МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»   
  
  
Институт радиоэлектроники и информационных технологий

Кафедра «Прикладная математика»

Лабораторная работа №2

Выполнил:

Студент группы 23-ПМ-2

Корноухов Илья Владимирович

Проверил:

Санников Николай Александрович

Нижний Новгород

2024

**Содержание**

[**Содержание** 2](#_Toc74168366)

[**Постановка задачи** 3](#_Toc74168367)

[**Теоретический материал** 4](#_Toc74168368)

[**Структура программы** 5](#_Toc74168368)

[**Пример работы программы** 6](#_Toc74168368)

[**Заключение** 8](#_Toc74168368)

[**Приложения** 9](#_Toc74168368)

**Постановка задачи**

Написать программу, выполняющую:

* Считывание КС грамматики из файла и вывод ее в консоль
* Проверку на принадлежность к классу КС-грамматик
* реализация эквивалентных преобразований грамматики, направленных на удаление бесполезных символов, недостижимых символов, ε-правил, цепных правил, левой факторизации правил, прямой левой рекурсии.

Разработать примеры контрольных входных данных

**Теоретический материал**

КС-грамматика называется приведенной, если она не имеет циклов, ε-правил и бесполезных символов.

Бесполезными символами грамматики называют нетерминалы, не порождающие терминальных строк, недостижимые нетерминалы, порождающие терминальные строки, т.е. множество символов

**Структура программы**

Программа написана на языке программирования Kotlin с использованием системы сборки Gradle.

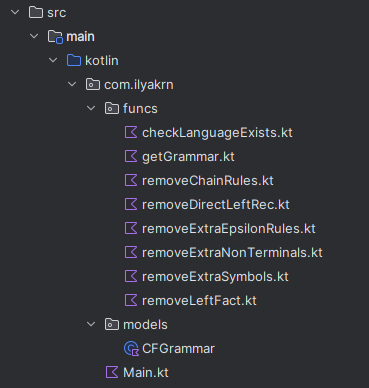


Рис. 1 - файлы программы

В приложениях 1-10 содержится исходный код программы.

В пакете funcs находятся файлы, содержащие функции для чтения грамматики, проверки приналежности к классу КС грамматик, эквивалентных преобразований

В пакете models находится файл с моделью КС грамматики.

В файле Main.kt находится функция main для запуска программы.

**Пример работы программы**

На рисунках 2-3 представлена работа программы: чтение КС грамматики и вывод КС грамматики, полученной эквивалентными преобразованиями .

