ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра вычислительных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по дисциплине

«Теория языков программирования и методы трансляции»

Выполнил:

студент гр. ИП-613

Плотников А.В.

Проверила:

Доцент кафедры ПМиК

Бах О.А.

Новосибирск, 2019

Содержание

1. Текст задания……………………………………………………………………………3
2. Описание алгоритма решения задачи………………………………………………….3
3. Описание основных блоков программы……………………………………………….4
4. Листинг кода программы………………………………………………………………..5

# 1. Текст задания

Преобразование конструкций, задающих язык. Написать программу, которая по заданной регулярной грамматике (грамматика может быть НЕ автоматного вида!, ЛЛ или ПЛ) построит эквивалентный ДКА (представление функции переходов в виде таблицы). Программа должна сгенерировать по исходной грамматике несколько цепочек в заданном диапазоне длин и проверить их допустимость построенным автоматом. Процессы построения цепочек и проверки их выводимости отображать на экране (по требованию). Предусмотреть возможность проверки цепочки, введённой пользователем.

# 2. Описание алгоритма решения задачи

1. Для начала, нужно научить программу представлять регулярную грамматику в автоматном виде, для дальнейшего построения по ней НКА. После построения НКА по алгоритму данному в 2.2.2, мы создаем ДКА эквивалентный НКА.

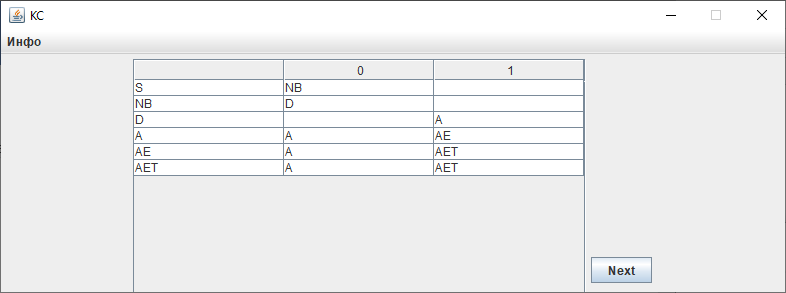


Рис. 1 Таблица с функциями перехода ДКА

1. Затем мы генерируем цепочки определенного диапазона и выводим их виде того как они были получены.После построения цепочки идет проверки её на автомате, и в случае верности цепочки она выводится.

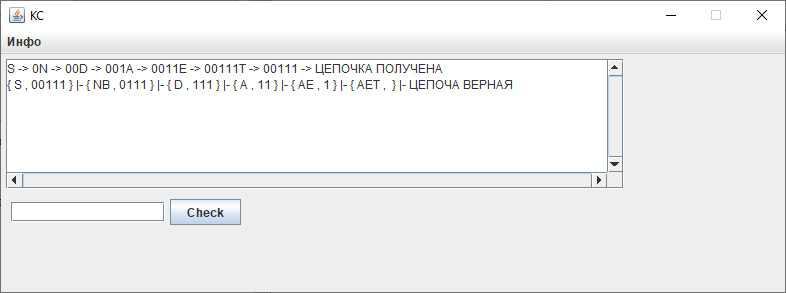


Рис . 2 Демонстрация цепочки и её проверки

1. Затем мы делаем так, чтобы пользователь тоже мог вводить свои цепочки на проверку данному ДКА автомату и в случае верного ответа выводи её разбор. В случае если цепочка не может быть получена на данном автомате то будет выведена причина ошибки.

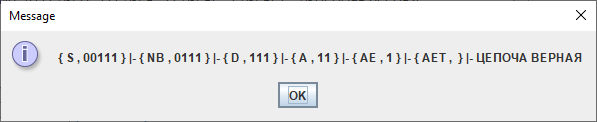


Рис.3 Верная цепочка введенная пользователем

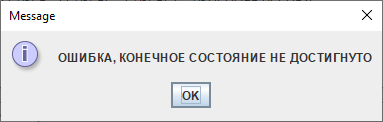


Рис. 4 Неверная цепочка

# 3. Описание основных блоков программы

Программа состоит из 1 классов.

