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** 7](#_Toc74168368)

[**Приложения** 7](#_Toc74168368)

**Постановка задачи**

Написать программу, выполняющую:

* Считывание ДКА из файла и вывод его в консоль
* Построение на его основе минимального автомата и вывод его в консоль

Разработать примеры контрольных входных данных

**Теоретический материал**

Конечный автомат может содержать лишние состояния двух видов:

недостижимые и эквивалентные состояния.

Состояния называются n-эквивалентными, если находясь в одном из этих состояний и получив на вход любую цепочку символов, длина которой не превышает n, автомат может прейти в одно и тоже множество конечных состояний.

Состояние называется недостижимым, если в него нельзя попасть из начального состояния.

Конечный автомат, не содержащий недостижимых и эквивалентных состояний, называется минимальным.

**Структура программы**

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

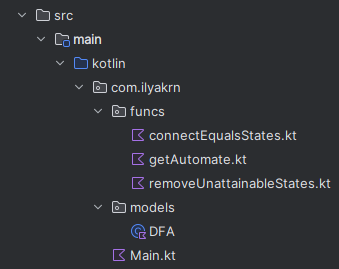


Рисунок 1 - файлы программы

На рисунке 1 представлена структура программы.

В приложениях 1-5 содержится исходный код программы.

В пакете funcs находятся файлы, содержащие функции для чтения автомата, исключения недостижимых и эквивалентных состояний автомата.

В пакете models находится файл с моделью ДКА.

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

**Пример работы программы**

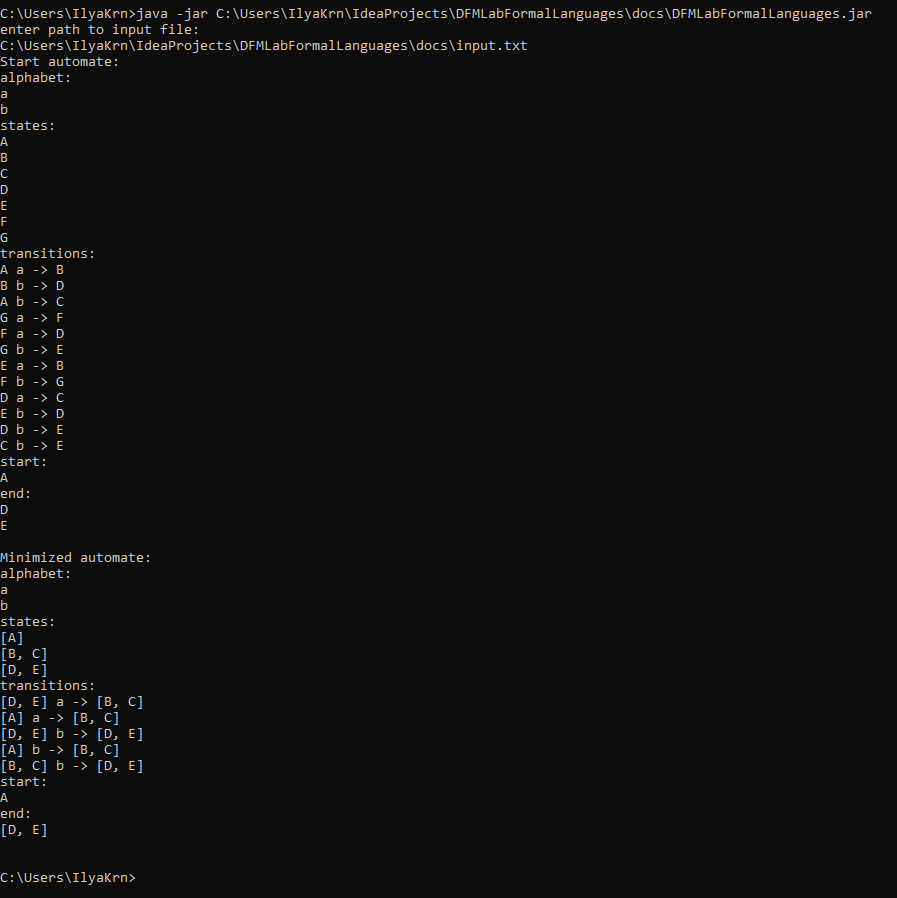
****

Рисунок 2 - пример работы программы

На рисунке 2 представлена работа программы: чтение автомата из файла, вывод исходного и минимального автоматов.