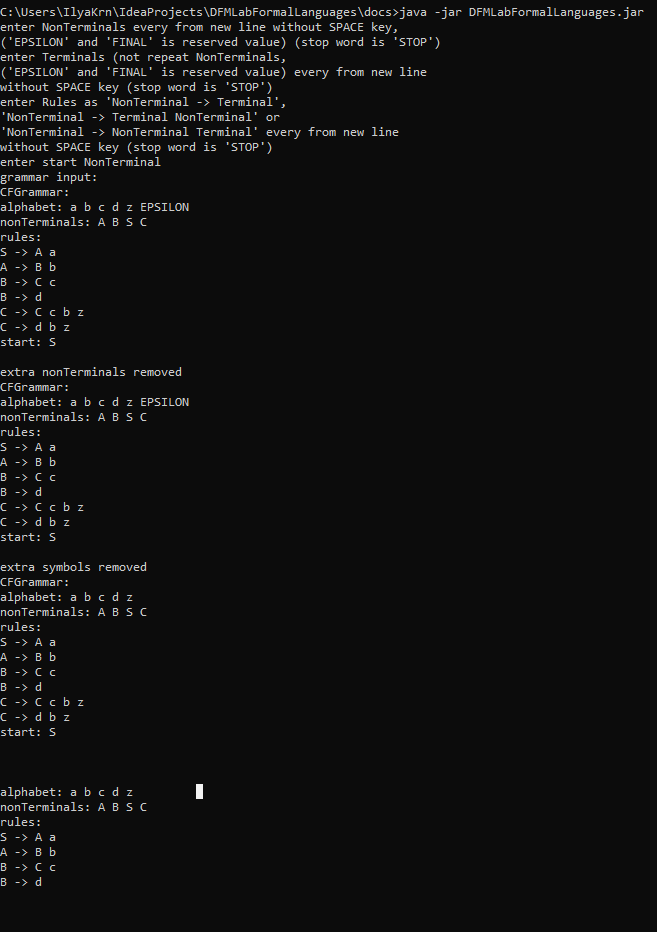


Рис. 2 - пример работы программы

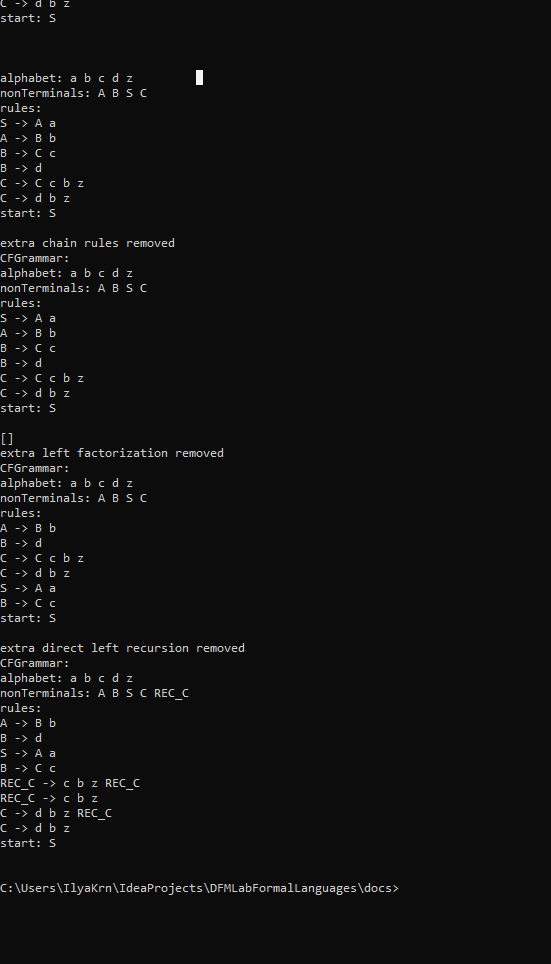


Рис. 3 - пример работы программы

**Заключение**

В ходе выполнения лабораторной работы был изучен необходимый материал и написана программа, считывающая КС грамматику и выполняющая эквивелентные преобразования КС грамматики

**Приложения**

Приложение 1 - файл checkLanguageExists.kt:

package com.ilyakrn.funcs  
  
import com.ilyakrn.models.CFGrammar  
  
fun checkLanguageExists(grammar: CFGrammar): Boolean {  
  
 val N = HashSet<String>()  
 val NNew = HashSet<String>()  
 var isStart = true  
 while (N != NNew || isStart) {  
 N.addAll(NNew)  
 isStart = false  
 for (rule in grammar.rules) {  
 var fl = false  
 rule.second.forEach {if (!N.contains(it) && !grammar.alphabet.contains(it)) {fl = true}}  
 if (fl)  
 continue  
 NNew.add(rule.first)  
  
 }  
 }  
  
 return N.contains(grammar.start)  
}

Приложение 2 - файл getGrammar.kt:

package com.ilyakrn.funcs  
  
import com.ilyakrn.models.CFGrammar  
import java.io.InputStream  
import java.io.PrintStream  
import java.util.\*  
import kotlin.collections.ArrayList  
import kotlin.collections.HashMap  
import kotlin.collections.HashSet  
  
  
fun getGrammar(stream: InputStream, out: PrintStream?): CFGrammar {  
 // Scanner of input stream  
 val scanner = Scanner(stream)  
 // initial values of grammar  
 val terminals = HashSet<String>()  
 val nonTerminals = HashSet<String>()  
 val rules = ArrayList<Pair<String, ArrayList<String>>>()  
 var start = ""  
 var linearly = 0 // 0 none; 1 right; -1 left;  
  