**public class** Main Главный класс

**public static void** rules() Метод позволяющий ввести правила для дальнейшей их проверки

**public static boolean** check\_Rules(String pp) Проверка правильности введение правил

**public static boolean** chain\_Avto() Создание грамматики автоматного вида и проверки на то возможно ли это.

**public static void** NKA() Метод для создания НКА

**public static void** DKA() Метод для создание ДКА

**public static void** generation(HashMap <String, ArrayList<String>> map, ArrayList <String> list,HashMap <String, ArrayList<String>> ans, HashMap<Integer, HashMap<String, String>> chain) Генерация цепочек определенного диапазона

**public static** String DKA\_check(String str)Проверка цепочки на принадлежность ДКА

# 4. Листинг кода программы

\*В листинге кода опущены строки форм для диалога с пользователем и методы для работы со строкой

**public class** Main {  
  
 **private static** String *jt\_str\_term*, *jt\_str\_noterm*, *jt\_str\_begin*, *jt\_str\_numb*, *jt\_str\_nume*;  
 **private static int** *num\_begin*, *num\_end*;  
 **private static** String[] *simbol* = {**"N"**, **"D"**, **"E"**, **"F"**, **"G"**, **"H"**, **"J"**};  
 **private static** ArrayList<String> *str\_temp\_term* = **new** ArrayList<>();  
 **private static** ArrayList <String> *str\_temp\_noterm* = **new** ArrayList<>();  
 **private static** ArrayList <String> *str\_Temp\_noterm\_True* = **new** ArrayList<>(); *// верный список нетерминалов* **private static** ArrayList <String> *str\_Temp\_noterm\_DKA* = **new** ArrayList<>(); *// DKA нетерминалов* **private static** HashMap<String, ArrayList<String>> *rule* = **new** HashMap<>();  
 **private static** HashMap<String, ArrayList<String>> *rule\_True* = **new** HashMap<>(); *// верный список правил* **private static** HashMap<String, HashMap<String, String>> *NKA\_map* = **new** HashMap<>(); *//НКА* **private static** HashMap<String, HashMap<String, String>> *DKA\_map* = **new** HashMap<>(); *//ДКА* **private static** HashMap<String, HashMap<String, String>> *DKA\_map\_True* = **new** HashMap<>(); *//ДКА без лишних состояний* **private static** ArrayList<String> *end\_pos* = **new** ArrayList<>(); *//Множество конечных состояний* **private static** HashMap<Integer, HashMap<String, String>> *number\_chain* = **new** HashMap<>();  
 **private static** HashMap <String, ArrayList<String>> *answer* = **new** HashMap<>();  
 **private static** JFrame *myWindow*;  
 **private static** JPanel *jp\_main*;  
 **private static** JPanel *jp\_rule*;  
 **private static** JPanel *jp\_table*;  
 **private static** JPanel *jp\_chain*;  
 **private static int** *index* = 0;

**public static void** main(String[] args) {  
 *myWindow* = **new** JFrame(**"KC"**);  
 *myWindow*.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
 *str\_temp\_term*.add(**"#"**);  
 JMenuBar menubar = **new** JMenuBar();  
 *// создаем меню* menubar.add(*info*(*myWindow*));  
 *jp\_main* = **new** JPanel(**new** VerticalLayout());  
 *jp\_rule* = **new** JPanel(**new** VerticalLayout());  
 *jp\_table* = **new** JPanel();  
 *jp\_chain* = **new** JPanel(**new** VerticalLayout());  
 *//Панель для терминальных* JPanel panel1 = **new** JPanel();  
 JLabel lb\_term = **new** JLabel(**"Терминальные символы: "**);  
 JTextField jt\_term = **new** JTextField(15);  
 panel1.add(lb\_term);  
 panel1.add(jt\_term);  
 *jp\_main*.add(panel1);  
 *//Панель для не терминальных* panel1 = **new** JPanel();  
 JLabel lb\_noterm = **new** JLabel(**"Нетерминальные символы: "**);  
 JTextField jt\_noterm = **new** JTextField(15);  
 panel1.add(lb\_noterm);  
 panel1.add(jt\_noterm);  
 *jp\_main*.add(panel1);  
 *//Начальный символ* panel1 = **new** JPanel();  
 JLabel jl\_begin = **new** JLabel(**"Начальный символ: "**);  
 JTextField jt\_begin = **new** JTextField(15);  
 panel1.add(jl\_begin);  
 panel1.add(jt\_begin);  
 *jp\_main*.add(panel1);  
 *//Диапазон* panel1 = **new** JPanel();  
 JLabel jl\_diap\_beg = **new** JLabel(**"От: "**);  
 JTextField jt\_diap\_beg = **new** JTextField(2);  
 panel1.add(jl\_diap\_beg);  
 panel1.add(jt\_diap\_beg);  
 JLabel jl\_diap\_end = **new** JLabel(**"До: "**);  
 JTextField jt\_diap\_end = **new** JTextField(2);  
 panel1.add(jl\_diap\_end);  
 panel1.add(jt\_diap\_end);  
 *jp\_main*.add(panel1);  
 JButton check = **new** JButton(**"Next"**);  
 check.addActionListener(**new** ActionListener() {  
  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
  
