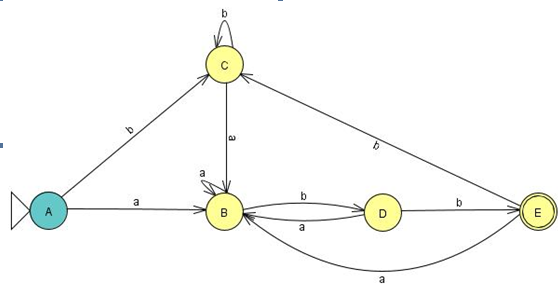
Alunos: Anderson Ferreira Canel 16102264, Leonardo Souza

1. Faça um implementação em C ou Java, do diagrama de transição de estado do AFD a seguir.



import java.util.ArrayList;

import java.util.Arrays;

import java.util.HashMap;

import java.util.List;

import javax.swing.JOptionPane;

/\* @Anderson Ferreira Canel \ Leonardo Souza

\*

\* \* Este código é capaz de verificar a aceitação de uma palavra por um dado automato,

\* caso queira verificar a aceitação de outras palavras por outros automatos altere a

\* regra de produção e os conjuntos de estados e simbolos hardcoded.

\* \*/

public class AFD {

static HashMap<Integer, String> conjuntoDeEstadosMap = new HashMap<Integer, String>();

static HashMap<Integer, String> conjuntoDeEstadosFinaisMap = new HashMap<Integer, String>();

static String alfabetoImprime;

static String conjuntoDeEstadosTerminaisImprime;

public static void main(String[] args) {

/\*\*

\* Trecho responsável por receber o conjunto de simbolos(Alfabeto) e armazenar o

\* mesmo sem os caracteres desnecessários em um array de char.

\*/

String alfabeto = "{a,b}";

alfabetoImprime = alfabeto;

alfabeto = removeNulos(alfabeto);// Removendo {,}

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

Character[] conjuntodeSimbolos\_Alfabeto = new Character[alfabeto.length()];

int z = 0;

for (char ch : alfabeto.toCharArray()) {

alf.add(ch);

conjuntodeSimbolos\_Alfabeto[z] = ch;

z++;

}

/\*\*

\* Trecho responsável por receber o conjunto de estados(Terminais e não

\* Terminais), e armazenar o mesmo sem os caracteres desnecessários em um array

\* de int, convertendo a posição de um estado formecido no conjunto em um valor

\* númerico, em ordem crescente correspondente a ordem dos estados fornecidos.

\*/

String conjuntoDeEstadosTerminaisEnaoTerminais = "{A,B,C,D,E}";

conjuntoDeEstadosTerminaisImprime = conjuntoDeEstadosTerminaisEnaoTerminais;

conjuntoDeEstadosTerminaisEnaoTerminais = removeNulos(conjuntoDeEstadosTerminaisEnaoTerminais);// Removendo {,}

int estados = conjuntoDeEstadosTerminaisEnaoTerminais.length();

int[] conjuntoDeEstados = new int[estados];

int a = 0;

for (Character ch : conjuntoDeEstadosTerminaisEnaoTerminais.toCharArray()) {

conjuntoDeEstadosMap.put(a, ch.toString());

// IMPRIMINDO CONJUNTO DE ESTADOS NO CONSOLE

conjuntoDeEstados[a] = a;

a++;

}

/\*\*

\* Trecho responsável por receber entrada do conjundo de regras de

\* transição(Regra de Produção), funciona da seguinte forma:

\* # ESTADO (LADO ESQUERDO), CONSOME(CENTRO); VAI PARA ESTADO(LADO DIREITO)#

\*/

String[] funcDelta = new String[conjuntodeSimbolos\_Alfabeto.length

\* conjuntoDeEstadosTerminaisEnaoTerminais.length()];

funcDelta[0] = "A,a;B";

funcDelta[1] = "A,b;C";

funcDelta[2] = "B,a;B";

funcDelta[3] = "B,b;D";

funcDelta[4] = "C,a;B";

funcDelta[5] = "C,b;C";

funcDelta[6] = "D,a;B";

funcDelta[7] = "D,b;E";

funcDelta[8] = "E,a;B";

funcDelta[9] = "E,b;C";

String[] estadoPartidaS = new String[funcDelta.length];

String[] caracConsumidoS = new String[funcDelta.length];