  
 // reading NonTerminals  
 out?.println("enter NonTerminals every from new line without SPACE key,\n('EPSILON' and 'FINAL' is reserved value) (stop word is 'STOP')")  
 var inputLine = scanner.nextLine()  
 while (inputLine != "STOP") {  
 if (!inputLine.contains(" ") && !nonTerminals.contains(inputLine) && inputLine != "EPSILON" && inputLine != "FINAL")  
 nonTerminals.add(inputLine)  
 else  
 out?.println("wrong NonTerminal $inputLine skipped, try again")  
 inputLine = scanner.nextLine()  
 }  
 // checking empty input  
 if (nonTerminals.size == 0)  
 throw RuntimeException("NonTerminals not found")  
  
  
 // reading Terminals  
 out?.println("enter Terminals (not repeat NonTerminals,\n('EPSILON' and 'FINAL' is reserved value) every from new line\nwithout SPACE key (stop word is 'STOP')")  
 inputLine = scanner.nextLine()  
 while (inputLine != "STOP") {  
 if (!inputLine.contains(" ") && !nonTerminals.contains(inputLine) && !terminals.contains(inputLine) && inputLine != "EPSILON" && inputLine != "FINAL")  
 terminals.add(inputLine)  
 else  
 out?.println("wrong Terminal $inputLine skipped, try again")  
 inputLine = scanner.nextLine()  
 }  
 // checking empty input  
 if (terminals.size == 0)  
 throw RuntimeException("NonTerminals not found")  
 // empty word  
 terminals.add("EPSILON")  
  
  
 // reading Rules  
 out?.println("enter Rules as 'NonTerminal -> Terminal',\n'NonTerminal -> Terminal NonTerminal' or\n'NonTerminal -> NonTerminal Terminal' every from new line\nwithout SPACE key (stop word is 'STOP')")  
 inputLine = scanner.nextLine()  
 while (inputLine != "STOP") {  
 val lineSep = inputLine.split(" ")  
 if (lineSep.size < 3){  
 out?.println("wrong Rule $inputLine skipped, try again")  
 }  
 if (!nonTerminals.contains(lineSep[0])){  
 out?.println("wrong Rule $inputLine skipped, try again")  
 }  
 var fl = false  
 for (i in lineSep.indices){  
 if(i > 1 && !nonTerminals.contains(lineSep[i]) && !terminals.contains(lineSep[i])){  
 out?.println("wrong Rule $inputLine skipped, try again")  
 fl = true  
 break  
 }  
 }  
 if (fl)  
 continue  
 val r = ArrayList<String>()  
 for (i in lineSep.indices){  
 if(i > 1 && (!nonTerminals.contains(lineSep[i]) || !terminals.contains(lineSep[i]))){  
 r.add(lineSep[i])  
 }  
 }  
 rules.add(Pair(lineSep[0], r))  
 inputLine = scanner.nextLine()  
 }  
 // checking empty input  
 if (rules.size == 0)  
 throw RuntimeException("Rules not found")  
  
  
 // reading starting NonTerminal  
 out?.println("enter start NonTerminal")  
 inputLine = scanner.nextLine()  
 while (true) {  
 if (nonTerminals.contains(inputLine))  
 start = inputLine  
 if (start != "")  
 break  
 out?.println("wrong start NonTerminal $inputLine skipped, try again")  
 if (scanner.hasNext())  
 inputLine = scanner.nextLine()  
 }  
  
 if (nonTerminals.size == 0 || terminals.size == 0 || rules.size == 0)  
 throw RuntimeException("Input is empty")  
  
 return CFGrammar(terminals, nonTerminals, rules, start)  
}

Приложение 3 - файл removeChainRules.kt:

package com.ilyakrn.funcs  
  
import com.ilyakrn.models.CFGrammar  
  
fun removeChainRules(grammar: CFGrammar): CFGrammar {  
  
 val outNTs = HashMap<String, HashSet<String>>()  
  
