2025 г.

ВАСИЛ ТОДОРОВ СТАНЧЕВ КС 2к

[име на фирмата]

Контекстно-свободна граматика

Документация към проектът по ООП

Съдържание

[Задание на курсовия проект: 2](#_Toc198576035)

[1. Увод 3](#_Toc198576036)

[1.1. Описание и идея на проекта 3](#_Toc198576037)

[1.2. Цел и задачи на разработката 3](#_Toc198576038)

[1.3. Структура на документацията 3](#_Toc198576039)

[2. Преглед на предметната област 4](#_Toc198576040)

[2.1. Основни концепции и алгоритми, които ще бъдат използвани 4](#_Toc198576041)

[2.2. Подходи, методи за решаване на поставените задачи (евентуално модели и стандарти) 4](#_Toc198576042)

[2.3. Функционални изисквания (потребителски изисквания, права, роли, статуси, диаграми и качествени (нефункционални) изисквания) 4](#_Toc198576043)

[3. Проектиране 5](#_Toc198576044)

[3.1. Обща структура на проекта (пакети, които ще се реализират) 5](#_Toc198576045)

[3.2. Схеми (фигура на действие) 5](#_Toc198576046)

[3.3. Идеи за организация на клас 5](#_Toc198576047)

[4. Реализация 6](#_Toc198576048)

[4.1. Реализация на класове (с JavaDoc - включва реализацията на функционалностите с техните класове и малки фрагменти от кода за важните функционалности) 6](#_Toc198576049)

[4.1.1. Граматика 6](#_Toc198576050)

[4.1.2. Контекстно свободна граматика 11](#_Toc198576051)

[4.1.3. Правило 29](#_Toc198576052)

[4.1.4. Команда 36](#_Toc198576053)

[4.1.5. Параметри към команда 39](#_Toc198576054)

[4.1.6. Запазване на файл по име 43](#_Toc198576055)

[4.2. Алгоритми и оптимизации. 46](#_Toc198576056)

[5. Тестване 47](#_Toc198576057)

[5.1. Планиране, описание и създаване на тестови сценарии (създаване на примери за всички възможн входове на командите) 47](#_Toc198576058)

[6. Заключение 47](#_Toc198576059)

[6.1. Обобщение на изпълнението на началните цели 47](#_Toc198576060)

[7. Използвана литература 47](#_Toc198576061)

Задание на курсовия проект: Картина, която съдържа текст, екранна снимка, Шрифт, номер

Генерираното от ИИ съдържание може да е неправилно.

# Увод

# Описание и идея на проекта

Този проект има за цел да създаде приложение за работа с контекстно-свободни граматики (КСГ). Това позволява създаване, манипулиране и анализ на КСГ, операции между различни граматики, преобразуване на КСГ граматика в нормализирана форма на Чомски(НФЧ) и други. Проектът цели да бъде образователен ресурс и практичен инструмент за разработчици и студенти, изучаващи теория на формалните езици и компилатори. Контекстно-свободните граматики са фундаментална концепция в компютърните науки, използвана за описание на синтаксиса на програмни езици и други формални езици.

# Цел и задачи на разработката

Основната цел на този проект е да създаде гъвкава и лесна за използване *Java* реализация, която да поддържа основни операции с КСГ. Основните задачи включват:

1. Дефиниране на класове за представяне на граматики, нетерминали, терминали и правила.
2. Реализиране на функционалност за добавяне, премахване и редактиране на граматически правила.
3. Възможност за преобразуване на КСГ в нормална форма на Чомски (НФЧ).
4. Реализация на алгоритъма CYK (Cocke-Younger-Kasami) за проверка за принадлежност на низ към езика на КСГ.
5. Предоставяне на прост интерфейс за взаимодействие с приложението.

# Структура на документацията

Настоящата документация е структурирана по следния начин:

1. Увод: Предоставя общо описание на проекта, неговите цели и структурата на документацията.
2. Преглед на предметната област: Описва основните концепции, алгоритми и изисквания, свързани с контекстно-свободните граматики.
3. Проектиране: Детайлизира архитектурата на проекта.
4. Реализация: Описва реализацията на основните класове и алгоритми с фрагменти от код и JavaDoc.
5. Тестване: Очертава плановете и сценариите за тестване на приложението.
6. Заключение: Обобщава постигнатите резултати и бъдещите насоки за развитие.
7. Използвана литература: Изброява всички източници, използвани при разработването на проекта и документацията.