 *jt\_str\_term* = jt\_term.getText();  
 *jt\_str\_noterm* = jt\_noterm.getText();  
 *jt\_str\_begin* = jt\_begin.getText();  
 *jt\_str\_numb* = jt\_diap\_beg.getText();  
 *jt\_str\_nume* = jt\_diap\_end.getText();  
 **for** (String std:*jt\_str\_term*.split(**" "**)) {  
 *str\_temp\_term*.add(std);  
 }  
 **for** (String std:*jt\_str\_noterm*.split(**" "**)) {  
 *str\_temp\_noterm*.add(std);  
 }  
 String regex = **"-?\\d+(\\.\\d+)?"**;  
 **if** (!(*jt\_str\_numb*.matches(regex)) || !(*jt\_str\_nume*.matches(regex))) {  
 *str\_temp\_noterm*.clear();  
 *str\_temp\_term*.clear();  
 *str\_temp\_term*.add(**"#"**);  
 JOptionPane.*showMessageDialog*(*myWindow*, **"ОШИБКА, ДИАПАЗОН ЗАДАЕТСЯ ТОЛЬКО ЧИСЛАМИ"**);  
 **return**;  
 } **else** {  
 *num\_begin* = Integer.*parseInt*(*jt\_str\_numb*);  
 *num\_end* = Integer.*parseInt*(*jt\_str\_nume*);  
 **if** (*num\_begin* == 0 || *num\_end* == 0 || *num\_end* - *num\_begin* <= 0) {  
 *str\_temp\_noterm*.clear();  
 *str\_temp\_term*.clear();  
 *str\_temp\_term*.add(**"#"**);  
 JOptionPane.*showMessageDialog*(*myWindow*, **"ОШИБКА, ДИАПАЗОН ЗАДАН НЕ ВЕРНО"**);  
 **return**;  
 }  
 }  
  
 **if** (*str\_temp\_noterm*.isEmpty()) {  
 *str\_temp\_noterm*.clear();  
 *str\_temp\_term*.clear();  
 *str\_temp\_term*.add(**"#"**);  
 JOptionPane.*showMessageDialog*(*myWindow*, **"ОШИБКА, ПУСТОЕ ПОЛЕ"**);  
 **return**;  
 }  
 **if** (*str\_temp\_term*.isEmpty()) {  
 *str\_temp\_noterm*.clear();  
 *str\_temp\_term*.clear();  
 *str\_temp\_term*.add(**"#"**);  
 JOptionPane.*showMessageDialog*(*myWindow*, **"ОШИБКА, ПУСТОЕ ПОЛЕ"**);  
 **return**;  
 }  
 *//Проверка на вшивость* **boolean** flag = **true**;  
 **for** (String std: *str\_temp\_noterm*) {  
 **if** (std.equals(*jt\_str\_begin*)) {  
 flag = **false**;  
 }  
 }  
 **if** (flag) {  
 *str\_temp\_noterm*.clear();  
 *str\_temp\_term*.clear();  
 *str\_temp\_term*.add(**"#"**);  
 JOptionPane.*showMessageDialog*(*myWindow*, **"ОШИБКА, НАЧАЛЬНЫЙ СИМВОЛ НЕ ПРИНАДЛЕЖИТ НЕТЕРМИНАЛЬНЫМ"**);  
 **return**;  
 }  
  
 flag = **false**;  
 **for** (String std: *str\_temp\_noterm*) {  
 **for** (String std1 : *str\_temp\_term*) {  
 **if** (std.equals(std1)) {  
 flag = **true**;  
 }  
 }  
 }  
  
 **if** (flag) {  
 *str\_temp\_noterm*.clear();  
 *str\_temp\_term*.clear();  
 *str\_temp\_term*.add(**"#"**);  
 JOptionPane.*showMessageDialog*(*myWindow*, **"ОШИБКА, ТЕРМИНАЛЬНЫЕ СИВМОЛЫ ПРИНАДЛЕЖАТ НЕТЕРМИНАЛЬНЫМ"**);  
 **return**;  
 }  
 *str\_Temp\_noterm\_True*.addAll(*str\_temp\_noterm*);  
  
 *jp\_main*.setVisible(**false**);  
 *rules*();  
  
 }  
 });  
 *myWindow*.add(*jp\_main*);  
 *jp\_main*.add(check);  
 *// добавляем панель меню в окно  
 myWindow*.setJMenuBar(menubar);  
 *myWindow*.setVisible(**true**);  
 *myWindow*.setSize(800, 300);  
 *myWindow*.setResizable(**false**);  
  
}

**public static void** rules() {  
 JLabel pt = **new** JLabel();  
 pt.setText(*str\_temp\_noterm*.get(*index*));  
 JTextField pp = **new** JTextField(14);  
 JButton ne = **new** JButton(**"Next"**);  
 ne.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **if** (*str\_temp\_noterm*.size() == *index* + 1) {  
 **if**(*check\_Rules*(pp.getText())) {  
 **if** (*chain\_Avto*()) { *// Проверка на ввод  
 jp\_rule*.setVisible(**false**);  
 *str\_Temp\_noterm\_DKA*.addAll(*str\_Temp\_noterm\_True*);  
 *NKA*();  
 *DKA*();  
  