 for (a in grammar.nonTerminals){  
  
  
  
 val N = HashSet<String>()  
 val NNew = HashSet<String>()  
 N.add(a)  
 NNew.add(a)  
 var isStart = true  
 while (N != NNew || isStart) {  
 N.addAll(NNew)  
 isStart = false  
 for (b in N){  
 for (rule in grammar.rules) {  
 if (rule.first == b){  
 if (rule.second.size == 1 && grammar.nonTerminals.contains(rule.second[0]))  
 NNew.add(rule.second[0])  
 }  
 }  
 }  
 }  
 outNTs[a] = N  
 }  
  
 val newR = ArrayList<Pair<String, ArrayList<String>>>()  
 grammar.rules.forEach {  
 if (!(it.second.size == 1 && grammar.nonTerminals.contains(it.second[0]))) {  
 for(n in outNTs){  
 if (n.value.contains(it.first)){  
 newR.add(Pair(n.key, it.second))  
 }  
 }  
 }  
 }  
  
 return CFGrammar(grammar.alphabet, grammar.nonTerminals, newR, grammar.start)  
  
}

Приложение 4 - файл removeDirectLeftRec.kt:

package com.ilyakrn.funcs  
  
import com.ilyakrn.models.CFGrammar  
  
fun removeDirectLeftRec(grammar: CFGrammar): CFGrammar {  
 val newR = ArrayList<Pair<String, ArrayList<String>>>(grammar.rules)  
 val newNT = HashSet<String>(grammar.nonTerminals)  
  
 grammar.nonTerminals.forEach { x ->  
 val xRecRules = ArrayList<Pair<String, ArrayList<String>>>()  
 val xNormRules = ArrayList<Pair<String, ArrayList<String>>>()  
 grammar.rules.filter { it.first == x }.forEach{ rule ->  
 if (rule.second[0] == x) {  
 xRecRules.add(rule)  
 }  
 else{  
 xNormRules.add(rule)  
 }  
 }  
 if (xRecRules.isNotEmpty()){  
 val y = "REC\_" + x  
 newR.removeAll(xRecRules)  
 newR.removeAll(xNormRules)  
 newNT.add(y)  
 xRecRules.forEach { recRule ->  
 val newRule1 = Pair(y, arrayListOf<String>())  
 val newRule2 = Pair(y, arrayListOf<String>())  
 newRule1.second.addAll(recRule.second.subList(1, recRule.second.size))  
 newRule2.second.addAll(recRule.second.subList(1, recRule.second.size))  
 newRule1.second.add(y)  
 newR.add(newRule1)  
 newR.add(newRule2)  
 }  
 xNormRules.forEach { normRule ->  
 val newRule1 = Pair(x, ArrayList(normRule.second))  
 val newRule2 = Pair(x, ArrayList(normRule.second))  
 newRule1.second.add(y)  
 newR.add(newRule1)  
 newR.add(newRule2)  
 }  
  
  
 }  
  
 }  
  
  
 return CFGrammar(grammar.alphabet, newNT, newR, grammar.start)  
  
}

Приложение 5 - файл removeExtraEpsilonRules.kt:

package com.ilyakrn.funcs  
  
import com.ilyakrn.models.CFGrammar  
  
fun removeExtraEpsilonRules(grammar: CFGrammar): CFGrammar {  
 val N = HashSet<String>()  
 val NNew = HashSet<String>()  
 grammar.rules.forEach {  
 if (it.second.size == 1 && it.second.contains("EPSILON")) {  
 NNew.add(it.first)  
 N.add(it.first)  
 }  
 }  
 var isStart = true  
  
 while (N != NNew || isStart) {  
 N.addAll(NNew)  
 isStart = false  
 for (rule in grammar.rules) {  
 var fl = false  
 rule.second.forEach {  
 if (!N.contains(it)) {  
 fl = true  
 }  
 }  
 if (fl)  
 continue  
 NNew.add(rule.first)  
 }  
 }  
  
