UNIVERSIDAD NACIONAL DE SAN AGUSTIN

ESCUELA PROFESIONAL DE
CIENCIA DE LA COMPUTACIÓN

COMPILADORES

Práctica Nro. 06

Alumno :
Diego A. Gutierrez
Profesor:
Velazco Paredes, Yuber

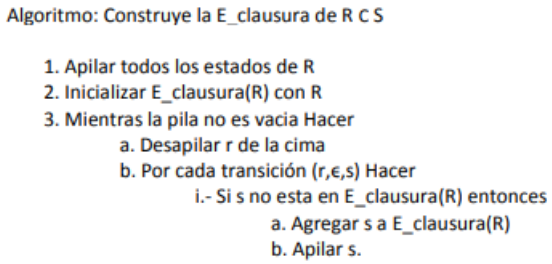
1 de julio de 2020

1. Objetivo

La presente práctica tiene como objetivo convertir Autómatas Finitos No Deterministas (AFN) a Autómatas finitos Deterministas (AFD).

2. Desarrollo

2.1. Construir el conjunto E_clausura de R C S. R es un subconjunto de S, y S es el conjunto de estados del autómata. En el apartado B) encontrará el formato del archivo de entrada.



2.1.1. Algoritmo desarrollado en clase

```
#include<iostream>
#include<vector>
#include <stack>
using namespace std;

bool verificar(vector<int>&v, int f){
    for (int i = 0; i < v.size(); i++)
    {
        if(v[i] == f){
            cout<<"repetido";
            return false;
        }

    }
    return true;
}
void claursura(){
```

```
18 int estados[13][3] = {{0,-1,1},{1,-1,2},{1,-1,4},{2,0,3},{4,1,5},{3,-1
19 int R[2] = {0,1};
20 vector<int> E_clausura;
21 stack<int> Pila;
22
23 //1. Apilar todos los estados de R
24 Pila.push(R[0]);
25 Pila.push(R[1]);
26
27
28 //2. Inicializar E_clausura(R) con R
29 E_clausura.push_back(R[0]);
30 E_clausura.push_back(R[1]);
31
32 //cout<<Pila.top();
33
34 //3. Mientras la pila no es vacia Hacer
35
36 while(!Pila.empty()){
37 //a. Desapilar r de la cima
38
39     int actual = Pila.top();
40     //cout<<actual<<endl;
41     Pila.pop();
42 //   . Por cada transici n (r,  ,s) Hacer
43     for (int i = 0; i < 13; i++)
44     {
45         if(estados[i][0] == actual){
46             if(estados[i][1] == -1 ){
47
48                 if(verificar(E_clausura,estados[i][2])){ // V
49                     cout<<"no se repite"<<endl;
50                     E_clausura.push_back(estados[i][2]);
51                     Pila.push(estados[i][2]);
52                 }
53             }
54         }
```

```
55      }
56  for (int i = 0; i < E_clausura.size(); i++)
57  {
58      cout<<E_clausura[i]<<',';
59  }
60
61  }
62  }
63
64
65  int main()
66  {
67      claursura();
68
69      return 0;
70  }
```

2.2. Desarrollar un programa que construya un AFD a partir de un AFN dado, mediante la construcción por subconjuntos.

Algoritmo: Construye D_est, el conjunto de estados de un AFD D, equivalente a un AFN N. También Construye D_tran, las funciones de transición del autómata D.

