

Manual de Usuario

PROYECTO 3

Mauricio G. Coello | A01328258 | 21 de junio del 2016

Introducción (FA)

Su fin es el estudio de dispositivos computacionales abstractos, originalmente empleado para el modelado del funcionamiento del cerebro, y es de gran utilidad en gran número de problemas.

Manual de Usuario

En este caso, se utilizó C++ como lenguaje para desarrollar esta utilidad, por lo que para su ejecución se requiere de un compilador tipo MinGW.

Para generar el ejecutable del código, abra la terminal de su ordenador, diríjase al directorio donde tiene el archivo .cpp. Una vez ahí, para compilar ejecute la siguiente línea de comando:

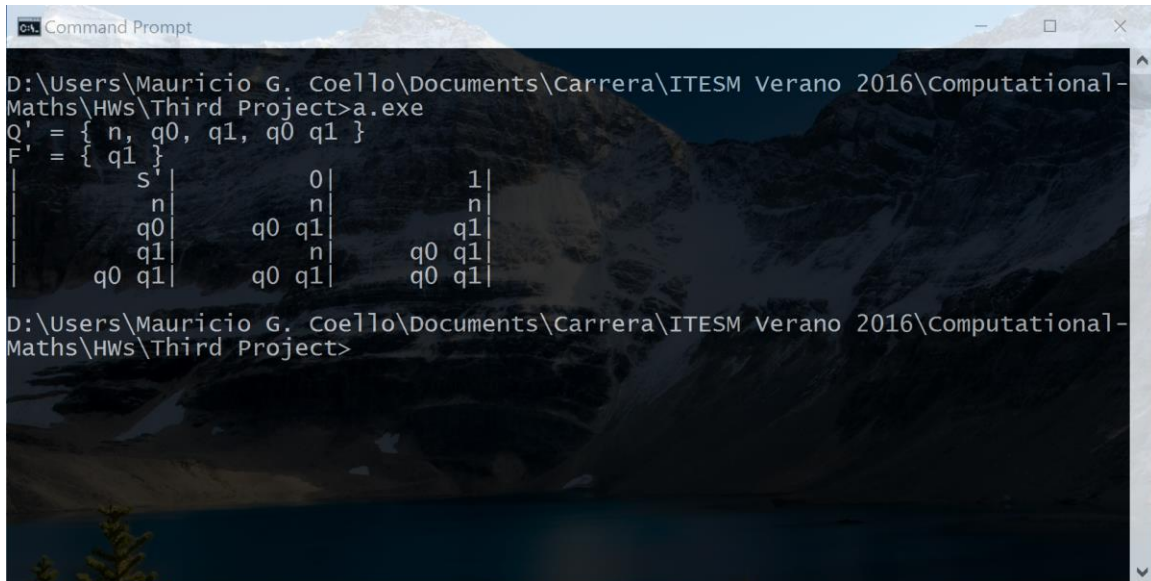
```
g++ -std=c++11 "Nombre_del_archivo".cpp
```

Para ejecutar el programa, es necesario tener un archivo .txt con el nombre de archivo "ex.txt" con el siguiente formato:

```
M =({ q0 q1 } { 0 1 } S q0 { q1 })  
q0 q0-q1 q1  
q1 / q0-q1
```

- M siendo la definición del autómata
- Los siguientes renglones son la tabla de transiciones, dada el primer estado del renglón el estado actual y los siguientes los que toma en orden ascendente con respecto a la entrada, definida en las segundas llaves de M
- Cuando son 2 estados en uno, se separan con - y los estados nulos se definen con /

Ejemplo de funcionalidad



```
Command Prompt
D:\Users\Mauricio G. Coello\Documents\Carrera\ITESM Verano 2016\Computational-
Maths\Hws\Third Project>a.exe
Q' = { n, q0, q1, q0 q1 }
F' = { q1 }
|   s' |           0 |           1 |
|   n  |           n |           n |
|   q0 |        q0 q1 |        q1 |
|   q1 |           n |        q0 q1 |
|   q0 q1 |        q0 q1 |        q0 q1 |

D:\Users\Mauricio G. Coello\Documents\Carrera\ITESM Verano 2016\Computational-
Maths\Hws\Third Project>
```

Descripción técnica

La implementación en código implica varias funciones, que separan el proceso iterativo para la generación del DFA

```
std::vector<std::vector<std::string> > combination (std::vector<std::string> states)
{
    //std::cout<<states.size();
    std::string tp;
    std::vector<std::string> tmp;
    std::vector<std::vector<std::string> > rtn;
    rtn.push_back( tmp );

    for (int i = 0; i < states.size(); i++)
    {
        std::vector< std::vector<std::string> > rtnTemp = rtn;

        for (int j = 0; j < rtnTemp.size(); j++)
            rtnTemp[j].push_back( states[i] );

        for (int j = 0; j < rtnTemp.size(); j++)
            rtn.push_back( rtnTemp[j] );
    }
    return rtn;
}
```

En la función anterior, se generan todos los estados que tendrá el DFA, utilizando 3 fors para la generación de todos los posibles subconjuntos de los estados totales del NFA dado.

```

std::vector<std::vector<std::string> > FillNewTable(std::vector<std::vector<std::string> > table, std::vector<std::vector<std::string> > ntable, std::vector<int> n)
{
    std::string tmp, var, fn;
    int x;
    for (int i = 0; i < 1; i++)
    {
        for (int j = 0; j < n.size(); j++)
        {
            ntable[i].push_back("/");
        }
    }
    for (int i = 1; i < ntable.size(); i++)
    {
        for (int j = 0; j < table.size(); j++)
        {
            if(j==0)
            {
                for (int k = 1; k <= n.size(); k++)
                {
                    ntable[i].push_back("");
                }
            }
            if(ntable[i][0].find("-"))
            {
                var=ntable[i][0];
                std::istringstream ss(var);
                while(std::getline(ss, tmp, "-"))
                {
                    x=findPos(tmp, table);
                    for (int k = 1; k < table[x].size(); k++)
                    {
                        if(k<table[x].size()-1)
                            insert(table[x][k], ntable[i][k], true);
                        else
                            insert(table[x][k], ntable[i][k], false);
                    }
                }
            }
            else
            {
                if(ntable[i][0]==table[j][0])
                {
                    for (int k = 1; k < table[j].size(); k++)
                    {
                        insert(table[j][k], ntable[i][k], false);
                    }
                }
            }
        }
    }
    return ntable;
}

```

En la función anterior, se buscan los estados que tomarían los nuevos estados generados en la función anterior, en base a las entradas definidas.

```

void printF(std::vector<std::vector<std::string> >ntable, std::vector<std::string> final)
{
    std::cout<<"F' = { ";
    for (int i = 0; i < ntable.size(); i++)
    {
        for (int j = 1; j < ntable[i].size(); j++)
        {
            for (int k = 0; k < final.size(); k++)
            {
                if(ntable[i][j].find(final[k])==std::string::npos)
                {
                    if(ntable[i][0]!="n")
                        std::cout<<ntable[i][0];
                    k=final.size();
                    j=ntable[i].size();
                }
            }
        }
    }
    std::cout<<" }"<<std::endl;
}

```

En esta función se detectan todos los subconjuntos que son finales, si estos contienen al menos un estado inicial.

Bibliografía

- Presentaciones de clase, Dr. Víctor de la Cueva, 2016