**Заключение**

В ходе выполнения лабораторной работы был изучен необходимый материал и написана программа, считывающая грамматику и строящая на ее основе детерминированный конечный автомат.

**Приложения**

Приложение 1 - файл DFA.kt:

package com.ilyakrn.models  
  
data class DFA(  
 var symb: HashSet<String>,  
 var q: HashSet<String>,  
 var transitions: HashMap<Pair<String, String>, String>,  
 var start: String,  
 var ends: HashSet<String>  
  
  
) {  
 override fun toString(): String {  
 var transitionsString = "{"  
 transitions.forEach {  
 transitionsString += "${it.key} -> ${it.value},\n"  
 }  
 transitionsString += "\n}\n"  
 var result = ""  
 result += "alphabet:\n"  
 symb.forEach {  
 result += "${it}\n"  
 }  
 result += "states:\n"  
 q.forEach {  
 result += "${it}\n"  
 }  
 result += "transitions:\n"  
 transitions.forEach{  
 result += "${it.key.first} ${it.key.second} -> ${it.value}\n"  
 }  
 result += "start:\n${start}\n"  
 result += "end:\n"  
 ends.forEach {  
 result += "$it\n"  
 }  
 return result  
 }  
}

Приложение 2 - файл connectEqualsStates.kt:

package com.ilyakrn.funcs  
  
import com.ilyakrn.models.DFA  
  
fun connectEqualsStates(a: DFA) : DFA {  
  
 var partitions = ArrayList<HashSet<String>>()  
 partitions.add(a.ends)  
 partitions.add(a.q.filter { !a.ends.contains(it) }.toHashSet())  
  
 partitions = recursivePartitions(partitions, a.symb, a.transitions)  
  
 val newTransitions = HashMap<Pair<String, String>, String>()  
  
 for (partitionKey in partitions) {  
 for (transition in a.transitions) {  
 if (partitionKey.contains(transition.key.first)){  
 var nextState = HashSet<String>()  
 for (partitionNext in partitions){  
 if (partitionNext.contains(transition.value)){  
 nextState = partitionNext  
 }  
 }  
 if (nextState.isNotEmpty()) {  
 newTransitions[Pair(partitionKey.toString(), transition.key.second)] = nextState.toString()  
 }  
 }  
 }  
 }  
 val newEnds = HashSet<String>()  
 val newStates = HashSet<String>()  
 for (partition in partitions){  
 for (s in partition){  
 if (a.ends.contains(s)){  
 newEnds.add(partition.toString())  
 }  
 }  
 newStates.add(partition.toString())  
 }  
  
 return DFA(a.symb, newStates, newTransitions, a.start, newEnds)  
}  
  
private fun recursivePartitions(p: ArrayList<HashSet<String>>, symbols: HashSet<String>, transitions: HashMap<Pair<String, String>, String>): ArrayList<HashSet<String>>{  
 var pNew = ArrayList<HashSet<String>>()  
 for (eqClass in p) {  
 val newClass = HashSet<String>()  
 for (s in symbols) {  
 var newStates = HashSet<String>()  
 for (state in eqClass) {  
 transitions[Pair(state, s)]?.let {  
 if (newStates.isEmpty()){  
 for (i in p){  
 if (i.contains(it))  
 newStates = i  
 }  
 }  
 else{  
 if (!newStates.contains(it)){  
 newClass.add(state)  
 }  
 }  
 }  
 }  
 }  
 val oldClass = HashSet<String>()  
 if (newClass.isNotEmpty()){  
 oldClass.addAll(eqClass)  
 oldClass.removeAll(newClass)  
 pNew.add(newClass)  
 pNew.add(oldClass)  
 }  
 else{  
 pNew.add(eqClass)  
 }  
 }  
 if (p == pNew)  
 return pNew  
 else  
 return recursivePartitions(pNew, symbols, transitions)  
  
