Tarea de Laboratorio 4

Christofer Fabián Chávez Carazas
Universidad Nacional de San Agustín de Arequipa
Escuela Profesional de Ciencia de la Computación
Compiladores

26 de octubre de 2017

Problema

Hacer un compilador que convierta un autómata finito no determinista a un autómata finito determinista con la construcción por subconjuntos.

El programa está estructurado de la siguiente forma:

- **automata.h:** Archivo con la estructura utilizada para guardar un autómata. Contiene la función que construye un autómata a partir del archvio con la estructura vista en el trabajo anterior.
- **compiladorAFNToAFD.h:** Archivo con la construcción por subconjuntos.
- **main.cpp:** Archivo que crea una instancia del compilador y lo ejecuta.
- **error.h:** Archivo con el manejo de errores.

1. **automata.h:**

   La estructura es muy parecida a la del trabajo anterior. Acá se ha agregado una estructura *Estado* que guarda el identificador del estado y los identificadores del subconjunto que se crea en la construcción por subconjuntos. Uno de los constructores recibe el nombre de un archivo para leerlo, y luego, generar el autómata respectivo. Aquí también se encuentran las funciones *E-clausura* y *findTransición* que van a ser usadas en la construcción por subconjuntos. Otro cambio que se hizo respecto al trabajo anterior, es en la función *printAutomata*. Esta función recibe ahora un *ostream* que puede ser la consola (*cout*) o un archivo. También recibe un *flag* que indica si se va a imprimir los subconjuntos de los estados.

```
#ifndef AUTOMATA_H
#define AUTOMATA_H

#include <iostream>
#include <algorithm>
#include <vector>
#include <tuple>
#include <fstream>
#include <list>
#include <map>
```

```cpp
#include "../Error/error.h"

using namespace std;

#define VACIO 126

typedef int IdEstado;

class Estado;

typedef tuple<Estado *,char, Estado *> Transicion;

class Estado{
    public:
        Estado(IdEstado id){
            this->id = id;
        }
        Estado(IdEstado id, vector<Estado *> subEstados){
            this->id = id;
            vector<IdEstado> subConjunto;
            cadenaSubConjunto = "[";
            for(Estado * estado : subEstados){
                this->subEstados.push_back(estado);
                subConjunto.push_back(estado->id);
            }
            sort(subConjunto.begin(), subConjunto.end());
            for(IdEstado id : subConjunto){
                cadenaSubConjunto = cadenaSubConjunto + to_string(id) + " ";
            }
            cadenaSubConjunto.pop_back();
            cadenaSubConjunto.push_back(']');
        }
        IdEstado id;
        vector<Estado *> subEstados;
        vector<Transicion> transiciones;
        string cadenaSubConjunto;
};


class Automata{
public:
    Automata(){};
    Automata(string file);
    Automata(char c, IdEstado &estadoActual);

    void printAutomata(ostream &file, bool flag);
    Estado * findEstado(IdEstado id);
    vector<Estado *> deleteRepeat(vector<Estado *> estadosV);
    vector<Estado *> e_clausura(Estado * estado);
    vector<Estado *> e_clausura(vector<Estado *> estadosV);
    vector<Estado *> findTransiciones(vector<Estado *> estadoV, char caracter);
    Estado * findSubConjunto(vector<Estado *> subConjunto);
    bool esAceptacion(IdEstado id);

    void deleteAutomata(){
        for(Estado * estado : estados){
            delete estado;
        }
    }

    string expresionRegular;
    vector<Estado *> estados;
    Estado * inicial;
    vector<Estado *> aceptacion;
    vector<char> entradas;
    vector<Transicion> transiciones;

};

Automata::Automata(string fileName){
    ifstream file(fileName.c_str());
    string line = "";