 } **else** {  
 *rule*.clear();  
 *index* = 0;  
 *jp\_rule*.removeAll();  
 *jp\_rule*.updateUI();  
 *rules*();  
 }  
 } **else** {  
 *rule*.clear();  
 *index* = 0;  
 *jp\_rule*.removeAll();  
 *jp\_rule*.updateUI();  
 *rules*();  
 }  
 } **else** {  
 **if**(*check\_Rules*(pp.getText())) {  
 *index*++;  
 } **else** {  
 *rule*.clear();  
 *index* = 0;  
 }  
 *jp\_rule*.removeAll();  
 *jp\_rule*.updateUI();  
 *rules*();  
 }  
 }  
 });  
 *jp\_rule*.add(pt);  
 *jp\_rule*.add(pp);  
 *jp\_rule*.add(ne);  
 *myWindow*.add(*jp\_rule*);  
}  
  
**public static boolean** check\_Rules(String pp) {  
 **if** (pp.equals(**""**)) {  
 JOptionPane.*showMessageDialog*(*myWindow*, **"ПУСТАЯ СТРОКА"**);  
 **return false**;  
 }  
 ArrayList<String> templ = **new** ArrayList<>();  
 **for** (String std: pp.split(**" "**)) {  
 templ.add(std);  
 }  
  
 **for** (String std: templ) {  
 **for** (String s:*str\_temp\_term*) {  
 std = std.replaceAll(s, **""**);  
 }  
 **for** (String s:*str\_temp\_noterm*) {  
 std = std.replaceAll(s, **""**);  
 }  
 **if** (!(std.equals(**""**))) {  
 JOptionPane.*showMessageDialog*(*myWindow*, **"В ПРАВИЛЕ ЕСТЬ СИМВОЛЫ КОТОРЫХ НЕТУ В ГРАММАТИКЕ"**);  
 **return false**;  
 }  
 }  
 *rule*.put(*str\_temp\_noterm*.get(*index*), templ);  
 **return true**;  
}  
  
**public static boolean** chain\_Avto() {  
 **int** flag = 0;  
 **for** (String s: *str\_temp\_noterm*) {  
 **for** (String str : *rule*.get(s)) {  
 **if**(str.length() == 2) {  
 **if** (*check\_Simbol*(str.substring(0, 1), *str\_temp\_term*) && *check\_Simbol*(str.substring(1, 2), *str\_temp\_noterm*)) {  
 flag = 1;  
 **break**;  
 } **else if** (*check\_Simbol*(str.substring(0, 1), *str\_temp\_noterm*) && *check\_Simbol*(str.substring(1, 2), *str\_temp\_term*)) {  
 flag = 2;  
 **break**;  
 }  
 }  
 }  
 **if** (flag != 0) {  
 **break**;  
 }  
 }  
  
 ArrayList <String> temp = **new** ArrayList<>();  
 **if** (flag == 0 || flag == 1) {  
 **for** (String s: *str\_temp\_noterm*) {  
 **for** (String str : *rule*.get(s)) {  
 **if** (str.length() > 2) {  
 **if** (str.length() - *countChar*(str) == 1 && *check\_Simbol*(str.substring(str.length() - 1), *str\_temp\_noterm*) ) {  
 String s\_temp\_nt = *find\_Simbol*();  
 String s\_temp\_nt1 = **""**;  
 temp.add(str.substring(0 , 1) + s\_temp\_nt);  
 ArrayList <String> ss = **new** ArrayList<>();  
 **for** (**int** i = 1; i < *countChar*(str) - 1; i++ ) {  
 s\_temp\_nt1 = *find\_Simbol*();  
 ss.add(str.substring(i, i + 1) + s\_temp\_nt1);  
 *rule\_True*.put(s\_temp\_nt, ss);  
 s\_temp\_nt = s\_temp\_nt1;  
 ss = **new** ArrayList<>();  
 }  
 ss.add(str.substring(str.length() - 2));  
 *rule\_True*.put(s\_temp\_nt, ss);  
  
