Universidade Federal do Espírito Santo

Uarlley do Nascimento Amorim - Matricula: 2018205346
15 de agosto de 2021

Linguagens Formais e Autômatos

Professor: Faimison Rodrigues Porto

Ciência da computação

Sumário

1 Descrição do Trabalho

Um autômato finito determinístico (AFD) é uma máquina de estados finita que aceita ou rejeita cadeias de símbolos gerando um único ramo de computação para cada cadeia de entrada. O objetivo deste trabalho é desenvolver um programa que simula o funcionamento de um AFD a partir dos seguintes dados:

- Quantidade de estados;
- Estado inicial;
- Quantidade de estados terminais;
- Estados terminais;
- Quantidade de transições;
- Lista de transições.
- Quantidade de testes;
- Cadeias de teste.

A saída deve ser, para da caso de teste, "aceita"ou "rejeita"seguido de uma quebra de linha.

2 Bibliotecas Utilizadas

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
```

3 Estrutura de Transição

Para simplificar o método de transição, foi implementada uma estrutura com 3 campos, estado antes da transição, caractere de transição, e, por fim, o estado após a transição.

```c
typedef struct t_transition{
    int from;
    int to;
    char character;
}transition;
```

4 Variáveis Globais

Para simplificar a utilização das funções, os arrays que armazenam o alfabeto, transições e estados terminais foram declarados globalmente juntamente com as variáveis que guardam o tamanho dos mesmos.

```c
char* alphabet; //alphabet definition
int alphabetSize; //alphabet size
transition *transitions; //transitions between states
int numTransitions; //number of transitions
int *terminalStates; //terminal states
int numTerminalStates; // number of terminal states
```

5 Funções Auxiliares

Nesta sessão, serão apresentadas as funções auxiliares, utilizadas para modularizar e simplificar o entendimento da implementação.

5.1 É estado terminal

Esta função booleana verifica se um determinado estado é ou não um estado terminal. Para isso, o estado a ser verificado é passado como parâmetro, e, então, o mesmo é buscado no array de estados terminais e, se encontrado, é retornado *true* e $false$ caso contrário.

```
16 bool isTerminal(int state){
17     //returns whether the state is terminal or not
18     for(int i = 0; i < numTerminalStates; i++){
19         if(state == terminalStates[i]) return true;
20     }
21     return false;
22 }
```

5.2 Pertence ao alfabeto

Esta função verifica se um determinado caractere pertence ou não ao alfabeto do AFD. Para isso, o caractere passado é buscado no array que guarda o alfabeto e, se encontrado a função é finalizada retornando *true*. O fim da busca sinaliza a ausência do caractere no alfabeto, logo a função é finalizada retornando $false$.

```
23 bool isInAlphabet(char character){
24     //checks if the character belongs to the alphabet
25     for(int i = 0; i < alphabetSize; i++){
26         if(alphabet[i] == character) return true;
27     }
28     return false;
29 }
```

5.3 Procura próximo estado

A função abaixo, busca e retorna qual deve ser o próximo estado a partir do estado atual e o caractere de transição. Assim é buscado no array de transições uma transição cujo o estado antes da transição e caractere de transição são iguais aos que foram passados como parâmetros da função, e, se tal transição for encontrada, o estado após a transição é retornado e a função é finalizada. O fim da busca sinaliza a ausência de tal transição, dessa forma, a função é finalizada retornando -1.

```
30 int searchNextState(char character, int state){
31     //searches for the next state using the character
32     for(int i = 0; i < numTransitions; i++){
33         if(transitions[i].from == state && transitions[i].character == character) return
               transitions[i].to;
34     }
35     //returns -1 if theres no next state
36     return -1;
37 }
```

6 Função de validação

Esta função recursiva é responsável por, dada uma cadeia de caracteres, determinar se o autômato aceita ou rejeita tal cadeia.

A condição de parada da função, consiste em verificar se a cadeia está vazia ou se a leitura da mesma chegou ao fim. Se sim, é verificado se o estado atual é ou não um estado terminal. Se verdadeiro, a cadeia é válida, e, portanto, a função é finalizada retornando $true$. Caso contrário, a cadeia é rejeitada pelo autômato, e, portanto, a função é finalizada retornando $false$.

Se a cadeia não esta vazia e a leitura da mesma não chegou ao fim, é verificado se o caractere atual pertence ou não ao alfabeto através da função apresentada na seção (5.2), se o retorno da função for $false$, o autômato rejeita a cadeia, e portanto, a função é finalizada retornando $false$. Caso contrário, o próximo estado é buscado utilizando a função mostrada na seção (5.3). Se o retorno for igual a -1, significa que o estado atual não transita para nenhum outro estado através do caractere atual, o que torna a cadeia inválida e, com isso, a função é finalizada retornando $false$. Caso contrário, é realizada a chamada recursiva da função com o novo estado e o próximo caractere da cadeia.