# Преглед на предметната област

# Основни концепции и алгоритми, които ще бъдат използвани

1. Контекстно-свободна граматика (КСГ): Формална граматика, в която всяко производствено правило е от вида A → α, където A е нетерминален символ, а α е низ от терминали и/или нетерминали.
2. Терминални символи: Основните символи на езика, които не могат да бъдат заменени с други символи.
3. Нетерминални символи: Символи, които могат да бъдат заменени с други символи според граматическите правила. Един от нетерминалните символи е началният символ на граматиката.
4. Правила (продукции): Определят как нетерминалните символи могат да бъдат заменени с низове от терминали и/или нетерминали.
5. Език, генериран от граматика: Крайно множеството от низове от терминални символи, които могат да бъдат получени чрез прилагане на граматическите правила, започвайки от началния символ.
6. Нормална форма на Чомски (НФЧ): Контекстно-свободна граматика, в която всички правила са в една от следните форми: A → BC, A → a или S → ε, където A, B и C са нетерминали, a е терминал, а S е началният символ, който може да произвежда празен низ ε. Преобразуването в НФЧ е важно за много алгоритми за анализ на КСГ.
7. Алгоритъм CYK (Cocke-Younger-Kasami): Алгоритъм за динамично програмиране, който проверява дали даден низ може да бъде генериран от дадена контекстно-свободна граматика, която е в нормална форма на Чомски. Алгоритъмът създава таблица, която съдържа множества от нетерминали, които могат да генерират поднизове на входния низ.

# Подходи, методи за решаване на поставените задачи (евентуално модели и стандарти)

Проектът ще използва обектно-ориентиран подход за моделиране на концепциите на контекстно-свободните граматики. Ще бъдат създадени класове за представяне на граматики, правила и команди. Алгоритъмът CYK ще бъде реализиран съгласно стандартния подход за динамично програмиране. Преобразуването в нормална форма на Чомски ще следва стандартните стъпки за елиминиране на ε-продукции, единични продукции и съкращаване на дълги продукции.

# Функционални изисквания (потребителски изисквания, права, роли, статуси, диаграми и качествени (нефункционални) изисквания)

1. Потребителят трябва да може да създава празна контекстно-свободна граматика и да добавя към нея терминали и нетерминали.
2. Потребителят трябва да може да дефинира граматически правила, като указва нетерминала от лявата страна и низа от символи от дясната страна.
3. Потребителят трябва да може да премахва граматически правила по техния номер.
4. Приложението трябва да предоставя възможност за преобразуване на дадена КСГ в еквивалентна граматика в нормална форма на Чомски.
5. Потребителят трябва да може да провери дали даден низ принадлежи към езика, генериран от дадена КСГ (след преобразуване в НФЧ), използвайки алгоритъма CYK.
6. Приложението трябва да предоставя методи за извеждане на информация за граматиката, като списък на терминалите, нетерминалите и правилата.
7. Проектът ще използва интерфейс на командния ред за взаимодействие с потребителя, както е видно от предоставените класове.

# Проектиране

# Обща структура на проекта (пакети, които ще се реализират)

1. Граматика: Съдържа класове, които представляват основните концепции на граматиките, като „*Grammar“*, „*ContextSensitiveGrammar“* и „*Rules“*.
2. Команди: Съдържа класове, които реализират различните команди, които потребителят може да изпълнява чрез интерфейса на командния ред (например „*AddGrammar“*, „*AddRule“*, „*RemoveRule“*, „*Cyk“*).

# Схеми (фигура на действие)

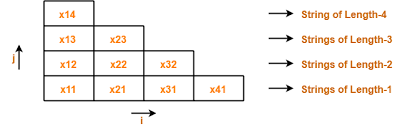


Figure 1

1. Фигура 1 демонстрира основната концепция зад *CYK* алгоритъма, при който се изгражда квадратна матрица, като всеки ред отговаря съответно на дължината на възможната дума и всяко блокче отговаря на различна комбинация от нетерминали.

# Идеи за организация на клас

1. Основният клас „*Grammar*“ ще бъде абстрактен и ще съдържа общи свойства и методи за всички видове граматики (терминали, нетерминали, правила, език, *ID*). Класът „*ContextSensitiveGrammar*“ ще наследява „*Grammar*“ и ще съдържа специфична функционалност за контекстно-свободни граматики, като методи за преобразуване в НФЧ и прилагане на алгоритъма *CYK*. Класът „*Rules*“ ще представлява едно граматическо правило и ще съдържа информация за неговия номер, нетерминал от лявата страна и низ от символи от дясната страна.
2. Пакетът за командите ще използва абстрактен клас „*Command*“, в който ще бъде извикван един единствен метод за изпълнение „*execute*“, като се използва идеята за полиморфизъм, като всяка команда ще го пренаписва сама за себе си. Класът „*CommandParameters*“ ще служи за предаване на параметри към командите, както и за предаване на текущия списък с граматиките.

# Реализация

# Реализация на класове (с JavaDoc - включва реализацията на функционалностите с техните класове и малки фрагменти от кода за важните функционалности)

* Ще бъдат представени основните класове, както и един клас от командите, който ще демонстрира пренаписването на метода „*execute*“.