String[] estadoDestinoS = new String[funcDelta.length];

int[] estadoPartida = new int[funcDelta.length];

int[] estadoDestino = new int[funcDelta.length];

char[] le = new char[funcDelta.length];

for (int i = 0; i < funcDelta.length; i++) {

if (funcDelta[i] == null) {

break;

}

String[] p1 = funcDelta[i].split(";");

String[] p2 = p1[0].split(",");

estadoPartidaS[i] = p2[0];

caracConsumidoS[i] = p2[1];

estadoDestinoS[i] = p1[1];

}

for (int p = 0; p < funcDelta.length; p++) {

String aux = estadoPartidaS[p];

int h = 0;

for (Character ch : conjuntoDeEstadosTerminaisEnaoTerminais.toCharArray()) {

for (int j = 0; j < (conjuntodeSimbolos\_Alfabeto.length

\* conjuntoDeEstadosTerminaisEnaoTerminais.length()); j++) {

if (estadoPartidaS[j].equals(ch.toString())) {

estadoPartida[j] = h;

}

}

for (int j = 0; j < (conjuntodeSimbolos\_Alfabeto.length

\* conjuntoDeEstadosTerminaisEnaoTerminais.length()); j++) {

if (estadoDestinoS[j].equals(ch.toString())) {

estadoDestino[j] = h;

}

}

h++;

}

aux = caracConsumidoS[p];

le[p] = aux.charAt(0);

}

/\*\*

\* Trecho responsável por receber o estado inicial, identificar posição

\* correspondente no conjunto de estados(Terminais e não terminais) e armazena o mesmo sem os caracteres

\* desnecessários em uma variável do tipo int com o valor correspondente a sua posição no conjunto de

\* estados.

\*/

String estadoIni = "{A}";

String estIniImprime = estadoIni;

int estadoi = 0;

estadoIni = removeNulos(estadoIni);

int u = 0;

for (Character ch : conjuntoDeEstadosTerminaisEnaoTerminais.toCharArray()) {

if (ch.toString().equals(estadoIni)) {

estadoi = u;

break;

}

u++;

}

/\*\*

\* Trecho responsável por receber o conjunto de estados finais (Terminais), e

\* armazenar o mesmo sem os caracteres desnecessários em um array de int,

\* indentificando a posição de um estado formecido no conjunto em um valor

\* númerico, corespondente a pocisão do mesmo no conjunto de estados (Terminais

\* e não Terminais)e armazenando estes valores em um array de tipo int.

\*/

String conjuntoEstadosTerminais = "{E}";

String conjEstTermImprime = conjuntoEstadosTerminais;

conjuntoEstadosTerminais = removeNulos(conjuntoEstadosTerminais);

int[] estadosf = new int[conjuntoEstadosTerminais.length()];

int b = 0, y = 0;

for (Character ch : conjuntoDeEstadosTerminaisEnaoTerminais.toCharArray()) {

for (Character ch1 : conjuntoEstadosTerminais.toCharArray()) {

if (conjuntoDeEstadosMap.get(y).equals(ch1.toString())) {

estadosf[b] = y;

b++;

break;

}

}

y++;

}

imprimirAutomato(alfabetoImprime, conjuntoDeEstadosTerminaisImprime, estadoPartida, estadoDestino, le,

estIniImprime, conjEstTermImprime);

/\*

\* Entrada realizada pelo usuário, realiza vereificação para checkar se a

\* palavra pode ser formada com os caracteres do conjunto de simbolo (alfabeto).

\*/

String palavraS;

boolean flagPal;

do {

int teste = 0;

int w = 0;

palavraS = JOptionPane.showInputDialog(null,

"Entre com a palavra a ser verificada: \nPara conferir os valores dos conjuntos e regras de produção digite 'i'\nPara sair digite s");

if (palavraS.equalsIgnoreCase("s")) {

break;

}

if (palavraS.equalsIgnoreCase("I")) {

imprimirAutomato(alfabetoImprime, conjuntoDeEstadosTerminaisImprime, estadoPartida, estadoDestino, le,

estadoIni, conjuntoEstadosTerminais);

palavraS = JOptionPane.showInputDialog(null,

"Entre com a palavra a ser verificada: \nPara conferir os valores dos cinjuntos e regras de produção digite 'i'\nPara sair digite s");

if (palavraS.equalsIgnoreCase("s")) {

break;