  
}

Приложение 3 - файл getAutomate.kt:

package com.ilyakrn.funcs  
  
import com.ilyakrn.models.DFA  
import java.io.InputStream  
import java.io.PrintStream  
import java.util.Scanner  
  
fun getAutomate(input: InputStream, output: PrintStream?): DFA {  
  
 val scanner = Scanner(input)  
 val alphabet = HashSet<String>()  
 val states = HashSet<String>()  
 val transitions = HashMap<Pair<String, String>, String>()  
 var start = ""  
 var ends = HashSet<String>()  
  
  
 output?.println("enter alphabet. STOP to end, 'EPSILON' reserved")  
 var inputLine = scanner.nextLine()  
 while (inputLine != "STOP") {  
 if (!inputLine.contains(" ") && inputLine.isNotEmpty() && !alphabet.contains(inputLine) && inputLine != "EPSILON")  
 alphabet.add(inputLine)  
 else  
 output?.println("wrong input $inputLine skipped, try again")  
 inputLine = scanner.nextLine()  
 }  
  
 output?.println("enter states. STOP to end")  
 inputLine = scanner.nextLine()  
 while (inputLine != "STOP") {  
 if (!inputLine.contains(" ") && inputLine.isNotEmpty() && !states.contains(inputLine))  
 states.add(inputLine)  
 else  
 output?.println("wrong input $inputLine skipped, try again")  
 inputLine = scanner.nextLine()  
 }  
  
 output?.println("enter transitions (State Symbol -> State). STOP to end")  
 inputLine = scanner.nextLine()  
 while (inputLine != "STOP") {  
 val tr = inputLine.split(" ")  
 if (tr.size == 4 && states.contains(tr[0]) && states.contains(tr[3]) && alphabet.contains(tr[1])) {  
 transitions[Pair(tr[0], tr[1])] = tr[3]  
 }  
 else  
 output?.println("wrong input $inputLine skipped, try again")  
 inputLine = scanner.nextLine()  
 }  
  
 output?.println("enter start state")  
 inputLine = scanner.nextLine()  
 while (true) {  
 if (states.contains(inputLine)) {  
 start = inputLine  
 break  
 }  
 else  
 output?.println("wrong input $inputLine skipped, try again")  
 inputLine = scanner.nextLine()  
 }  
  
 output?.println("enter end states. STOP to end")  
 inputLine = scanner.nextLine()  
 while (inputLine != "STOP") {  
 if (states.contains(inputLine))  
 ends.add(inputLine)  
 else  
 output?.println("wrong input $inputLine skipped, try again")  
 inputLine = scanner.nextLine()  
 }  
  
 return DFA(alphabet, states, transitions, start, ends)  
  
}

Приложение 4 - файл removeUnattainableStates.kt:

package com.ilyakrn.funcs  
  
import com.ilyakrn.models.DFA  
  
fun removeUnattainableStates(a: DFA): DFA {  
  
 val attainable = HashSet<String>()  
 var newAttainable = HashSet<String>()  
 newAttainable.add(a.start)  
  
 while (newAttainable != attainable){  
 attainable.addAll(newAttainable)  
 for (state in attainable){  
 for (symb in a.symb){  
 if (a.transitions[Pair(state, symb)] != null)  
 newAttainable.add(a.transitions[Pair(state, symb)]!!)  
 }  
 }  
 }  
 attainable.addAll(newAttainable)  
  
 val newTransitions = HashMap<Pair<String, String>, String>();  
 for (transition in a.transitions){  
 if (!attainable.contains(transition.key.first) || !attainable.contains(transition.value)){  
 continue  
 }  
 newTransitions[Pair(transition.key.first, transition.key.second)] = transition.value  
 }  
 var newEnds = HashSet<String>()  
 for (e in a.ends){  
 if (!attainable.contains(e))  
 continue  
 newEnds.add(e)  
 }  
 return DFA(a.symb, attainable, newTransitions, a.start, newEnds)  
  
}

Приложение 5 - файл Main.kt:

package com.ilyakrn  
  
import com.ilyakrn.funcs.connectEqualsStates  
import com.ilyakrn.funcs.getAutomate  
import com.ilyakrn.funcs.removeUnattainableStates  
import java.io.File  
import java.io.FileInputStream  
import java.util.Scanner  
  
fun main() {  
  
 println("enter path to input file:")  
 var a = getAutomate(FileInputStream(File(Scanner(System.`in`).nextLine())), null)  
 println("Start automate:\n$a")  
 a = removeUnattainableStates(a)  
 a = connectEqualsStates(a)  
 println("Minimized automate:\n$a")  
  
}