1. Inicializar D_est con E_clausura(s_0) y considerarlo no marcado.
2. Mientras exista un estado no marcado R en D_est Hacer
   a. Marcar R
   b. Por cada símbolo de entrada x, Hacer
      i. U ← E-clausura(Mover(R,x))
      ii. Si U no pertenece a D_est entonces
          Agregar U a D_est como un estado no marcado
      iii. D_tran(R,x) := U

2.2.1. Resolución

```
1  //COMPILADORES-CSUNSA
2  //DiegoGTZ
3  //AFND a FND
```

```
4  // Clausura E
5  #include<iostream>
6  #include<vector>
7  #include<string>
8  #include<tuple>
9  #include <algorithm>
10 #include <vector>
11 #include <tuple>
12 #include <fstream>
13 #include <list>
14 #include <map>
15 #include <stdlib.h>
16 using namespace std;
17 class Estado;
18
19 typedef tuple<Estado *,char, Estado *> Transicion;
20 typedef int IdEstado;
21 class Estado
22 {
23 public:
24     int                 id;
25     vector<Estado *>    subEstados;
26     vector<Transicion>  transiciones;
27     string              cadenaSubConjunto;
28 public:
29     Estado(int id){
30         this->id = id;
31     }
32     Estado(int id, vector<Estado*> subEstados){
33         this->id = id;
34         vector<int> subConjunto;
35         cadenaSubConjunto = "[";
36         for(Estado* estado: subEstados){
37             this->subEstados.push_back(estado);
38             subConjunto.push_back(estado->id);
39         }
40         sort(subConjunto.begin(),subConjunto.end());
```

```
41
42          for(int id :subConjunto ){
43              cadenaSubConjunto += to_string(id) + " ";
44          }
45          cadenaSubConjunto.push_back(']');
46      }
47  };
48
49  class Automata
50  {
51  public:
52      Estado *              Es_Inicial;
53      vector<Estado* >      Con_Estadodos;
54      vector<Estado*>       Es_aceptacion;
55      vector<int>           Entradas;
56      vector<Transicion>    Transiciones;
57
58  public:
59      Automata(){};
60      Automata(string _file){
61      ifstream file(_file.c_str());
62      string caracter = "";
63
64      int estado = 0 ;
65      while(file>>caracter){
66          if(estado == 0){
67              file>>caracter;
68              //cout<<caracter<<endl;
69              int m = stoi(caracter);
70              for (int i = 0; i < m; i++)
71              {
72                  file>>caracter;
73                  Con_Estadodos.push_back(new Estado(stoi(caracter)));
74              }
75              this->Es_Inicial = findEstado(0);
76              //cout<<"Estado inicial"<<Es_Inicial->id<<endl;
77              file>>caracter;
```

```
78              estado++;
79          }
80          else if(estado == 1){
81              //file>>caracter;
82              //cout<<caracter<<endl;
83              int num_Estados = stoi(caracter);
84              for(int i = 0; i < num_Estados; i++){
85                  file>>caracter;
86                  Es_aceptacion.push_back(new Estado(stoi(caracter)));
87              }
88              file>>caracter;
89              estado++;
90          }
91          else if(estado == 2 ){
92              int num_Esntradas = stoi(caracter);
93              for (int i = 0; i < num_Esntradas; i++)
94              {
95                  file>>caracter;
96
97                  int token = stoi(caracter);
98                  Entradas.push_back(token);
99              }
100             file>>caracter;
101             estado++;
102         }
103         else if(estado ==3){
104
105             int num_Transiciones = stoi(caracter);
106
107             for (int i = 0; i < num_Transiciones; i++)
108             {
109                 file>>caracter;
110                 int id_1 = stoi(caracter);
111                 Estado* estado1 = findEstado(id_1);
112
113                 file>>caracter;
114                 int trans = stoi(caracter);
```

```
115
116                 file>>caracter;
117
118                 int id_2 = stoi(caracter);
119                 Estado* estado2 = findEstado(id_2);
120
121                 //cout<< estado1->id<<" "<<trans<<" "<<estado2->id<<end
122
123                 Transiciones.push_back(make_tuple(estado1,trans,estado2
124                 estado1->transiciones.push_back(make_tuple(nullptr,tran
125             }
126         }
127
128     }
129
130     }
131     Estado* findEstado(int id){
132         for (Estado*res : Con_Estadodos)