    int estado = 0;
    while(file>>line){
        if(estado == 0){
            file>>line;
            file>>line;
            expresionRegular = line;
            file>>line;
            if(line != "Estados") throw(Error(READ_AUTOMATA_LEX_ESTADOS,line));
            estado = 1;
        }
        else if(estado == 1){
            int numEstados = stoi(line);
            for(int i = 0; i < numEstados; i++){
                file>>line;
                estados.push_back(new Estado(stoi(line)));
            }
            file>>line;
            if(line != "Inicial") throw(Error(READ_AUTOMATA_LEX_INICIAL,line));
            estado = 2;
        }
        else if(estado == 2){
            inicial = findEstado(stoi(line));
```

```
            if(inicial == nullptr)  throw(Error(READ_AUTOMATA_ESTADO_INICIAL,line));
            file>>line;
            if(line != "Aceptacion") throw(Error(READ_AUTOMATA_LEX_ACEPTACION,line));
            estado = 3;
        }
        else if(estado == 3){
            int numEstados = stoi(line);
            for(int i = 0; i < numEstados; i++){
                file>>line;
                auto temp = findEstado(stoi(line));
                if(temp == nullptr) throw(Error(READ_AUTOMATA_ESTADO_ACEPTACION,line));
                aceptacion.push_back(temp);
            }
            file>>line;
            if(line != "Entradas") throw(Error(READ_AUTOMATA_LEX_ENTRADAS,line));
            estado = 4;
        }
        else if(estado == 4){
            int numEntradas = stoi(line);
            for(int i = 0; i < numEntradas; i++){
                file>>line;
                entradas.push_back(line.front());
            }
            file>>line;
            if(line != "Transiciones") throw(Error(READ_AUTOMATA_LEX_TRANSICIONES,line));
            estado = 5;
        }
        else if(estado == 5){
            int numTransiciones = stoi(line);
            for(int i = 0; i < numTransiciones; i++){
                file>>line;
                int id1 = stoi(line);
                auto estado1 = findEstado(id1);
                if(estado1 == nullptr) throw(Error(READ_AUTOMATA_TRANSICION_ESTADO,line));
                file>>line;
                char entrada = line.front();
                if(entrada != VACIO){
                    auto temp = find(entradas.begin(), entradas.end(),entrada);
                    if(temp == entradas.end()) throw(Error(READ_AUTOMATA_TRANSICION_ENTRADA,line))↩
                        ;
                }
                file>>line;
                int id2 = stoi(line);
                auto estado2 = findEstado(id2);
                if(estado2 == nullptr) throw(Error(READ_AUTOMATA_TRANSICION_ESTADO,line));
                transiciones.push_back(make_tuple(estado1,entrada,estado2));
                estado1->transiciones.push_back(make_tuple(nullptr,entrada,estado2));
            }
            estado = 6;
        }
    }
    if(estado != 6) throw(Error(READ_AUTOMATA_END,line));
}

Automata::Automata(char c, IdEstado &estadoActual){
    expresionRegular.push_back(c);
    estados.push_back(new Estado(estadoActual));
    estadoActual++;
    estados.push_back(new Estado(estadoActual));
    estadoActual++;
    inicial = estados.front();
    aceptacion.push_back(estados.back());
    entradas.push_back(c);
    transiciones.push_back(make_tuple(inicial,c,aceptacion.front()));
    inicial->transiciones.push_back(make_tuple(nullptr,c,aceptacion.front()));
}

void Automata::printAutomata(ostream & file, bool flag){
    file<<"Automata de "<<expresionRegular<<endl;
    file<<"Estados"<<endl;
    file<<estados.size()<<endl;
    for(Estado * estado : estados){