# Граматика

package grammar;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.Random;

public abstract class Grammar {

protected int id;

protected List<Character> terminals;

protected List<Character> nonTerminals;

protected List<Rules> rules;

protected List<String> language;

/\*\*

\* Конструктор, който създава нова граматика с даден списък с терминали

\* @param terminals

\*/

Grammar(List<Character> terminals){

id = generateId();

setTerminals(terminals);

nonTerminals = new ArrayList<>();

rules = new ArrayList<>();

language = new ArrayList<>();

}

/\*\*

\* Конструктор, който създава нова граматика с всичките атрибути

\* @param terminals

\*/

Grammar(List<Character> terminals, List<Character> nonTerminals, List<Rules> rules, List<String> language){

id = generateId();

setTerminals(terminals);

setNonTerminals(nonTerminals);

setRules(rules);

setLanguage(language);

}

/\*\*

\* Метод, който създава уникален идентификатор в граница(1-1000)

\* @return

\*/

public int generateId(){

Random random = new Random();

int minNum = 1;

int maxNum = 1000;

return random.nextInt(maxNum - minNum + 1) + minNum;

}

/\*\*

\* Метод, който връща уникалния идентификатор на дадената граматика

\* @return

\*/

public int getId() {

return id;

}

/\*\*

\* Метод, който връща копие на списъка от терминали на дадената граматика

\* @return

\*/

public List<Character> getTerminals() {

return Collections.unmodifiableList(terminals);

}

/\*\*

\* Метод, който връща копие на списъка от нетерминали на дадената граматика

\* @return

\*/

public List<Character> getNonTerminals() {

return Collections.unmodifiableList(nonTerminals);

}

/\*\*

\* Метод, който връща копие на списъка от правила на дадената граматика

\* @return

\*/

public List<Rules> getRules() {

return Collections.unmodifiableList(rules);

}

/\*\*

\* Метод, който връща копие на списъка от терминали на дадената граматика

\* @return

\*/

public List<String> getLanguage() {

return Collections.unmodifiableList(language);

}

/\*\*

\* Метод, който обновява списъка с терминалит

\* @param terminals

\*/

public void setTerminals(List<Character> terminals) {

this.terminals = terminals != null ? new ArrayList<>(terminals) : new ArrayList<>();

}

/\*\*

\* Метод, който обновява списъка с нетерминалит

\* @param nonTerminals

\*/

public void setNonTerminals(List<Character> nonTerminals) {

this.nonTerminals = nonTerminals != null ? new ArrayList<>(nonTerminals) : new ArrayList<>();

}

/\*\*

\* Метод, който обновява списъка с правила

\* @param rules

\*/

public void setRules(List<Rules> rules) {

this.rules = rules != null ? new ArrayList<>(rules) : new ArrayList<>();

}

/\*\*

\* Метод, който обновява езика

\* @param language

\*/

public void setLanguage(List<String> language) {

this.language = language != null ? new ArrayList<>(language) : new ArrayList<>();

}

}

# Контекстно свободна граматика

package grammar;

import commands.CustomException;

import java.io.Serializable;

import java.util.\*;

public class ContextSensitiveGrammar extends Grammar implements Serializable {

/\*\*

\* Конструктор, който създава нова КС граматика с даден списък с терминали

\* @param terminals

\*/

public ContextSensitiveGrammar(List<Character> terminals){

super(terminals);

}

/\*\*

\* Конструктор, който създава нова КС граматика с всичките атрибути

\* @param terminals

\*/

public ContextSensitiveGrammar(List<Character> terminals, List<Character> nonTerminals, List<Rules> rules, List<String> language){

super(terminals,nonTerminals,rules,language);

}

/\*\*

\* Метод, който връща дадена граматика по даден уникален идентификатор

\* @param id

\* @param grammars

\* @return

\* @throws CustomException

\*/

public static ContextSensitiveGrammar returnGrammarById(int id, List<ContextSensitiveGrammar> grammars) throws CustomException{

for (int i = 0; i < grammars.size(); i++) {

if (grammars.get(i).getId()==id){

return grammars.get(i);

}

}

throw new CustomException("The id doesn't exist");

}

/\*\*

\* Метод, който добавя нова граматика в списъка с граматики

\* @param grammars

\* @param args

\* @throws CustomException

\*/

public static void addGrammar(List<ContextSensitiveGrammar> grammars, List<String> args)throws CustomException {

if (args.isEmpty()) {

throw new CustomException("There weren't any terminals provided");

}

List<Character> terminals = new ArrayList<>();

int count = args.size();

for (int i = 0; i < count; i++)

{

char terminal = args.get(i).charAt(0);

terminals.add(terminal);

}

ContextSensitiveGrammar grammar = new ContextSensitiveGrammar(terminals);

grammars.add(grammar);