}

}

//Variável reponsável por receber a validação da palavra pelo automato

flagPal = VerificaPalavra(palavraS, conjuntodeSimbolos\_Alfabeto);

if (!flagPal) {

} else {

char[] palavra = palavraS.toCharArray();

int estadoa = estadoi;

for (int p = 0; p < palavra.length; p++) {

for (int k = 0; k < funcDelta.length; k++) {

if ((palavra[p] == le[k]) && (estadoPartida[k] == estadoa)) {

estadoa = estadoDestino[k];

w++;

break;

} else {

}

}

for (int k = 0; k < conjuntoEstadosTerminais.length(); k++) {

if (estadoa == estadosf[k]) {

teste = 1;

} else {

teste = 0;

}

}

}

if (teste == 1) {

JOptionPane.showMessageDialog(null, "PALAVRA ACEITA PELO AUTOMATO\n\n");

// break;

} else {

JOptionPane.showMessageDialog(null, "PALAVRA NÃO ACEITA PELO AUTOMATO\n\n");

// break;

}

}

} while (!palavraS.equalsIgnoreCase("s"));

JOptionPane.showMessageDialog(null, "Voce finalizou a aplição, obrigado!");

}

// ---------------METODOS UTILIZADOS NO CÓDIGO------------

//IMPRIME CONJUNTOS E REGRAS DE PRODUÇÃO

private static void imprimirAutomato(String alf, String est, int[] estadoPartida, int[] estadoDestino, char[] le,

String estIn, String conjuntoEstadosFinais) {

String[] estP = new String[estadoPartida.length];

;

int b = 0;

for (int key : estadoPartida) {

estP[b] = conjuntoDeEstadosMap.get(key);

b++;

}

String[] estD = new String[estadoDestino.length];

;

int c = 0;

for (int key : estadoDestino) {

estD[c] = conjuntoDeEstadosMap.get(key);

c++;

}

JOptionPane.showMessageDialog(null,

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n" + "\tIMPRIMINDO DADOS DO AUTOMATO\n"

+ "\t\t\t ==>NOTAÇÃO UTILIZADA <== \n" + "\tO conjunto de simbolos - alfabeto: Σ \n"

+ "\tO conjunto dos estados terminais e não terminais: Q = {S1, S2...}\n"

+ "\tAs transicoes: (δ: Q × Σ → Q)\n" + "\tO estado Inicial: q0\n"

+ "\tO conjunto dos estados terminais: F\n" + "\tM = (Q, Σ, (δ: Q × Σ → Q), q0, F)\n"

+ "\n\t\t ==>DADOS INFORMADOS <==\n" + "\tΣ = " + alf + "\n" + "" + "\tQ = " + est + "\n"

+ "\tδ = \n" + "ESTADO PARTIDA: Q" + Arrays.toString(estP) + "\n"

+ "CARACTER CONSUMIDO: Σ" + Arrays.toString(le) + "\n" + "ESTADO DESTINO: Q"

+ Arrays.toString(estD) + "\n" + "" + "\tq0 = " + estIn + "\n" + "" + "\tF = "

+ conjuntoEstadosFinais + "\n" + "" + "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

//VERIFICA SE PALAVRA PERTENCE AO ALFABETO

private static boolean VerificaPalavra(String palavra, Character[] alf) {

int cont = 0;

for (int x = 0; x < palavra.length(); x++) {

Character caracPalavra = palavra.charAt(x);

for (int y = 0; y < alf.length; y++) {

if (caracPalavra.equals(alf[y])) {

cont++;

}

}

}

if (cont == palavra.length()) {

return true;

} else {

JOptionPane.showMessageDialog(null,

"A palavra \"" + palavra

+ "\" contém simbolos não pertencentes ao conjunto de simbolos (alfabeto,Σ= "

+ alfabetoImprime + ")!",

"WARNING", JOptionPane.WARNING\_MESSAGE);

return false;

}

}

//REMOVE CARACTERES DE FORMATAÇÃO DO CONJUNTO, EX: {,}

public static String removeNulos(String conjunto) {

String[] nulos = { "{", "}", "," };// identificando carateres de formatação do conjunto

for (String n : nulos) {

conjunto = conjunto.replace(n, "");

}

return conjunto;

}

}

Teste o autômato com várias palavras para vê se ele aceita ou rejeita as palavras.

**A saída deverá dizer se a palavra foi aceita ou rejeitada conforme os princípios dos AF.**