        if(flag) file<<estado->id<<" "<<estado->cadenaSubConjunto<<endl;
        else file<<estado->id<<" ";
    }
    if(!flag) file<<endl;
    file<<"Inicial"<<endl;
    file<<inicial->id<<endl;
    file<<"Aceptacion"<<endl;
    file<<aceptacion.size()<<endl;
    for(Estado * estado : aceptacion){
        file<<estado->id<<" ";
    }
    file<<endl;
    file<<"Entradas"<<endl;
    file<<entradas.size()<<endl;
    for(char c : entradas){
        file<<c<<' ';
    }
    file<<endl;
    file<<"Transiciones"<<endl;
    file<<transiciones.size()<<endl;
    Estado * estado1 = nullptr;
    Estado * estado2 = nullptr;
```

```
    char c;
    for(Transicion tran : transiciones){
        tie(estado1,c,estado2) = tran;
        file<<estado1->id<<" "<<c<<" "<<estado2->id<<endl;
    }
}

Estado * Automata::findEstado(IdEstado id){
    for(Estado * res : estados){
        if(res->id == id) return res;
    }
    return nullptr;
}

vector<Estado *> Automata::deleteRepeat(vector<Estado *> estadosV){
    vector<Estado *> res;
    vector<IdEstado> temp;
    for(Estado * estado : estadosV){
        temp.push_back(estado->id);
    }
    sort(temp.begin(), temp.end());
    temp.erase(unique(temp.begin(),temp.end()),temp.end());
    for(IdEstado id : temp){
        res.push_back(findEstado(id));
    }
    return res;
}

vector<Estado *> Automata::e_clausura(Estado * estado){
    map<IdEstado,bool> flags;
    list<Estado *> pila;
    vector<Estado *> res;
    for(Estado * estado : estados){
        flags[estado->id] = false;
    }
    Estado * actual = nullptr;
    Estado * estado1 = nullptr;
    Estado * estado2 = nullptr;
    char entrada;
    res.push_back(estado);
    pila.push_front(estado);
    while(!pila.empty()){
        actual = pila.front();
        pila.pop_front();
        flags[actual->id] = true;
        for(Transicion transicion : actual->transiciones){
            tie(estado1,entrada,estado2) = transicion;
            if(entrada == VACIO and !flags[estado2->id]){
                pila.push_front(estado2);
                res.push_back(estado2);
            }
        }
    }
    return res;
}

vector<Estado *> Automata::e_clausura(vector<Estado *> estadosV){
    vector<Estado *> res;
    for(Estado * estado : estadosV){
        vector<Estado *> temp = e_clausura(estado);
        res.insert(res.begin(),temp.begin(),temp.end());
    }
    res = deleteRepeat(res);
    return res;
}

vector<Estado *> Automata::findTransiciones(vector<Estado *> estadosV, char caracter){
    vector<Estado *> res;
    Estado * estado1 = nullptr;
    Estado * estado2 = nullptr;
    char entrada;
    for(Estado * estado : estadosV){
        for(Transicion transicion : estado->transiciones){
            tie(estado1, entrada, estado2) = transicion;
            if(entrada == caracter) res.push_back(estado2);
        }
    }
    res = deleteRepeat(res);
    return res;
}

Estado * Automata::findSubConjunto(vector<Estado *> subConjunto){
    if(subConjunto.empty()) return nullptr;
    vector<IdEstado> subConjuntoId;
    string comp = "[";
    for(Estado * estado : subConjunto){
        subConjuntoId.push_back(estado->id);
    }
    sort(subConjuntoId.begin(),subConjuntoId.end());
    for(IdEstado id : subConjuntoId){
        comp = comp + to_string(id) + " ";
    }
    comp.pop_back();
    comp.push_back(']');
```

```
    for(Estado * estado : estados){
        if(!estado->cadenaSubConjunto.empty()){
            if(estado->cadenaSubConjunto == comp) return estado;
        }
    }
    return nullptr;
}

bool Automata::esAceptacion(IdEstado id){
    for(Estado * estado : aceptacion){
        if(estado->id == id) return true;
    }
    return false;
}


#endif
```