}

/\*\*

\* Метод, който отпечатва всичките правила за дадена граматика

\* @param rules

\*/

public static void printRules(List<Rules> rules){

for (int i = 0; i < rules.size(); i++) {

rules.get(i).printRuleInfo();

}

}

/\*\*

\* Метод, който взима нетерминалната част от дадена описателна част на правило

\* @param describingPart

\*/

private void extractNonTerminal(String describingPart){

for (char letter : describingPart.toCharArray()) {

if (Character.isUpperCase(letter)){

if (!nonTerminals.contains(letter)) {

nonTerminals.add(letter);

}

}

}

}

/\*\*

\* Метод, който проверява дали терминалите в дадената описателна част са част от азбуката на граматиката

\* @param describingPart

\* @return

\* @throws CustomException

\*/

private boolean checkDescribingPart(String describingPart)throws CustomException{

if(describingPart.equals("final")){

return true;

}

for (char character: describingPart.toCharArray()) {

if (Character.isLowerCase(character)) {

if (!this.terminals.contains(character)) {

return false;

}

}

}

return true;

}

/\*\*

\* Метод, който добавя ново правило

\* @param describingPart

\* @throws CustomException

\*/

public void addRule(String describingPart) throws CustomException{

if (!this.checkDescribingPart(describingPart)){

throw new CustomException("The terminals inside the describing part of the rule don't match with the entered alphabet!");

}

Scanner scanner = new Scanner(System.in);

if (rules.isEmpty()){

Rules rule = new Rules(1,'S', describingPart);

rules.add(rule);

nonTerminals.add('S');

this.extractNonTerminal(describingPart);

}

else {

int number = rules.size() + 1;

System.out.println("Choose a non-terminals to begin the rule:");

System.out.printf("{ ");

for (int i = 0; i < nonTerminals.size(); i++) {

System.out.printf("%c ",nonTerminals.get(i));

}

System.out.printf("}");

System.out.println();

char nonTerminal = scanner.next().charAt(0);

boolean isTrue = false;

for (int i = 0; i < nonTerminals.size(); i++){

if (nonTerminals.get(i).equals(nonTerminal)){

Rules rule = new Rules(number, nonTerminal, describingPart);

rules.add(rule);

this.extractNonTerminal(describingPart);

isTrue = true;

}

}

if (!isTrue) {

throw new CustomException("The non-terminal is not inside the list of non-terminals, the rule was not added");

}

}

}

/\*\*

\* Метод, който премахва дадено правило по дадено поредно число

\* @param ruleNumber

\* @throws CustomException

\*/

public void removeRule(int ruleNumber) throws CustomException {

Rules ruleToRemove = Rules.getRuleByNumber(this.rules, ruleNumber);

if (ruleToRemove == null) {

throw new CustomException("No rule found with number: " + ruleNumber);

}

this.rules.remove(ruleToRemove);

System.out.println("Rule №" + ruleNumber + " was removed");

}

/\*\*

\* Метод, който отпечатва уникалния идентификатор на граматиката

\*/

public void listId(){

System.out.printf("Grammar id: %d\n", this.id);

}

/\*\*

\* Метод, който проверява дали езикът в дадена граматика е празен

\* @param grammars

\* @param id

\*/

public static void isLanguageEmpty(List<ContextSensitiveGrammar> grammars, int id){

for (int i = 0; i < grammars.size(); i++) {

if (grammars.get(i).getId() == id){

if (grammars.get(i).getLanguage().isEmpty()){

System.out.println("The language in this grammar is empty");

}

}

}

}

/\*\*

\* Метод, който отпечатва всичките начални правила, като дава право на потребителя да направи избор за начално правило

\* @return

\*/

private int showRulesWithS(){

Scanner scanner = new Scanner(System.in);

int number = 0;

System.out.println("Choose between these rules to begin the word:");

for (int i = 0; i < rules.size(); i++) {

if(rules.get(i).getNonTerminal() == 'S'){

rules.get(i).printRuleInfo();

}

}

System.out.println("Number: ");

number = Integer.parseInt(scanner.nextLine());

for (int i = 0; i < rules.size(); i++) {

if(rules.get(i).getNonTerminal() == 'S' && rules.get(i).getNumber() == number){

return rules.get(i).getNumber();

}

}

return 0;

}

/\*\*

\* Метод, който добавя нова дума в езикът на дадена граматика

\* @throws CustomException

\*/

public void addWord() throws CustomException{

Scanner scanner = new Scanner(System.in);

boolean isPossible = Rules.checkForFinal(rules);

if (isPossible) {

int number = showRulesWithS();

Rules rule = Rules.getRuleByNumber(rules, number);

String word = rule.getDescribingPart();

do {

System.out.printf("Current word: %s\n", word);

ContextSensitiveGrammar.printRules(rules);