 } **else if** (str.length() - *countChar*(str) == 0) {  
 String s\_temp\_nt = *find\_Simbol*();  
 String s\_temp\_nt1 = **""**;  
 temp.add(str.substring(0 , 1) + s\_temp\_nt);  
 ArrayList <String> ss = **new** ArrayList<>();  
 **for** (**int** i = 1; i < *countChar*(str) - 1; i++ ) {  
 s\_temp\_nt1 = *find\_Simbol*();  
 ss.add(str.substring(i, i + 1) + s\_temp\_nt1);  
 *rule\_True*.put(s\_temp\_nt, ss);  
 s\_temp\_nt = s\_temp\_nt1;  
 ss = **new** ArrayList<>();  
 }  
 ss.add(str.substring(str.length() - 1) + **"T"**);  
 *rule\_True*.put(s\_temp\_nt, ss);  
 } **else** {  
 **return false**;  
 }  
 } **else if** (str.length() == 2 && *check\_Simbol*(str.substring(0, 1), *str\_temp\_term*) && *check\_Simbol*(str.substring(1, 2), *str\_temp\_term*)) {  
 String s\_temp\_nt = *find\_Simbol*();  
 String s\_temp\_nt1 = **""**;  
 temp.add(str.substring(0 , 1) + s\_temp\_nt);  
 ArrayList <String> ss = **new** ArrayList<>();  
 **for** (**int** i = 1; i < *countChar*(str) - 1; i++ ) {  
 s\_temp\_nt1 = *find\_Simbol*();  
 ss.add(str.substring(i, i + 1) + s\_temp\_nt1);  
 *rule\_True*.put(s\_temp\_nt, ss);  
 s\_temp\_nt = s\_temp\_nt1;  
 ss = **new** ArrayList<>();  
 }  
 ss.add(str.substring(str.length() - 1) + **"T"**);  
 *rule\_True*.put(s\_temp\_nt, ss);  
 } **else if** (str.length() == 2 && *check\_Simbol*(str.substring(0, 1), *str\_temp\_noterm*) && *check\_Simbol*(str.substring(1, 2), *str\_temp\_noterm*)) {  
 **return false**;  
 } **else if** (str.length() == 2 && *check\_Simbol*(str.substring(0, 1), *str\_temp\_noterm*) && *check\_Simbol*(str.substring(1, 2), *str\_temp\_term*)) {  
 **return false**;  
 } **else if** (str.length() == 1 && *check\_Simbol*(str.substring(0, 1), *str\_temp\_noterm*)) {  
 **return false**;  
 } **else** {  
 temp.add(str);  
 }  
 }  
 *rule\_True*.put(s, temp);  
 temp = **new** ArrayList<>();  
 }  
  
 } **else** {  
 **for** (String s: *str\_temp\_noterm*) {  
 **for** (String str : *rule*.get(s)) {  
 **if** (str.length() > 2) {  
 **if** (str.length() - *countChar*(str) == 0) {  
 String s\_temp\_nt = *find\_Simbol*();  
 String s\_temp\_nt1 = **""**;  
 temp.add(s\_temp\_nt + str.substring(str.length() - 1));  
 ArrayList <String> ss = **new** ArrayList<>();  
 **for** (**int** i = str.length() - 2; i > 0; i-- ) {  
 s\_temp\_nt1 = *find\_Simbol*();  
 ss.add(s\_temp\_nt1 + str.substring(i, i + 1));  
 *rule\_True*.put(s\_temp\_nt, ss);  
 s\_temp\_nt = s\_temp\_nt1;  
 ss = **new** ArrayList<>();  
 }  
 ss.add(**"T"** + str.substring(0, 1));  
 *rule\_True*.put(s\_temp\_nt, ss);  
 } **else** {  
 **return false**;  
 }  
 } **else if** (str.length() == 2 && *check\_Simbol*(str.substring(0, 1), *str\_temp\_term*) && *check\_Simbol*(str.substring(1, 2), *str\_temp\_term*)) {  
 String s\_temp\_nt = *find\_Simbol*();  
 String s\_temp\_nt1 = **""**;  
 temp.add(s\_temp\_nt + str.substring(str.length() - 1));  
 ArrayList <String> ss = **new** ArrayList<>();  
 **for** (**int** i = str.length() - 2; i > 0; i-- ) {  
 s\_temp\_nt1 = *find\_Simbol*();  
 ss.add(s\_temp\_nt1 + str.substring(i, i + 1));  
 *rule\_True*.put(s\_temp\_nt, ss);  
 s\_temp\_nt = s\_temp\_nt1;  
 ss = **new** ArrayList<>();  
 }  
 ss.add(**"T"** + str.substring(0, 1));  
 *rule\_True*.put(s\_temp\_nt, ss);  
 } **else if** (str.length() == 2 && *check\_Simbol*(str.substring(0, 1), *str\_temp\_noterm*) && *check\_Simbol*(str.substring(1, 2), *str\_temp\_noterm*)) {  
 **return false**;  
 } **else if** (str.length() == 2 && *check\_Simbol*(str.substring(0, 1), *str\_temp\_term*) && *check\_Simbol*(str.substring(1, 2), *str\_temp\_noterm*)) {  
 **return false**;  
 } **else if** (str.length() == 1 && *check\_Simbol*(str.substring(0, 1), *str\_temp\_noterm*)) {  
 **return false**;  
 } **else** {  
 temp.add(str);  
 }  
 }  
 *rule\_True*.put(s, temp);  
 temp = **new** ArrayList<>();  
 }  
 }  
 temp = **new** ArrayList<>();  
 temp.add(**"#"**);  
 *rule\_True*.put(**"T"**, temp);  
 **return true**;  
  
