дисциплине «Теория автоматов и формальных языков».

Тема: «Преобразование КС-грамматики без -правил в эквивалентную КС-грамматику без цепных правил».

Выполнил: студент группы

ПИН 17.06

Иванов Р.В

Проверил:

Кандидат физико-математических наук Карлов Б.Н

Тверь 2020

Оглавление

[Введение 3](#_Toc37697999)

[Аналитическая часть 4](#_Toc37698000)

[Постановка задачи 4](#_Toc37698001)

[Алгоритм 4](#_Toc37698002)

[Корректность алгоритма 4](#_Toc37698003)

[Проектная часть 5](#_Toc37698004)

[Описание программы 5](#_Toc37698005)

[Тестовый пример. 8](#_Toc37698006)

[Источники 10](#_Toc37698007)

# Введение

Для того, чтобы понять, что такое КС-грамматика нужно обратиться к определению Формальной грамматики.

* Формальный язык – множество символьных цепочек.
* Формальная грамматика – набор правил, с помощью которых порождаются цепочки формального языка.
* Формальные грамматики можно преобразовать в конечные распознаватели и обрабатывающие автоматы, которые распознают/транслируют соответствующие множества цепочек.

Пример формальной грамматики:

1. <предложение> → <подлежащее><сказуемое><дополнение>
2. <подлежащее>→<прилагательное><существительное>
3. <дополнение>→<прилагательно><существительное>
4. <сказуемое>→сдает
5. <прилагательное>→каждый
6. <существительное>→экзамен
7. <существительное>→студент

В результате получим такое предложение: «каждый студент сдает каждый экзамен».

Теперь, когда стало понятно, что из себя представляет формальная грамматика, можно рассмотреть КС грамматику.

Контекстно-свободной грамматикой (КС-грамматикой) называется система , где , − непересекающиеся конечные множества терминальных и нетерминальных символов (терминалов и нетерминалов) соответственно; − некоторый выделенный символ, называемый аксиомой грамматики, − конечное множество правил. Терминальные символы называются также основными, нетерминальные − вспомогательными символами.

Каждое правило в КС-грамматике имеет вид , где и

КС грамматика допускает наличие цепных правил.

Цепное правило – это правило вида ,где A и B – нетерминалы.

# Аналитическая часть

**Цель**: удалить из грамматики цепные правила.

## Постановка задачи

Пусть Γ — контекстно-свободная грамматика, содержащая цепные правила. Требуется построить эквивалентную грамматику Γ′, не содержащую цепных правил. Задача удаления цепных правил из грамматики возникает при попытке её приведения к нормальной форме Хомского.

## Алгоритм

**Цепная пара** (*unit pair*) — упорядоченная пара  в которой , используя только цепные правила.

Алгоритм удаления цепных правил из грамматики:

1. Найти все цепные пары в грамматике Γ.
2. Для каждой цепной пары  добавить в грамматику  все правила вида , где  — нецепное правило из .
3. Удалить все цепные правила

Найти все цепные пары можно по индукции:

**Базис.**  — цепная пара для любого нетерминала, так как  за ноль шагов.

**Индукция.** Если пара — цепная, и есть правило , то  — цепная пара.

Нетрудно понять, что такой алгоритм найдет все цепные правила грамматики Γ, и только их.

## Корректность алгоритма

Пусть — контекстно-свободная грамматика. — грамматика, полученная в результате применения алгоритма к . Тогда .

Покажем, что .

Пусть . Тогда имеет левое порождение . Где бы в левом порождении ни использовалось цепное правило, нетерминал в правой части становится крайним слева в выводимой цепочке и сразу же заменяется. Таким образом, левое порождение в можно разбить на последовательность шагов, в которых ноль или несколько цепных правил сопровождаются нецепным. Заметим, что любое нецепное правило, перед которым нет цепных, образует такой шаг. Но по построению каждый из этих шагов может быть выполнен одним её правилом. Таким образом, , то есть .

Покажем, что .