System.out.println("Enter the number of the rule you want to use:");

number = Integer.parseInt(scanner.nextLine());

rule = Rules.getRuleByNumber(rules, number);

if (rule == null){

throw new CustomException("Rule doesn't exist");

}

else{

word = Rules.connectTerminals(rule,word);

}

if (rule.getDescribingPart().equals("final")){

word=word.replace("final","");

}

} while (!rule.getDescribingPart().equals("final"));

language.add(word);

}

else{

throw new CustomException("There isn't any end rule");

}

}

/\*\*

\* Метод, който превръща дадена граматика да е в нормална форма на Чомски

\* @param grammar

\* @return

\* @throws CustomException

\*/

public static ContextSensitiveGrammar convertToChomskyNormalForm(ContextSensitiveGrammar grammar) throws CustomException {

if (Rules.isInChomskyNormalForm(grammar.getRules())) {

throw new CustomException("Grammar is already in Chomsky normal form.");

}

List<Rules> newRules = new ArrayList<>();

List<Character> newNonTerminals = new ArrayList<>(grammar.getNonTerminals());

List<Character> terminals = grammar.getTerminals();

Map<Character, Character> terminalMap = new HashMap<>();

int ruleCounter = 1;

char newNonTerminalChar = grammar.getNonTerminals().get(grammar.getNonTerminals().size() - 1);

for (Rules rule : grammar.getRules()) {

String rhs = rule.getDescribingPart();

if (rhs.equals("final")) {

newRules.add(new Rules(ruleCounter++, rule.getNonTerminal(), rhs));

continue;

}

if (rhs.length() > 1) {

StringBuilder newRhs = new StringBuilder();

for (char symbol : rhs.toCharArray()) {

if (terminals.contains(symbol)) {

if (!terminalMap.containsKey(symbol)) {

while (newNonTerminals.contains(newNonTerminalChar)) {

newNonTerminalChar++;

}

terminalMap.put(symbol, newNonTerminalChar);

newNonTerminals.add(newNonTerminalChar);

newRules.add(new Rules(ruleCounter++, newNonTerminalChar, String.valueOf(symbol)));

newNonTerminalChar++;

}

newRhs.append(terminalMap.get(symbol));

} else {

newRhs.append(symbol);

}

}

newRules.add(new Rules(ruleCounter++, rule.getNonTerminal(), newRhs.toString()));

} else {

newRules.add(new Rules(ruleCounter++, rule.getNonTerminal(), rhs));

}

}

List<Rules> finalRules = new ArrayList<>();

for (Rules rule : newRules) {

String rhs = rule.getDescribingPart();

if (rhs.equals("final")) {

finalRules.add(rule);

continue;

}

if (rhs.length() <= 2) {

finalRules.add(rule);

} else {

char currentLhs = rule.getNonTerminal();

String remainingRhs = rhs;

while (remainingRhs.length() > 2) {

char first = remainingRhs.charAt(0);

while (newNonTerminals.contains(newNonTerminalChar)) {

newNonTerminalChar++;

}

char newNonTerminal = newNonTerminalChar;

newNonTerminals.add(newNonTerminal);

finalRules.add(new Rules(ruleCounter++, currentLhs, "" + first + newNonTerminal));

currentLhs = newNonTerminal;

remainingRhs = remainingRhs.substring(1);

newNonTerminalChar++;

}

finalRules.add(new Rules(ruleCounter++, currentLhs, remainingRhs));

}

}

return new ContextSensitiveGrammar(terminals, newNonTerminals, finalRules, grammar.getLanguage());

}

/\*\*

\* Метод, който проверява дали дадена дума може да бъде генерирана в граматика, която е в нормална форма на Чомски(Използвайки CYK алгоритъм)

\* @param grammar

\* @param word

\* @return

\* @throws CustomException

\*/

public static boolean isWordInLanguage(ContextSensitiveGrammar grammar, String word) throws CustomException{

if (!Rules.isInChomskyNormalForm(grammar.getRules())) {

throw new CustomException("Grammar isn't in chomsky normal form.");

}

int n = word.length();

List<Rules> rules = grammar.getRules();

Set<Character> nonTerminals = new HashSet<>(grammar.getNonTerminals());

char startSymbol = 'S';

Set<Character>[][] P = new HashSet[n][n];

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

P[i][j] = new HashSet<>();

}

}

for (int i = 0; i < n; i++) {

char terminal = word.charAt(i);

for (Rules rule : rules) {

if (rule.getDescribingPart().length() == 1 && rule.getDescribingPart().charAt(0) == terminal) {

P[i][i].add(rule.getNonTerminal());

}

}

}

for (int l = 2; l <= n; l++) {

for (int i = 0; i <= n - l; i++) {

int j = i + l - 1;

for (int k = i; k < j; k++) {

for (Rules rule : rules) {

String describingPart = rule.getDescribingPart();

if (describingPart.length() == 2) {

char A = describingPart.charAt(0);

char B = describingPart.charAt(1);

if (P[i][k].contains(A) && P[k + 1][j].contains(B)) {

P[i][j].add(rule.getNonTerminal());