}

**public static void** NKA() {  
 *end\_pos*.add(**"T"**);  
 *str\_temp\_noterm*.add(**"T"**);  
 *str\_Temp\_noterm\_True*.add(**"T"**);  
 HashMap <String, String> temp = **new** HashMap<>();  
 **for** (String str3: *str\_temp\_term*) {  
 **for** (String str1: *str\_Temp\_noterm\_True*) {  
 String p = **""**;  
 **for** (String str2: *rule\_True*.get(str1)) {  
 **if** (str2.contains(str3)) {  
 **if** (str2.length() == 2) {  
 **if** (*check\_Simbol*(str2.substring(0, 1), *str\_Temp\_noterm\_True*)) {  
  
 p += str2.substring(0, 1);  
 } **else** {  
 p += str2.substring(1, 2);  
 }  
 } **else** {  
 p += str1;  
 *end\_pos*.add(str1);  
 }  
 }  
 }  
 **boolean** flag\_dka = **true**;  
 **for** (String t\_s:*str\_Temp\_noterm\_DKA*) {  
 **if** (p.equals(t\_s) && !p.isEmpty()) {  
 flag\_dka = **false**;  
 **break**;  
 }  
 }  
 **if** (flag\_dka && !p.isEmpty()) {  
 **boolean** flag\_end = **true**;  
 **for** (String pts: *end\_pos*) {  
 **if** (p.equals(pts)) {  
 flag\_end = **false**;  
 }  
 }  
 **if** (flag\_end) {  
 **boolean** flag\_end1 = **false**;  
 **for** (String pts: *end\_pos*) {  
 **if** (p.contains(pts)) {  
 flag\_end1 = **true**;  
 }  
 }  
 **if** (flag\_end1) {  
 *end\_pos*.add(p);  
 }  
 }  
 *str\_Temp\_noterm\_DKA*.add(p);  
 }  
 temp.put(str1, p);  
 }  
 **if** (!str3.equals(**"#"**)) {  
 *NKA\_map*.put(str3, temp);  
 }  
 temp = **new** HashMap<>();  
  
 }  
}  
  
**public static void** DKA() {  
 HashMap<String, String> temp = **new** HashMap<>();  
 ArrayList<String> m1 = **new** ArrayList<>(*str\_Temp\_noterm\_DKA*);  
 **while** (**true**) {  
 **for** (String str1 : *str\_temp\_term*) {  
 **if** (str1.equals(**"#"**)) {  
 **continue**;  
 }  
 **for** (String str2 : *str\_Temp\_noterm\_DKA*) {  
 String p = **""**;  
 **boolean** flag\_dka = **true**;  
 **for** (String t\_s : *str\_Temp\_noterm\_True*) {  
 **if** (str2.equals(t\_s)) {  
 flag\_dka = **false**;  
 **break**;  
 }  
 }  
 **if** (flag\_dka) {  
 String pp = **""**;  
 **for** (**int** i = 0; i < str2.length(); i++) {  
 pp += *NKA\_map*.get(str1).get(str2.substring(i, i + 1));  
 }  
  
 temp.put(str2, pp);  
 **boolean** flag\_dka1 = **true**;  
 **for** (String t\_s : *str\_Temp\_noterm\_DKA*) {  
 **if** (pp.equals(t\_s)) {  
 flag\_dka1 = **false**;  
 **break**;  
 }  
 }  
 **if** (flag\_dka1 && !pp.isEmpty()) {  
 **boolean** flag\_end = **true**;  
 **for** (String pts: *end\_pos*) {  
 **if** (pp.equals(pts)) {  
 flag\_end = **false**;  
 }  
 }  
 **if** (flag\_end) {  
 **boolean** flag\_end1 = **false**;  
 **for** (String pts: *end\_pos*) {  
 **if** (pp.contains(pts)) {  
 flag\_end1 = **true**;  
 }  
 }  
 **if** (flag\_end1) {  
 *end\_pos*.add(pp);  
 }  
 }  
 m1.add(pp);  
 }  
 } **else** {  
  
 temp.put(str2, *NKA\_map*.get(str1).get(str2));  
 }  
 }  
 *DKA\_map*.put(str1, temp);  
 temp = **new** HashMap<>();  
 }  
 **if** (m1.size() == *str\_Temp\_noterm\_DKA*.size()) {  
 **break**;  
 } **else** {  
 *DKA\_map*.clear();  
 *str\_Temp\_noterm\_DKA*.clear();  
 *str\_Temp\_noterm\_DKA*.addAll(m1);  
 }  
 }  
  