  
 val newR = ArrayList<Pair<String, ArrayList<String>>>()  
 newR.addAll(grammar.rules)  
 grammar.rules.forEach {  
 if (it.second.size == 1 && it.second.contains("EPSILON")) {  
 newR.remove(it)  
 }  
 }  
 var additRules = ArrayList<Pair<String, ArrayList<String>>>()  
  
 for (rule in newR) {  
 val NIndexs = HashSet<Int>()  
 for (i in rule.second.indices){  
 if (N.contains(rule.second[i]))  
 NIndexs.add(i)  
 }  
 if(NIndexs.size == 0)  
 continue  
 val indexCombs = HashSet<HashSet<Int>>()  
  
 for (i in 1..NIndexs.size){  
 indexCombs.addAll(getCombs(hashSetOf(hashSetOf()), NIndexs, i))  
 }  
  
 for (i in indexCombs) {  
 var nr = Pair(rule.first, ArrayList(rule.second))  
 i.sortedBy { -it }.forEach {  
 nr.second.removeAt(it)  
 }  
 if (nr.second.size != 0 && !(nr.second.size == 1 && nr.second.contains("EPSILON"))) {  
 additRules.add(nr)  
 }  
 }  
 }  
 newR.addAll(additRules)  
  
  
  
  
 var newNT = grammar.nonTerminals  
 var newS = "NEW\_" + grammar.start  
 if (N.contains(grammar.start)){  
 newR.add(Pair(newS, arrayListOf(grammar.start)))  
 newR.add(Pair(newS, arrayListOf("EPSILON")))  
 newNT.add(newS)  
 }  
 else {  
 newS = grammar.start  
 }  
  
 return CFGrammar(grammar.alphabet, newNT, newR, newS)  
  
}  
  
fun getCombs(comb: HashSet<HashSet<Int>>, vals: HashSet<Int>, deep: Int): HashSet<HashSet<Int>> {  
 for (c in comb){  
 if (c.size == deep){  
 return comb  
 }  
 }  
 val next = HashSet<HashSet<Int>>()  
 for (c in comb){  
 for (i in vals){  
 val temp = HashSet<Int>()  
 temp.addAll(c)  
 temp.add(i)  
 next.add(temp)  
 }  
 }  
 return getCombs(next, vals, deep)  
}

Приложение 6 - файл removeExtraNonTerminals.kt:

package com.ilyakrn.funcs  
  
import com.ilyakrn.models.CFGrammar  
  
fun removeExtraNonTerminals(grammar: CFGrammar): CFGrammar {  
 val N = HashSet<String>()  
 val NNew = HashSet<String>()  
 var isStart = true  
 while (N != NNew || isStart) {  
 N.addAll(NNew)  
 isStart = false  
 for (rule in grammar.rules) {  
 var fl = false  
 rule.second.forEach {if (!N.contains(it) && !grammar.alphabet.contains(it)) {fl = true}}  
 if (fl)  
 continue  
 NNew.add(rule.first)  
  
 }  
 }  
  
 val newR = ArrayList<Pair<String, ArrayList<String>>>()  
 val newNT = HashSet<String>()  
 grammar.nonTerminals.forEach {  
 if (N.contains(it)) {  
 newNT.add(it)  
 }  
 }  
 grammar.rules.forEach {  
 if (newNT.contains(it.first)) {  
 var fl = true  
 it.second.forEach { it1 ->  
 if (!N.contains(it1) && !grammar.alphabet.contains(it1)) {  
 fl = false  
 }  
 }  
 if (fl){  
 newR.add(it)  
 }  
 }  
 }  
  
 return CFGrammar(grammar.alphabet, newNT, newR, grammar.start)  
  