Пусть , то есть . Так как каждое правило эквивалентно последовательности из нуля или нескольких цепных правил , за которой следует нецепное правило из Γ, то из следует . Таким образом, каждый шаг порождения в может быть заменен одним или несколькими шагами в . Собрав эти последовательности шагов, получим, что то есть

# Проектная часть

**Задача**:

Написать программу для задачи:

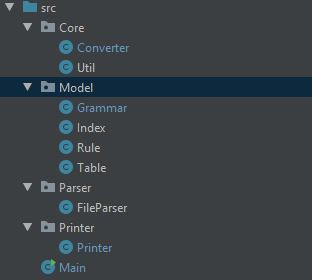
Преобразовать КС-грамматику без – правил в эквивалентную КС-грамматику без цепных правил

Разработать формат входных и выходных данных, удобный для человека.

Входные данные должны считываться из файла, выходные данные должны выводиться в файл. Прокомментировать каждый логически законченный фрагмент программы.

## Описание программы

Программа разрабатывалась на языке Java (8-ой версии).

Начнем с **package Model**-

Который содержит в себе модельные классы:

**Класс Grammar** – грамматика. Состоит из полей: аксиома и набора правил, так же имеет геторы для доступа к этим полям

**Класс Index** – служебный класс

В состав которого входят:

Поля: cntNT – количество нетерминалов; strNT – массив из нетерминалов;startPos – массив из индекса стартовых позиций нового терминала; chains – массив из количество цепных правил у одного нетерминала; nonChains – массив из количество не цепных правил у одного нетерминала; table – объект класса Table; Доступ к полям осуществлен посредством геторов и сеторов.

**Класс Rule –** правила. Состоит из полей: nt – нетерминал и terminalString - терминал; имеет геторы для доступа к этим полям.

**Класс Table** – таблица. Состоит из полей: table – двумерный массив, показывающий цепные правила, size – размер двумерного массива. Доступ к полям осуществлен посредством геторов и сеторов.

**Package Core** - содержит основные классы для работы с правилами.

**Класс Converter** - преобразует нашу грамматику в эквивалентную без цепных правил. Конструктор которого принимает на вход грамматику.

Публичный метод **eliminateChainRules() –** избавляет нашу грамматику от цепных правил, путем вызова приватных методов , а так же записывает результат в файл и выводит на консоль.

Приватный метод **presortGrammar(Grammar grammar, Index index)** -на вход принимает грамматику и индекс. Сортирует грамматику и устанавливает количество нетерминалов в индексе

Приватный метод **searchingForChainRules(Grammar grammar, Index index)** – на вход принимает грамматику и индекс. Устанавливает в индексе количество цепных и не цепных правил для каждого нетерминала, так же устанавливает их стартовые позиции.

Приватный метод **initTable(Grammar grammar, Index index)** - на вход принимает грамматику и индекс. Заполняет таблицу цепными правилами типа

Приватный метод **fillTable(Table matrix)** – на вход принимает уже существующую таблицу. Дополняет таблицу правилами вида: из этого следует .

Класс **Util** – утилитный класс.

Публичный метод **swap(List<Rule> list, int indexA, int indexB)** – на вход принимает список правил, и два индекса. Меняет местами правила в списке правил по индексу.

Публичный метод **int findPos(char[] strNt, char c)** – на вход принимает массив нетерминалов и терминал , возвращает позицию терминала в массиве, если позиция не найдена то возвращает -1.

**Package Parser** - содержит класс для работы с файлом.

Класс **FileParser** – класс для парсинга файла в грамматику.

Приватный метод **String readFile(String filePath)** – на вход принимает путь к файлу, возвращает содержимое файла в виде строки

Приватный метод **List<Rule> getRules(String filePath) –** на вход принимает путь к файлу, возвращает лист правил и устанавливает аксиому.

Публичный метод **Grammar getGrammar(String filePath) ) –** на вход принимает путь к файлу, возвращает грамматику.

**Package Printer** - содержит класс для работы с файлом.

Класс **Printer** – класс для парсинга файла в грамматику.