  
 ArrayList <String> true\_nonterm;  
 ArrayList <String> temp\_nonterm = **new** ArrayList<>();  
 temp\_nonterm.add(*jt\_str\_begin*);  
 System.***out***.println(*DKA\_map*);  
 **do** {  
 true\_nonterm = **new** ArrayList<>();  
 true\_nonterm.addAll(temp\_nonterm);  
 **for** (String fnt: true\_nonterm) {  
 **for** (String ft: *str\_temp\_term*) {  
 **if** (ft.equals(**"#"**)) {  
 **continue**;  
 }  
 String lom = *DKA\_map*.get(ft).get(fnt);  
 **if**(!lom.isEmpty()) {  
 **boolean** flag\_end = **true**;  
 **for** (String pts: true\_nonterm) {  
 **if** (lom.equals(pts)) {  
 flag\_end = **false**;  
 }  
 }  
 **if** (flag\_end) {  
 temp\_nonterm.add(lom);  
 }  
 }  
 }  
 }  
 }**while** (true\_nonterm.size() != temp\_nonterm.size());  
  
 System.***out***.println(true\_nonterm);  
  
 **for** (String str1: *str\_temp\_term*) {  
 **if** (str1.equals(**"#"**)) {  
 **continue**;  
 }  
 HashMap <String, String> pp = **new** HashMap<>();  
 **for** (String str2: true\_nonterm) {  
 pp.put(str2, *DKA\_map*.get(str1).get(str2));  
 }  
 *DKA\_map\_True*.put(str1, pp);  
 }  
  
 System.***out***.println(*DKA\_map\_True*);  
  
 String[] stp = **new** String[*str\_temp\_term*.size()];  
 **int** i = 1;  
 stp[0] = **""**;  
 **for** (String str: *str\_temp\_term*) {  
 **if** (str.equals(**"#"**)) {  
 **continue**;  
 }  
 stp[i] = str;  
 i++;  
 }  
  
 String[][] stp\_mas = **new** String[true\_nonterm.size()][*str\_temp\_term*.size()];  
 **for** (**int** j = 0; j < true\_nonterm.size(); j++) {  
 String[] st = **new** String[*str\_temp\_term*.size()];  
 st[0] = true\_nonterm.get(j);  
 **int** f = 1;  
 **for** (String spo: *str\_temp\_term*) {  
 **if** (spo.equals(**"#"**)) {  
 **continue**;  
 }  
  
 st[f] = *DKA\_map\_True*.get(spo).get(true\_nonterm.get(j));  
 f++;  
 }  
 stp\_mas[j] = st;  
 }  
  
 JTable table = **new** JTable(stp\_mas,stp);  
 JScrollPane scroll = **new** JScrollPane(table);  
 *jp\_table*.add(scroll);  
 JButton ne = **new** JButton(**"Next"**);  
 ne.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 *generation*(*rule\_True*, *str\_Temp\_noterm\_True*, *answer*, *number\_chain*);  
 *jp\_table*.setVisible(**false**);  
 System.***out***.println(*answer*);  
 System.***out***.println(*number\_chain*);  
 JTextArea textArea = **new** JTextArea(7, 60);  
 **for** (**int** i = *num\_begin*; i <= *num\_end*; i++) {  
 String pt = **new** String();  
 **for**(String str: *answer*.get(Integer.*toString*(i))) {  
 String sss = *number\_chain*.get(i).get(str);  
 **for** (**int** j = 1; j <= sss.length(); j++) {  
 **for** (String p: *number\_chain*.get(i).keySet()) {  
 **if** (*number\_chain*.get(i).get(p).equals(sss.substring(0, j)))  
 {  
 pt += p + **" -> "**;  
 **break**;  
 }  
 }  
 }  
 textArea.append( pt + **"ЦЕПОЧКА ПОЛУЧЕНА \n"**);  
 textArea.append(*DKA\_check*(str));  
 pt = **new** String();  
 }  
 }  
  
 JScrollPane scroll1 = **new** JScrollPane(textArea,  
 JScrollPane.***VERTICAL\_SCROLLBAR\_ALWAYS***,  
 JScrollPane.***HORIZONTAL\_SCROLLBAR\_ALWAYS***);  
 *jp\_chain*.add(scroll1);  
 JPanel panel = **new** JPanel();  
 JButton button = **new** JButton(**"Check"**);  
 JTextField text = **new** JTextField(15);  
 button.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 JOptionPane.*showMessageDialog*(*myWindow*, *DKA\_check*(text.getText()));  
 }  
 });  
  
 panel.add(text);  
 panel.add(button);  
 *jp\_chain*.add(panel);  
 *myWindow*.add(*jp\_chain*);  
  