}

Приложение 7 - файл removeExtraSymbols.kt:

package com.ilyakrn.funcs  
  
import com.ilyakrn.models.CFGrammar  
  
fun removeExtraSymbols(grammar: CFGrammar): CFGrammar {  
 val W = HashSet<String>()  
 W.add(grammar.start)  
  
 val WNew = HashSet<String>()  
 WNew.add(grammar.start)  
 var isStart = true  
  
 while (W != WNew || isStart) {  
 W.addAll(WNew)  
 isStart = false  
 for (rule in grammar.rules) {  
 if (W.contains(rule.first)){  
 WNew.addAll(rule.second)  
 }  
 }  
 }  
  
 val newR = ArrayList<Pair<String, ArrayList<String>>>()  
 val newT = HashSet<String>()  
 val newNT = HashSet<String>()  
 W.forEach {  
 if (grammar.alphabet.contains(it))  
 newT.add(it)  
 else  
 newNT.add(it)  
 }  
 grammar.rules.forEach {  
 if (newNT.contains(it.first)) {  
 var fl = true  
 it.second.forEach { it1 ->  
 if (!newNT.contains(it1) && !newT.contains(it1)) {  
 fl = false  
 }  
 }  
 if (fl){  
 newR.add(it)  
 }  
 }  
 }  
  
 return CFGrammar(newT, newNT, newR, grammar.start)  
  
}

Приложение 8 - файл removeLeftFact.kt:

package com.ilyakrn.funcs  
  
import com.ilyakrn.models.CFGrammar  
  
fun removeLeftFact(grammar: CFGrammar): CFGrammar {  
 val newR = ArrayList<Pair<String, ArrayList<String>>>()  
 val newNT = HashSet<String>()  
 val allPrefixes = HashSet<ArrayList<String>>()  
 grammar.nonTerminals.forEach { x ->  
 val xRights = HashSet<ArrayList<String>>()  
 grammar.rules.forEach {  
 if (it.first == x){  
 xRights.add(it.second)  
 }  
 }  
 val prefixes = HashSet<ArrayList<String>>()  
  
 var deep = 0  
 xRights.forEach {  
 if (deep < it.size) deep = it.size  
 }  
 for (i in 0 until deep) {  
 prefixes.addAll(getPrefixes(hashSetOf(arrayListOf()), xRights, i))  
 }  
 if (prefixes.contains(emptyList()))  
 prefixes.remove(emptyList())  
 val newPrefixes = HashSet<ArrayList<String>>(prefixes)  
 prefixes.forEach { prefix1 ->  
 prefixes.forEach { prefix2 ->  
 var count = 0  
 xRights.forEach {  
 if (prefix2.size < it.size) count++  
 }  
 if (count < 2) {  
 newPrefixes.remove(prefix2)  
 }  
 else {  
 if (prefix1.size < prefix2.size) {  
 var fl = true  
 for (i in prefix1.indices) {  
 if (prefix1[i] != prefix2[i]) {  
 fl = false  
 }  
 }  
 if (fl || prefix1.isEmpty())  
 newPrefixes.remove(prefix1)  
 }  
 }  
 }  
 }  
 prefixes.clear()  
 prefixes.addAll(newPrefixes)  
 allPrefixes.addAll(prefixes)  
  
 prefixes.forEach { prefix ->  
 newNT.add("PREFIX\_" + prefix.toString())  
 for (rule in xRights.filter {  
 for (i in prefix.indices) {  
 if (prefix[i] != it[i]) {  
 return@filter false  
 }  
 }  
 return@filter true  
 }) {  
 newR.add(Pair("PREFIX\_" + prefix.toString(), rule.subList(prefix.size, rule.size).toCollection(ArrayList())))  
 }  
 newR.add(Pair(x, arrayListOf("PREFIX\_" + prefix.toString())))  
 }  
 }  
 println(newR)  
 grammar.rules.forEach { rule ->  
 var fl = true  
 allPrefixes.forEach { prefix ->  
 var fl1= true  
 for (i in prefix.indices) {  
 if (prefix.size < rule.second.size && prefix[i] != rule.second[i]) {  
 fl = false  
 }  
 }  
 if (fl1) {  
 fl1 = false  
 }  
 }  
 if (fl)  
 newR.add(rule)  
 }  
 newNT.addAll(grammar.nonTerminals)  
  
 val temp = HashSet<Pair<String, ArrayList<String>>>(newR)  
 newR.clear()  
 temp.forEach {  
 newR.add(it)  
 }  
  
 return CFGrammar(grammar.alphabet, newNT, newR, grammar.start)  
  