133         {
134             if(res->id == id) return res;
135         }
136         return nullptr;
137     }
138
139     vector<Estado *> deleteRepeat(vector<Estado *> estadosV){
140         vector<Estado *> res;
141         vector<int> temp;
142         for(Estado * estado : estadosV){
143             temp.push_back(estado->id);
144         }
145         sort(temp.begin(), temp.end());
146         temp.erase(unique(temp.begin(),temp.end()),temp.end());
147         for(int id : temp){
148             res.push_back(findEstado(id));
149         }
150         return res;
151     }
```

```
152
153     vector<Estado* > E_clausura(Estado * estado){
154         map<int , bool> E_visitado;
155         list<Estado* > Pila;
156         vector<Estado*> token;
157         for(Estado * estado: Con_Estadodos)
158         {
159             E_visitado[estado->id] = false;
160         }
161         Estado * actual = nullptr;
162         Estado * estado1 = nullptr;
163         Estado * estado2 = nullptr;
164         int entrada;
165         token.push_back(estado);
166         Pila.push_front(estado);
167         while(!Pila.empty()){
168             actual = Pila.front();
169             Pila.pop_front();
170             E_visitado[actual->id] = true;
171             for(Transicion transicion : actual->transiciones){
172                 tie(estado1,entrada,estado2) = transicion;
173                 //cout<<"Entrada1 "<<entrada<<endl;
174                 if(entrada == -1 and !E_visitado[estado2->id]){
175                     Pila.push_front(estado2);
176                     token.push_back(estado2);
177                 }
178             }
179
180         }
181         return token;
182     }
183
184     vector<Estado *> E_clausura(vector<Estado*> _estados){
185         vector<Estado*> v;
186         for(Estado* estado : _estados){
187             vector<Estado*> temp = E_clausura(estado);
188             v.insert(v.begin(),temp.begin(),temp.end());
```

```
189         }
190         v = deleteRepeat(v);
191         return v;
192     }
193     vector<Estado* > findTransiciones(vector<Estado* > _estados, int tr
194         vector<Estado*> v;
195         Estado * r = nullptr;
196         Estado * s = nullptr;
197         int tran;
198         for(Estado* estado: _estados){
199             for (Transicion transicion : estado->transiciones)
200             {
201                 tie(r,tran,s) = transicion;
202                 if(tran == trans) v.push_back(s);
203             }
204         }
205         v= deleteRepeat(v);
206         return v;
207     }
208     Estado * findSub_Conjunto(vector<Estado* > subconjunto){
209         if(subconjunto.empty()){
210             return nullptr;
211         }
212         vector<int> subconjunto_ID;
213         string comp = "[";
214         for(Estado * estado : subconjunto){
215             subconjunto_ID.push_back(estado->id);
216         }
217         sort(subconjunto_ID.begin(),subconjunto_ID.end());
218         for(int id : subconjunto_ID){
219             comp = comp + to_string(id) + " ";
220         }
221
222         comp.push_back(']');
223         //cout<<"Comp"<<comp<<endl;
224         for(Estado* estado : Con_Estadodos){
225             if(!estado->cadenaSubConjunto.empty()){
```

```
226                 if(estado->cadenaSubConjunto == comp) return estado;
227             }
228         }
229         return nullptr;
230     }
231     bool Estado_Aceptacion(int id){
232         for(Estado* estado : Es_aceptacion){
233             if(estado->id == id)return true;
234         }
235         return false;
236     }
237 
238     void OutFileAutomta(ostream & file , bool run){
239         file<<"Estados"<<endl;
240         file<<Con_Estadodos.size()<<endl;
241         for(Estado* estado : Con_Estadodos){
242             if(run){
243                 file<<estado->id<<" "<<estado->cadenaSubConjunto<<endl;
244             }
245             else{
246                 file<<estado->id<<" ";
247             }
248         }
249         if(!run){file<<endl;}
250         file<<"Inicial"<<endl;
251         file<<Es_Inicial->id<<endl;
252         file<<"Aceptacion"<<endl;
253         file<<Es_aceptacion.size()<<endl;
254         for(Estado*estado : Es_aceptacion){file<<estado->id<<" ";}