  
 }  
 });  
 *jp\_table*.add(ne);  
 *myWindow*.add(*jp\_table*);  
}  
  
**public static void** generation(HashMap <String, ArrayList<String>> map, ArrayList <String> list,  
 HashMap <String, ArrayList<String>> ans, HashMap<Integer, HashMap<String, String>> chain) {  
 **boolean** flag\_all = **true**;  
 **boolean** flag\_term = **true**;  
 **boolean** flag\_add = **true**;  
 **int** counter = 0;  
 **int** add\_tree = 1;  
 ArrayList<String> m1 = **new** ArrayList<>();  
 ArrayList<String> m2 = **new** ArrayList<>();  
 ArrayList<String> answer = **new** ArrayList<>();  
 String split1 = **""**, split2 = **""**;  
 String s = **null**;  
 String ss = **null**;  
 HashMap <String, String> chain\_tamp = **new** HashMap<>();  
 m1.add(*jt\_str\_begin*);  
 **for** (**int** f = *num\_begin*; f <= *num\_end*; f++) {  
 answer = **new** ArrayList<>();  
 flag\_all = **true**;  
 flag\_term = **true**;  
 flag\_add = **true**;  
 m1.clear();  
 m2.clear();  
 split1 = **""**;  
 split2 = **""**;  
 s = **null**;  
 ss = **null**;  
 m1.add(*jt\_str\_begin*);  
 counter = 0;  
 chain\_tamp = **new** HashMap<>();  
 chain\_tamp.put(**"S"**, **"1"**);  
 **while** (flag\_all) {  
 **for** (String str1 : m1) {  
 split1 = **""**;  
 split2 = **""**;  
 counter = 0;  
 flag\_term = **true**;  
 add\_tree = 1;  
 **while** (flag\_term) {  
 s = str1.substring(counter, counter + 1);  
 **for** (String str2 : list) {  
 **if** (str2.equals(s)) {  
 flag\_term = **false**;  
 **if** (counter == 0) {  
 split2 = str1.substring(counter + 1);  
 } **else** {  
 split2 = str1.substring(counter + 1);  
 split1 = str1.substring(0, counter);  
 }  
 counter = 0;  
 }  
 }  
  
 counter++;  
 }  
  
 **for** (String str : map.get(s)) {  
 flag\_add = **true**;  
 ss = split1 + str + split2;  
 **if** (str.equals(**"#"**)) {  
 ss = split1 + split2;  
 }  
 **if** (*countCharsimple*(ss) > f) {  
 **continue**;  
 }  
  
  
 **for** (String str3 : list) {  
 **if** (ss.contains(str3)) {  
 flag\_add = **false**;  
 m2.add(ss);  
 chain\_tamp.put(ss, chain\_tamp.get(str1) + add\_tree);  
 add\_tree++;  
 **break**;  
 }  
 }  
  
 **boolean** flag = **true**;  
 **if** (flag\_add && ss.length() == f) {  
 chain\_tamp.put(ss, chain\_tamp.get(str1) + add\_tree);  
 add\_tree++;  
 **for** (String sss : answer) {  
 **if** (sss.equals(ss)) {  
 flag = **false**;  
 }  
 }  
 **if** (flag) {  
 answer.add(ss);  
 }  
 }  
 }  
 }  
  
 m1.clear();  
 **for** (String str : m2) {  
 m1.add(str);  
 }  
 m2.clear();  
 **if** (m1.isEmpty()) {  
 flag\_all = **false**;  
 }  
 }  
 ans.put(Integer.*toString*(f), answer);  
 chain.put(f, chain\_tamp);  
 }  
  
}

**public static** String DKA\_check(String str) {  
 String sost = *jt\_str\_begin*;  
 String s = **"{ "** + sost + **" , "** + str + **" } |- "**;  
 **try** {  
  
 **for** (**int** i = 0; i < str.length(); i++) {  
 sost = *DKA\_map\_True*.get(str.substring(i, i + 1)).get(sost);  
 s += **"{ "** + sost + **" , "** + str.substring(i + 1) + **" } |- "**;  
 System.***out***.println(s);  
 }  
 **boolean** spt = **true**;  
 **for** (String ks : *end\_pos*) {  
 **if** (ks.equals(sost)) {  
 spt = **false**;  
 }  
 }  
 **if** (spt) {  
 JOptionPane.*showMessageDialog*(*myWindow*, **"ОШИБКА, КОНЕЧНОЕ СОСТОЯНИЕ НЕ ДОСТИГНУТО"**);  
 **return "ERROR"**;  
 }  
 } **catch** (Exception e) {  
 JOptionPane.*showMessageDialog*(*myWindow*, **"ЦЕПОЧКА НЕВОЗМОЖНА"**);  
 **return "ERROR"**;  
 }  
 **return** s + **"ЦЕПОЧА ВЕРНАЯ \n"**;  
 }

}