}  
  
fun getPrefixes(prefixes: HashSet<ArrayList<String>>, rights: HashSet<ArrayList<String>>, deep: Int): HashSet<ArrayList<String>>{  
 prefixes.forEach { prefix ->  
 if (deep == prefix.size) return prefixes  
 }  
 if (prefixes.isEmpty())  
 return hashSetOf()  
 val addedPrefixes = ArrayList<ArrayList<String>>()  
 prefixes.forEach { prefix ->  
 rights.forEach { right ->  
 if(prefix.size < right.size){  
 var fl = true  
 for (i in prefix.indices){  
 if (prefix[i] != right[i]){  
 fl = false  
 break  
 }  
 }  
 if (fl){  
 val newPrefix = ArrayList<String>(prefix)  
 newPrefix.add(right[prefix.size])  
 addedPrefixes.add(newPrefix)  
 }  
 }  
 }  
 }  
  
 val temp = HashSet<ArrayList<String>>()  
 val nextPrefixes = ArrayList<ArrayList<String>>()  
 addedPrefixes.forEach { next ->  
 if (temp.contains(next)){  
 nextPrefixes.add(next)  
 }  
 temp.add(next)  
 }  
 return getPrefixes(nextPrefixes.toHashSet(), rights, deep)  
  
}  
  
fun main() {  
 println(getPrefixes(hashSetOf(arrayListOf()), hashSetOf(  
 arrayListOf("d", "b", "z", "Z" ),  
 arrayListOf("d", "b", "z")  
 ), 3))  
}

Приложение 9 - файл CFGrammar.kt:

package com.ilyakrn.models;  
  
data class CFGrammar (  
 var alphabet: HashSet<String>,  
 var nonTerminals: HashSet<String>,  
 var rules: ArrayList<Pair<String, ArrayList<String>>>,  
 var start: String  
) {  
 override fun toString(): String {  
 var alphabetS = "alphabet: "  
 alphabet.forEach {  
 alphabetS += "$it "  
 }  
 var nonTerminalsS = "nonTerminals: "  
 nonTerminals.forEach {  
 nonTerminalsS += "$it "  
 }  
 var rulesS = "rules:\n"  
 rules.forEach {  
 var r = ""  
 it.second.forEach { it1 ->  
 r += "$it1 "  
 }  
 rulesS += "${it.first} -> $r\n"  
 }  
 return "CFGrammar:\n$alphabetS\n$nonTerminalsS\n${rulesS}start: $start\n"  
 }  
}

Приложение 10 - файл Main.kt:

package com.ilyakrn  
  
import com.ilyakrn.funcs.\*  
import java.io.File  
import java.io.FileInputStream  
import java.util.\*  
  
fun main() {  
  
 var g = getGrammar(FileInputStream("C:\\Users\\IlyaKrn\\IdeaProjects\\DFMLabFormalLanguages\\src\\main\\resources\\test.txt"), System.out)  
 println("grammar input: ")  
 println(g)  
 if(checkLanguageExists(g)){  
 g = removeExtraNonTerminals(g)  
 println("extra nonTerminals removed")  
 println(g)  
 g = removeExtraSymbols(g)  
 println("extra symbols removed")  
 println(g)  
 g = removeExtraEpsilonRules(g)  
 println("extra epsilon rules removed")  
 println(g)  
 g = removeChainRules(g)  
 println("extra chain rules removed")  
 println(g)  
 g = removeLeftFact(g)  
 println("extra left factorization removed")  
 println(g)  
 g = removeDirectLeftRec(g)  
 println("extra direct left recursion removed")  
 println(g)  
 return  
 }  
 println("language not exists")  
  
  
}