```
39 bool isValid(char *expression, int currentState, int pos, int end){
40     //checks if the expression is empty
41     if(pos > end || expression[pos] == '-') {
42         //returns whether the expression ended in a terminal state or not
43          if (isTerminal(currentState)) return true;
44          else return false;
45     }
46     //checks if the current character belongs to the alphabet
47     if(!isInAlphabet(expression[pos])) return false;
48 
49     //changes to the next state
50     int newState = searchNextState(expression[pos], currentState);
51     //returns false if theres no next state
52     if(newState == -1) return false;
53 
54     //makes the recursive call with the new state and expression
55     return isValid(expression, newState, pos+1, end);
56 }
```

7 Função Principal

A função *main* é responsável por realizar a leitura dos dados de entrada, alocar a memoria necessária para armazenar tais dados e validar a lista de cadeias. Após realização dos testes, a memória alocada é liberada e o programa é finalizado.

```
57 
58 int main(){
59     int states, initStateSize, i = 0, inputs;
60     //reads the input list
61     scanf("%d", &states);
62     scanf("%d", &alphabetSize);
63     alphabet = (char *)malloc((alphabetSize+1)*sizeof(char));//allocates the alphabet array
64     getchar();
65 
66     while(i < alphabetSize){ //reads the alphabet ignoring spaces
67         alphabet[i] = getchar();
68         if(alphabet[i] != ' ') {
69             i++;
70         }
```

```
71      }
72      alphabet[alphabetSize] = 0;
73      scanf("%d", &initStateSize);
74      scanf("%d", &numTerminalStates);
75
76      terminalStates = (int *)malloc(numTerminalStates*sizeof(int)); //allocates the array of
            terminal states
77      for(i = 0; i < numTerminalStates; i++){
78          scanf("%d", &terminalStates[i]);
79      }
80      scanf("%d",&numTransitions);
81      getchar();
82      transitions = (transition *)malloc(numTransitions*sizeof(transition)); //allocates the array
            of transitions
83      for(i = 0; i < numTransitions; i++){ //reads the transitions ignoring spaces
84          scanf("%d",&transitions[i].from);
85          getchar();
86          scanf("%c",&transitions[i].character);
87          getchar();
88          scanf("%d",&transitions[i].to);
89      }
90      //reads, tests and print the correct output
91      scanf("%d", &inputs);
92      getchar();
93      for(i = 0; i < inputs; i++){
94          char test[50];
95          fgets(test,50,stdin);
96          test[strlen(test)-1] = 0;
97          if(isValid(test, 0,0,strlen(test)-1)) printf("aceita\n");
98          else printf("rejeita\n");
99      }
100     //frees the allocated memory
101     free(alphabet);
102     free(transitions);
103     free(terminalStates);
104     return 0;
105 }
```

8 Análise da Complexidade

Seja T um AFD, m a quantidade de transições de T, A o alfabeto, k a quantidade de caracteres em A, C a cadeia de caracteres a ser avaliada e n a quantidade total de caracteres presentes em C. Assim, no pior caso (caso em que o autômato aceita a cadeia C), para cada caractere em C, é verificado se o mesmo pertence ao alfabeto A e qual das m transições será a próxima, totalizando $(k + m)$ iterações. Dessa forma, podemos assumir que, como o processo é repetido para cada caractere de C, a complexidade do programa é $O(n \cdot (k + m))$.

9 Conclusões

Como os testes disponibilizados no Coffe eram de complexidade baixa, o programa conseguiu obter a resposta em um tempo médio de 0.002 segundos. Dessa forma, a fim de avaliar o desempenho e confiabilidade do programa de forma mais profunda, foi criado um programa utilizando a linguagem C++ e um bash script, mostrados na seção 10, para gerar arquivos de teste, utiliza-los no programa e, em seguida, gerar arquivos de saída. A complexidade e tamanho dos testes podem ser alterados de forma simples. A partir disso, foram

testados instâncias com mais de 50 cadeias, cujo o tamanho ultrapassava 500 caracteres, e ainda assim, o programa gerou os arquivos de saída em um tempo médio de 0.3 segundos.

Em relação à utilização de memoria, no programa, foi alocada memoria para o array de transições, alfabeto e estados terminais, e, como tais array são considerados pequenos até para testes complexos, uma quantidade consideravelmente pequena de memória é necessária para executa-lo.

10 Apêndices

10.1 Apêndice 1