255         file<<endl;
256         file<<"Entradas"<<endl;
257         file<<Entradas.size()<<endl;
258         for(int c : Entradas){file<<c<<" ";}
259         file<<endl;
260         file<<"Transiciones"<<endl;
261         file<<Transiciones.size()<<endl;
262         Estado* r = nullptr;
```

```
263         Estado* s = nullptr;
264         int c;
265         for(Transicion  trans : Transiciones){
266             tie(r,c,s) = trans;
267             file<<r->id<<" "<<c<<" "<<s->id<<endl;
268         }
269     }
270     void Graficar(string tipo)
271     {
272         string name = tipo;
273         ofstream file(tipo +".dot");
274         file<<"digraph {\n";
275         Estado* r = nullptr;
276         Estado* s = nullptr;
277         int c1;
278         file<<"rankdir = LR;"<<endl;
279         file<<"size = "<<'"'<<8<<',' <<'5'<<'"'<<endl;
280 
281         file<<"node [shape = doublecircle];"<<Es_aceptacion[0]->id<<';'
282         file<<"node [shape = point];" <<'i'<< ';'<<endl;
283         file<<"node [shape = circle];"<<endl;
284         file<<"i->"<<Es_Inicial->id<<';'<<endl;
285 
286 
287         for(Transicion  trans : Transiciones){
288             tie(r,c1,s) = trans;
289             file<<r->id<<"->"<<s->id<<"[     label = "<<c1<<"]"<<";\n";
290 
291         }
292         file<<"}";
293         file.close();
294         string comando = "dot -Tjpg " + name + ".dot -o" + name + ".jpg
295         const char * c  =  comando.c_str();
296         system(c);
297 
298     }
299 
```

```
300 };
301 class AFND_to_AFD
302 {
303 public:
304     int           tmp_Estado;
305     Automata      _ANFD;
306     Automata      AFD;
307 public:
308     AFND_to_AFD(string inFile, string outFile){
309         Automata ANFD(inFile);
310         this->_ANFD = ANFD;
311         tmp_Estado = 0;
312         list<Estado* > Pila;
313         //cout<<"es inicial"<<ANFD.Es_Inicial->id<<endl;
314         this->AFD.Es_Inicial = new Estado(tmp_Estado, ANFD.E_clausura({
315         //cout<<"Estado inicial de AFD" << AFD.Es_Inicial->cadenaSubCon
316         this->AFD.Con_Estadodos.push_back(this->AFD.Es_Inicial);
317         this->AFD.Entradas = ANFD.Entradas;
318         //cout<<AFD.Entradas.size()<<endl;
319         Pila.push_front(AFD.Con_Estadodos.back());
320         Estado * actual = nullptr;
321         Estado * temp = nullptr;
322
323         tmp_Estado++;
324
325         while(!Pila.empty()){
326             actual = Pila.front();
327             //cout<<"Actual"<<actual->id<<endl;
328             Pila.pop_front();
329
330             for(int entrada : ANFD.Entradas){
331                 vector<Estado* > subConjunto = ANFD.E_clausura(ANFD.fin
332                 //cout<<subConjunto[2]->id<<endl;
333                 if(!subConjunto.empty()){
334                     temp = AFD.findSub_Conjunto(subConjunto);
335                     //cout<<temp<<endl;
336                     if(temp == nullptr){
```

```
337                             temp = new Estado(tmp_Estado,subConjunto);
338                             tmp_Estado++;
339                             AFD.Con_Estadodos.push_back(temp);
340                             Pila.push_front(temp);
341                         }
342                         AFD.Transiciones.push_back(make_tuple(actual,entrad
343                         actual->transiciones.push_back(make_tuple(nullptr,e
344                     }
345                 }
346             }
347             Acc_E_Aceptacion();
348             ofstream out(outFile.c_str());
349             AFD.OutFileAutomta(out,true);
350             AFD.Graficar("AFD");
351             _ANFD.Graficar("AFND");
352         }
353         void Acc_E_Aceptacion(){
354             for(Estado * estado : this->AFD.Con_Estadodos ){
355                 for(Estado * conj_estados : estado->subEstados){
356                     if(_ANFD.Estado_Aceptacion(conj_estados->id)){
357                         AFD.Es_aceptacion.push_back(estado);
358                         break;
359                     }
360                 }
361             }
362 
363         }
364 
365 };
366 
367 int main()
368 {
369     string          inFile = "in.txt";
370     string          outFile = "OUT.txt";
371     AFND_to_AFD     prueba(inFile,outFile);
372     return 0;
373 
```

```
374 }
```