}

}

}

}

}

}

return P[0][n - 1].contains(startSymbol);

}

/\*\*

\* Метод, който прави действието "Звезда на Клини" върху езика на дадена граматика

\* @param grammar

\* @return

\*/

public static ContextSensitiveGrammar kleeneStar(ContextSensitiveGrammar grammar) {

int maxLength = 3;

Set<String> result = new HashSet<>();

result.add("");

Set<String> current = new HashSet<>();

current.add("");

for (int i = 1; i <= maxLength; i++) {

Set<String> next = new HashSet<>();

for (String prefix : current) {

for (char ch : grammar.getTerminals()) {

next.add(prefix + ch);

}

}

result.addAll(next);

current = next;

}

List<String> resultList = new ArrayList<>(result);

List<String> language = new ArrayList<>(grammar.getLanguage());

language.addAll(resultList);

return new ContextSensitiveGrammar(grammar.getTerminals(), grammar.getNonTerminals(), grammar.getRules(),language);

}

}

# Правило

package grammar;

import java.util.List;

public class Rules {

private int number;

private char nonTerminal;

private String describingPart;

/\*\*

\* Конструктор, който приема параметрите за дадено правило

\* @param number

\* @param nonTerminal

\* @param describingPart

\*/

Rules(int number, char nonTerminal, String describingPart)

{

setNumber(number);

setNonTerminal(nonTerminal);

setDescribingPart(describingPart);

}

/\*\*

\* Метод, който връща поредния номер на дадено правило

\* @return

\*/

public int getNumber() {

return number;

}

/\*\*

\* Метод, който връща нетерминалната част на дадено правило

\* @return

\*/

public char getNonTerminal() {

return nonTerminal;

}

/\*\*

\* Метод, който връща описателната част на дадено правило

\* @return

\*/

public String getDescribingPart() {

return describingPart;

}

/\*\*

\* Метод, който променя поредния номер на дадено правило

\* @return

\*/

public void setNumber(int number) {

this.number = number;

}

/\*\*

\* Метод, който променя нетерминалната част на дадено правило

\* @return

\*/

public void setNonTerminal(char nonTerminal) {

this.nonTerminal = nonTerminal;

}

/\*\*

\* Метод, който променя описателната част на дадено правило

\* @return

\*/

public void setDescribingPart(String describingPart) {

this.describingPart = describingPart;

}

/\*\*

\* Метод, който връща стринг съдържащ информация за дадено правило

\* @return

\*/

@Override

public String toString() {

StringBuilder stringBuilder = new StringBuilder();

stringBuilder.append(String.format("Rule №%d: ", this.getNumber()));

stringBuilder.append(String.format("%c -> %s", this.getNonTerminal(), this.getDescribingPart()));

stringBuilder.append("\n");

return stringBuilder.toString();

}

/\*\*

\* Метод, който изписва информацията за дадено правило

\*/

void printRuleInfo(){

System.out.println(this.toString());

}

public static Rules getRuleByNumber(List<Rules> rules, int number){

for (int i = 0; i < rules.size(); i++) {

if (rules.get(i).getNumber() == number){

return rules.get(i);

}

}

return null;

}

/\*\*

\* Метод, който проверява дали има финално правило

\* @param rules

\* @return

\*/

public static boolean checkForFinal(List<Rules> rules){

for (int i = 0; i < rules.size(); i++) {

if (rules.get(i).getDescribingPart().equals("final")){

return true;

}

}

if (rules.isEmpty()){

System.out.println("There aren't any rules added");

return false;

}

return false;

}

/\*\*

\* Метод, който съединява терминали и връща стринг, който представлява нова дума

\* @param rule

\* @param word

\* @return

\*/

public static String connectTerminals(Rules rule, String word){

String newWord;

String character = Character.toString(rule.getNonTerminal());

if (word.contains(character)){

newWord = word.replaceAll(character, rule.getDescribingPart());

return newWord;

}

else{

return word;

}

}

/\*\*

\* Метод, който проверява дали дадените правила на дадена граматика са в нормалната форма на Чомски

\* @param rules

\* @return

\*/

public static boolean isInChomskyNormalForm(List<Rules> rules) {

for (Rules rule : rules) {

String describingPart = rule.getDescribingPart();

if (describingPart.length() == 1 && Character.isLowerCase(describingPart.charAt(0))) {

continue;

}

if (describingPart.length() == 2 && Character.isUpperCase(describingPart.charAt(0)) && Character.isUpperCase(describingPart.charAt(1))) {

continue;

}

if (describingPart.equals("final")) {

continue;

}

return false;