```
106 #include <bits/stdc++.h>
107 using namespace std;
108
109
110 int maxTests = 50;
111 int minExpressionSize = 40;
112 int maxExpressionSize = 199;
113 int minStates = 3;
114 int maxStates = 10;
115 int minTransitions = 50;
116 int maxTransitions = 70;
117 int maxTerminalStates = 5;
118 string output = "Teste.txt";
119 string alphabet = "abcdef";
120 vector<int> terminalStates;
121
122 struct transition{
123     int from;
124     int to;
125     char character;
126
127     friend bool operator== (transition E1, transition E2){
128         if(E1.from == E2.from && E1.character == E2.character) return true;
129         return false;
130     }
131     friend ostream &operator<<( ostream &output, const transition &E ) {
132          output << E.from << " " << E.character << " " << E.to;
133          return output;
134     }
135 };
136
137 vector<transition> transitions;
138 transition emptyTransition = {-1,-1,'0'};
139
140 bool isInVector(vector<auto> array, auto var){
141
142     for( auto element : array){
143         if(var == element) return true;
144     }
145     return false;
146 }
147
148 transition generateTransition(int states){
149     transition T;
150     T.from = rand()%states;
151     T.character = alphabet[rand()%alphabet.size()];
152     T.to = rand()%states;
153     while(isInVector(transitions, T)){
```

```
154         T.from = rand()%states;
155         T.character = alphabet[rand()%alphabet.size()];
156         T.to = rand()%states;
157     }
158     return T;
159 }
160 
161 transition searchTerminalState(int currentState){
162     transition validTransition = emptyTransition;
163     for( transition T: transitions){
164         if(T.from == currentState){
165             if(isInVector(terminalStates, T.to)) return T;
166             validTransition = T;
167         }
168     }
169     return validTransition;
170 }
171 
172 string searchNextCharacter(string expression, int *currentState, int expressionSize){
173 
174     for(int i = 0; i < expressionSize; i++){
175         for( transition T: transitions){
176             if(T.from == *currentState){
177                 *currentState = T.to;
178                 expression += T.character;
179                 return expression;
180             }
181         }
182     }
183     while(!isInVector(terminalStates, *currentState) && expression.size() < maxExpressionSize){
184         transition newTransition = searchTerminalState(*currentState);
185         if(newTransition == emptyTransition) return expression;
186         *currentState = newTransition.to;
187         expression += newTransition.character;
188     }
189     return expression;
190 }
191 
192 
193 string generateExpression(int currentState){
194 
195     string expression = "";
196     string newExp = "";
197     int expressionSize = minExpressionSize + rand()%(maxExpressionSize - minExpressionSize);
198     for(int i = 0; i < expressionSize; i++){
199         newExp = searchNextCharacter(expression, &currentState, expressionSize);
200         if(newExp == expression) return expression;
201         expression = newExp;
202         random_shuffle(transitions.begin(), transitions.end());
203     }
204     return expression;
205 }
206 
207 void generateTest(){
208     srand(clock());
209     int states = (minStates + rand()%(maxStates - minStates + 1));
210     fstream file;
211     file.open(output,ios::out);
212     file << states << endl;
213     file << alphabet.size() << " ";
214 
```

```
215     for (char character : alphabet) file << character << " ";
216     file << endl;
217     file << '1' << endl;
218 
219     int terminalStatesSize = 1 + rand()%(maxTerminalStates);
220     file << terminalStatesSize << " ";
221 
222     for(int i = 0; i < terminalStatesSize; i++){
223         int aux = rand()%(states);
224         if(isInVector(terminalStates,aux)){
225             i--;
226         }
227         else{
228             terminalStates.push_back(aux);
229             file << aux << " ";
230         }
231     }
232     file << endl;
233     int size = 0;
234     int qntTransitions = states*alphabet.size();
235     file << qntTransitions << endl;
236     for(int i = 0; i < states; i++){
237         for(int j = 0; j < alphabet.size(); j++){
238             transitions.push_back({i, rand()%states, alphabet[j]});
239             file << transitions[size] << endl;
240             size++;
241         }
242     }
243 
244 
245     file << maxTests << endl;
246     for(int i = 0 ; i < maxTests; i++){
247         file << generateExpression(0) << endl;
248     }
249 
250 
251 }
252 
253 int main(){
254     generateTest();
255 }
```

10.2 Apêndice 2

```
257 FLAGS=""
258 rm Teste.txt
259 g++ -o generator.exe -fconcepts testeGen.cpp
260 ./generator.exe
261 g++ trabalho1.c
262 ./a.out <Teste.txt
263 rm generator.exe
```