Публичный метод **printGrammar(String message, Grammar grammar) –** на вход подается сообщение и грамматика, метод печатает грамматику на консоль и заносит в файл.

Публичный метод **printTable(String message, Index index)–** на вход подается сообщение и индекс, метод печатает таблицу на консоль и заносит в файл.

Публичный метод **printGrammarWithoutChains(Grammar grammar, Index index) –** на вход подается грамматика и индекс, метод печатает грамматику без цепных правил на консоль и заносит в файл.

Публичный метод **flush()** – сохраняет изменения сделанные в файле.

## Тестовый пример.

На вход подается файл с грамматикой, где первая строчка – аксиома

Дальше идут правила вида: «нетерминал» -> «терминал» или

«нетерминал» -> «терминал» | «терминал» | «терминал»

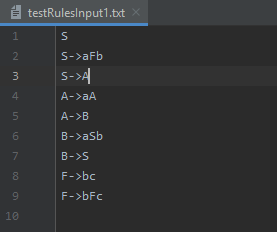
Например:

S->aba

S->A

Идентична записи

S->aba|A

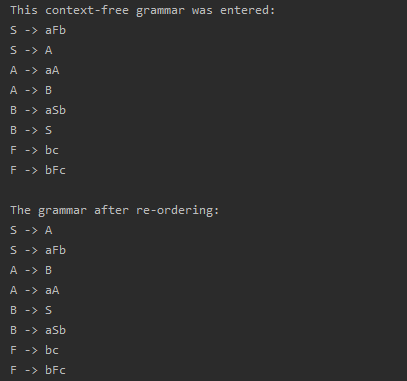


Main программы выглядит следующим образом

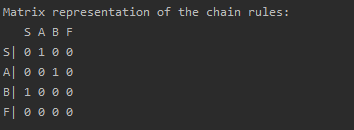
public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 try {  
 System.*out*.println("Введите путь к грамматике");  
 String path = scanner.nextLine();  
  
 Converter converter = new Converter(new FileParser().getGrammar(path),

new File(".").getAbsolutePath());  
 converter.eliminateChanRules();  
  
 }catch (Exception e)  
 {  
 System.*err*.println(e.toString());  
 }  
 scanner.nextLine();  
}

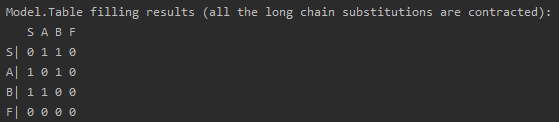
Сначала выводим введенную грамматику, потом сортируем ее



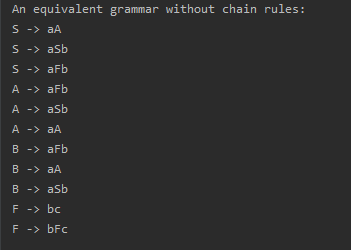
Выводим матрицу цепных правил:



Дополняем матрицу из ходя из правил выше:



Исходя из матрицы строим грамматику без цепных правил



# Источники

* [https://neerc.ifmo.ru/wiki/index.php?title=Удаление\_цепных\_правил\_из\_грамматики#.D0.9A.D0.BE.D1.80.D1.80.D0.B5.D0.BA.D1.82.D0.BD.D0.BE.D1.81.D1.82.D1.8C\_.D0.B0.D0.BB.D0.B3.D0.BE.D1.80.D0.B8.D1.82.D0.BC.D0.B0](https://neerc.ifmo.ru/wiki/index.php?title=Удаление_цепных_правил_из_грамматики%23.D0.9A.D0.BE.D1.80.D1.80.D0.B5.D0.BA.D1.82.D0.BD.D0.BE.D1.81.D1.82.D1.8C_.D0.B0.D0.BB.D0.B3.D0.BE.D1.80.D0.B8.D1.82.D0.BC.D0.B0)
* <https://ru.wikipedia.org/wiki/Контекстно-свободная_грамматика>
* Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции Том 1. Синтаксический анализ – Издательство: Мир, 1978. - 614 с.