}

return true;

}

}

# Команда

package commands;

import java.io.\*;

import java.util.\*;

import java.util.function.Supplier;

import grammar.ContextSensitiveGrammar;

public abstract class Command {

private static final Map<String, Supplier<? extends Command>> commands = new HashMap<>();

static {

commands.put("list", ListId::new);

commands.put("print", PrintGrammar::new);

commands.put("printAll", PrintAllGrammars::new);

commands.put("help", Help::new);

commands.put("addGrammar", AddGrammar::new);

commands.put("addRule", AddRule::new);

commands.put("removeRule", RemoveRule::new);

commands.put("addWord", AddWord::new);

commands.put("union", Union::new);

commands.put("concat", Concat::new);

commands.put("chomsky", Chomsky::new);

commands.put("cyk", Cyk::new);

commands.put("iter", Iter::new);

commands.put("chomskify", Chomskify::new);

commands.put("empty", Empty::new);

commands.put("closeFile", CloseFile::new);

commands.put("openFile", OpenFile::new);

commands.put("saveAsFile", SaveAsFile::new);

commands.put("save", SaveFile::new);

}

/\*\*

\* Извикване на метода за командите, когато няма подадени параметри. Създава нова инстанция с "null" стойности

\* @throws CustomException

\*/

public void execute()throws CustomException {

execute(new CommandParameters(null, null,null));

}

/\*\*

\* Абстрактен метод за изпълнение на команда с параметри.

\* @param parameters

\* @throws CustomException

\*/

public abstract void execute(CommandParameters parameters) throws CustomException;

/\*\*

\* Главният метод за цялата програма, който служи като главно пространство за обработка на командите и техните параметри

\* @param grammars

\* @throws CustomException

\*/

public static void instructions(List<ContextSensitiveGrammar> grammars) throws CustomException{

Scanner scanner = new Scanner(System.in);

String commandLine = "";

String commandName = "";

CommandParameters context = new CommandParameters();

File currentFile = null;

System.out.println("Type help to see the available commands:");

while(!context.getCommand().contains("exit")){

try {

System.out.println("Enter command:");

System.out.printf("> ");

commandLine = scanner.nextLine();

context = new CommandParameters(grammars, commandLine,currentFile);

context.setFile(currentFile);

commandName = context.getCommand();

Supplier<? extends Command> commandSupplier = commands.get(commandName);

if (commandSupplier != null) {

Command command = commandSupplier.get();

command.execute(context);

grammars = context.getGrammars();

currentFile = context.getFile();

} else if (commandName.contains("exit")) {

System.out.println("Exiting...");

} else {

System.out.println("Enter a valid command. Type help to see the full list!");

}

} catch (Exception e){

System.out.println("An error occurred: " + e.getMessage());

}

}

}

}

# Параметри към команда

package commands;

import grammar.ContextSensitiveGrammar;

import java.io.\*;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class CommandParameters {

private List<ContextSensitiveGrammar> grammars;

private List<String> args;

private String command;

private File file;

/\*\*

\* Конструктор без параметри, който инициализира празна команда

\*/

public CommandParameters(){

command="";

}

/\*\*

\* Конструктор за подадените параметри, който зарежда всяка променлива в дадения атрибут

\* @param grammars

\* @param commandLine

\* @throws CustomException

\*/

public CommandParameters(List<ContextSensitiveGrammar> grammars, String commandLine, File file)throws CustomException {

this.grammars = grammars;

this.args=new ArrayList<>();

if (commandLine == null || commandLine.isEmpty()){

throw new CustomException("Command line is missing");

}

String[] commandElements = commandLine.split(" ");

command = commandElements[0];

for (int i = 1; i < commandElements.length; i++) {

this.args.add(commandElements[i]);

}

this.file=file;

}

/\*\*

\* Метод, който връща списък от граматики

\* @return

\*/

public List<ContextSensitiveGrammar> getGrammars() {

return grammars;

}

/\*\*

\* Метод, който връща списъка от аргументите към дадена команда

\* @return

\*/

public List<String> getArgs() {

return args;

}

/\*\*

\* Метод, който връща името на дадена команда

\* @return

\*/

public String getCommand() {

return command;

}

/\*\*

\* Метод, който връща обектът към даден файл

\* @return

\*/

public File getFile() {

return file;

}

/\*\*

\* Метод, който обновява обекта към даден файл

\* @param file

\*/

public void setFile(File file) {

this.file = file;

}

/\*\*

\* Метод, който обновява списъка с граматики

\* @param grammars

\*/

public void setGrammars(List<ContextSensitiveGrammar> grammars) {

this.grammars = grammars;