Como se puede observar en nuestro código presentado arriba nuestro codigo cuenta con la siguiente estructura :

2.2.2. Resultado

Para evaluar los resultados obtenidos usaremos un archivo de entrada llamado in.txt propuesto para esta practica.

```
1  Estados
2  11
3  0 1 2 3 4 5 6 7 8 9 10
4  Estados_de_Aceptacion
5  1
6  10
7  Entradas
8  2
9  0 1
10 Transiciones
11 13
12 0 -1 1
13 1 -1 2
14 1 -1 4
15 2 0 3
16 4 1 5
17 3 -1 6
18 5 -1 6
19 6 -1 7
20 6 -1 1
21 0 -1 7
22 7 0 8
23 8 1 9
24 9 1 10
```

- En nuestra función main() creamos nuestro prueba de tipo autómata el cual va a recibir dos parámetros el archivo con los datos de nuestro autómata no determinista que se encuentra en nuestro archivo int.txt propuesto anteriormente.

Y el nombre del archivo de salida donde obtendremos nuestros resultados del nuevo AFD.

```
int main()
{
    string          inFile = "in.txt";
    string          outFile = "OUT.txt";
    AFND_to_AFD     prueba(inFile,outFile);
    return 0;

}
```

- Corremos nuestro código.

```
C:\Users\PCO\Desktop\COMPILADORES6
λ g++ AFND-AFD.cpp -o  AFND-AFD.exe

C:\Users\PCO\Desktop\COMPILADORES6
λ AFND-AFD.exe
```

- Obteniendo como resultado nuestro archivo OUT.txt.

```
1  Estados
2  5
3  0 [0 1 2 4 7 ]
4  1 [1 2 3 4 6 7 8 ]
5  2 [1 2 4 5 6 7 ]
6  3 [1 2 4 5 6 7 9 ]
7  4 [1 2 4 5 6 7 10 ]
8  Inicial
9  0
10 Aceptacion
11 1
12 4
13 Entradas
14 2
15 0 1
16 Transiciones
17 10
18 0 0 1
19 0 1 2
20 2 0 1
```

21	2	1	2
22	1	0	1
23	1	1	3
24	3	0	1
25	3	1	4
26	4	0	1
27	4	1	2

- Para una mejor visualización hicimos uso de la librería graphviz para poder graficar nuestro nuevo AFD obtenido a partir del AFND propuesto.

```
Acc_E_Aceptacion();
ofstream out(outFile.c_str());
AFD.OutFileAutomta(out,true);
AFD.Graficar("AFD");
_ANFD.Graficar("AFND");
```