}

}

# Запазване на файл по име

package commands;

import grammar.ContextSensitiveGrammar;

import java.io.\*;

import java.util.ArrayList;

import java.util.List;

public class SaveFile extends Command{

/\*\*

\* Метод за записване на списък от граматики във файл

\* @param grammars

\* @param file

\*/

public static void saveList(List<ContextSensitiveGrammar> grammars, File file) {

try {

File parentDir = file.getParentFile();

if (parentDir != null && !parentDir.exists()) {

boolean dirsCreated = parentDir.mkdirs();

if (!dirsCreated) {

System.out.println("Failed to create directories.");

return;

}

}

try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(file))) {

out.writeObject(grammars);

System.out.println("The grammars were saved successfully");

}

} catch (IOException e) {

System.out.println("Error while saving: " + e.getMessage());

}

}

/\*\*

\* Метод за записване на граматика във файл

\* @param grammar

\* @param file

\*/

public void saveGrammar(ContextSensitiveGrammar grammar, File file) {

try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(file))) {

out.writeObject(grammar);

System.out.println("The grammar was save successfully");

} catch (IOException e) {

System.out.println("Error while saving: " + e.getMessage());

}

}

/\*\*

\* Пренаписан метод, който извиква метода за записване във файл, като предава името на файла, въведен от потребителя

\* @param parameters

\* @throws CustomException

\*/

@Override

public void execute(CommandParameters parameters) throws CustomException {

List<String> args = parameters.getArgs();

List<ContextSensitiveGrammar> grammars = parameters.getGrammars();

if (args.isEmpty()){

SaveFile.saveList(grammars, parameters.getFile());

}

else if (args.size() != 2){

throw new CustomException("Not enough parameters were provided.");

}

else {

try {

int id = Integer.parseInt(args.get(0));

File newFile = OpenFile.resolveFilePath(args.get(1));

ContextSensitiveGrammar grammar = ContextSensitiveGrammar.returnGrammarById(id, grammars);

this.saveGrammar(grammar, newFile);

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

}

}

# Алгоритми.

1. Преобразуване в нормална форма на Чомски: Този алгоритъм ще бъде реализиран в метода „*convertToChomskyNormalForm*“ на класа „*ContextSensitiveGrammar*“. Той ще следва стандартните стъпки за премахване на епсилон-продукции, единични продукции и замяна на продукции с повече от два нетерминала в дясната страна.
2. Алгоритъм CYK: Алгоритъмът CYK ще бъде реализиран в метода „*isWordInLanguage*“ на класа „*ContextSensitiveGrammar*“. Той ще използва динамично програмиране за изграждане на таблица, която показва кои нетерминали могат да генерират поднизове на входната дума. Времевата сложност на стандартния алгоритъм CYK е O(n^3 \* |G|), където n е дължината на входния низ, а |G| е броят на правилата в граматиката.

# Тестване

# Планиране, описание и създаване на тестови сценарии (създаване на примери за всички възможн входове на командите)

1. Ще бъдат създадени тестови сценарии за следните функционалности:
2. Създаване на граматики с различни набори от терминали и нетерминали.
3. Добавяне и премахване на валидни и невалидни граматически правила.
4. Преобразуване на прости и сложни КСГ в нормална форма на Чомски.
5. Проверка за принадлежност на низове към езика на граматиката с помощта на алгоритъма CYK за различни случаи (думи, които принадлежат и такива, които не принадлежат към езика).
6. Тестване на интерфейса на командния ред с различни комбинации от команди и параметри, включително случаи на грешни входни данни.
7. Примерни тестови сценарии за командите:
8. “addGrammar a b”: Очаква се да създаде нова граматика с терминали 'a' и 'b'.
9. „addRule <id> AB: Очаква се да добави правило S -> AB към текущата граматика.
10. „removeRule <id> 1“: Очаква се да премахне правило номер 1 от текущата граматика.
11. „cyk abc“: Очаква се да провери дали низът "abc" принадлежи към езика на текущата граматика.

# Заключение

# Обобщение на изпълнението на началните цели

Този документ имаше за цел да демонстрира идея за създаване на *Java* приложение за работа с контекстно-свободни граматики, предоставяйки основни функционалности като създаване, манипулиране, преобразуване в нормална форма на Чомски и проверка за принадлежност на низ с помощта на алгоритъма CYK. Предоставената структура и класове служат като добра основа за реализацията на тези цели, като са лесни за реализация и разбиране.

# Използвана литература

* <https://bg.wikipedia.org/wiki/%D0%99%D0%B5%D1%80%D0%B0%D1%80%D1%85%D0%B8%D1%8F_%D0%BD%D0%B0_%D0%A7%D0%BE%D0%BC%D1%81%D0%BA%D0%B8> – йерархия на чомски
* **А. Антонов, Вл. Николов, Ив. Пенев** – учебно помагало по предмет „Дискретни структури“
* **А. Антонов, Вл. Николов, Ив. Пенев** – ръководство за семинарни упражнения по предмет „Дискретни структури“