Figura 1: Llamado a la función Graficar()

```
}
void Graficar(string tipo)
{
    string name = tipo;
    ofstream file(tipo +".dot");
    file<<"digraph {\n";
    Estado* r = nullptr;
    Estado* s = nullptr;
    int c1;
    file<<"rankdir = LR;"<<endl;
    file<<"size = "<<'"'<<8<<',' <<'5'<<'"'<<endl;

    file<<"node [shape = doublecircle];"<<Es_aceptacion[0]->id<<';'<<endl;
    file<<"node [shape = point];" <<'i'<< ';'<<endl;
    file<<"node [shape = circle];"<<endl;
    file<<"i->"<<Es_Inicial->id<<';'<<endl;


    for(Transicion  trans : Transiciones){
        tie(r,c1,s) = trans;
        file<<r->id<<"->"<<s->id<<"[    label = "<<c1<<"]"<<";\n";

    }
    file<<"}";
    file.close();
    string comando = "dot -Tjpg " + name + ".dot -o" + name + ".jpg";
    const char * c  =  comando.c_str();
    system(c);

}
```

Figura 2: Función Graficar()

Los Codigos mostrados en la Figura 1 y Figura 2 son parte de nuestra clase Automata.

- El cual grafica a partir de un archivo que generamos con el código anterior.El archivo generado es AFD.dot y se muestra a continuacion.

```
digraph {
rankdir = LR;
size = "8,5"
node [shape = doublecircle];4;
node [shape = point];i;
node [shape = circle];
i->0;
0->1[   label = 0];
0->2[   label = 1];
2->1[   label = 0];
2->2[   label = 1];
1->1[   label = 0];
1->3[   label = 1];
3->1[   label = 0];
3->4[   label = 1];
4->1[   label = 0];
4->2[   label = 1];
}
```

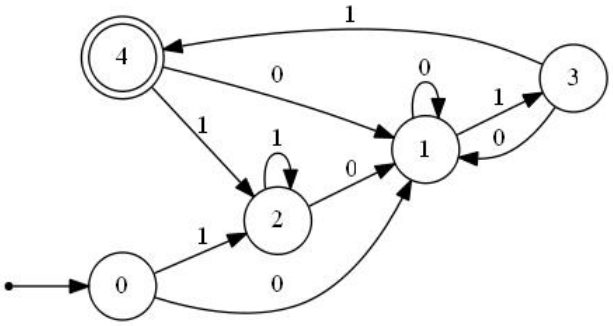


Figura 3: Grafica de nuestro AFD obtenido,haciendo uso de la libreria Graphviz

- También podemos graficar el AFND propuesto.

```
digraph {
rankdir = LR;
size = "8,5"
node [shape = doublecircle];10;
node [shape = point];i;
node [shape = circle];
```

```
7  i->0;
8  0->1[    label = -1];
9  1->2[    label = -1];
10 1->4[    label = -1];
11 2->3[    label = 0];
12 4->5[    label = 1];
13 3->6[    label = -1];
14 5->6[    label = -1];
15 6->7[    label = -1];
16 6->1[    label = -1];
17 0->7[    label = -1];
18 7->8[    label = 0];
19 8->9[    label = 1];
20 9->10[   label = 1];
21 }
```

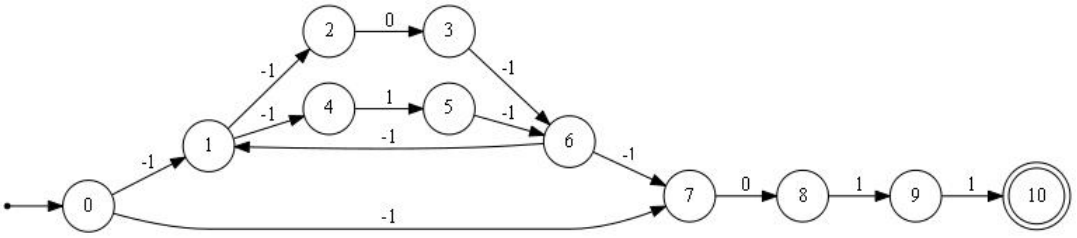


Figura 4: Grafica del AFND propuesto, haciendo uso de la libreria Graphviz

3. Enlace GitHub

- `https://github.com/DiegoGtz/Compiladores/tree/master/Laboratorio6`

Referencias

[1] Velazco Paredes Yuber, "*Curso de Compiladores - UNSA*", 2020