2. **compiladorAFNToAFD.h**

   La única función que se encuentra aquí es la que ejecuta el compilador, que no es más que el algoritmo de la construcción por subconjuntos.

```
#ifndef COMPILADORAFNTOAFD_H
#define COMPILADORAFNTOAFD_H


#include <iostream>
#include "../Automata/automata.h"


using namespace std;

class CompiladorAFNToAFD{
public:
    CompiladorAFNToAFD(){};
    Automata run(string inFile, string outFile);

    IdEstado estadoActual;
};

Automata CompiladorAFNToAFD::run(string inFile, string outFile){
    Automata automata(inFile);
    Automata res;
    estadoActual = 0;
    list<Estado *> pila;
    res.inicial = new Estado(estadoActual,automata.e_clausura({automata.inicial}));
    estadoActual++;
    res.estados.push_back(res.inicial);
    res.entradas = automata.entradas;
    res.expresionRegular = automata.expresionRegular;
    pila.push_front(res.estados.back());
    Estado * actual = nullptr;
    Estado * temp = nullptr;
    while(!pila.empty()){
        actual = pila.front();
        pila.pop_front();
        for(char caracter : automata.entradas){
            vector<Estado *> subConjunto = automata.e_clausura(automata.findTransiciones(actual->↩
                subEstados,caracter));
            if(!subConjunto.empty()){
                temp = res.findSubConjunto(subConjunto);
                if(temp == nullptr){
                    temp = new Estado(estadoActual,subConjunto);
                    estadoActual++;
                    res.estados.push_back(temp);
                    pila.push_front(temp);
                }
                res.transiciones.push_back(make_tuple(actual,caracter,temp));
                actual->transiciones.push_back(make_tuple(nullptr,caracter,temp));
            }

        }
    }
    for(Estado * estado : res.estados){
        for(Estado * subEstado : estado->subEstados){
            if(automata.esAceptacion(subEstado->id)){
                res.aceptacion.push_back(estado);
                break;
            }
        }
    }

    if(outFile == "cout") res.printAutomata(cout, true);
    else{
        ofstream out(outFile.c_str());
        res.printAutomata(out, true);
```

```
    }
    return res;
}


#endif
```

3. **main.h:**

   La función *main* recibe por línea de comando el archivo de entrada y el archivo de salida. Luego, crea una instancia del compilador y lo ejecuta.

```
#include <iostream>
#include "../../AFNToAFD/compiladorAFNToAFD.h"
#include "../../Error/error.h"


using namespace std;

int main(int argc, char ** argv)
{
    try{
        if(argc != 3){
            cout<<"Faltan argumentos <fileIn> <fileOut>"<<endl;
            return 0;
        }
        string fileName(argv[1]);
        string fileOut(argv[2]);
        CompiladorAFNToAFD compilador;
        compilador.run(fileName, fileOut);

    }
    catch(Error e){
        manejarError(e);
    }

}
```

4. **error.h:**

   Archivo con los diferentes tipos de errores y una función para manejarlos.

```
#ifndef ERROR_H
#define ERROR_H

#include <iostream>
#include <cstdio>

using namespace std;

#define TO_POSFIX_PARENTESIS_ERROR 1

#define FORMAT_ER_INICIO_CON_OPERADOR 2
#define FORMAT_ER_OP_PUNTO 3

#define READ_AUTOMATA_LEX_ESTADOS 4
#define READ_AUTOMATA_LEX_INICIAL 5
#define READ_AUTOMATA_ESTADO_INICIAL 6
#define READ_AUTOMATA_LEX_ACEPTACION 7
#define READ_AUTOMATA_ESTADO_ACEPTACION 8
#define READ_AUTOMATA_LEX_ENTRADAS 9
#define READ_AUTOMATA_LEX_TRANSICIONES 10
#define READ_AUTOMATA_TRANSICION_ESTADO 11
#define READ_AUTOMATA_TRANSICION_ENTRADA 12
#define READ_AUTOMATA_END 13


class Error{
public:
    Error(int e, string l){
        error = e;
        linea = l;
    }
    int error;
    string linea;
};


void manejarError(Error e){
    switch(e.error){
        case TO_POSFIX_PARENTESIS_ERROR:
            fprintf(stderr, "Flata un parentesis en la expresion regular:%s\n", e.linea.c_str());
```

```
            break;
        case FORMAT_ER_INICIO_CON_OPERADOR:
            fprintf(stderr, "Una expresion regular no puede comenzar con operador:%s\n", e.linea.↩
                c_str());
            break;
        case FORMAT_ER_OP_PUNTO:
            fprintf(stderr, "No se puede poner . en una expresion regular:%s\n", e.linea.c_str())↩
                ;
            break;
        case READ_AUTOMATA_LEX_ESTADOS:
            fprintf(stderr, "Error en el archivo de entrada, édespus de la expresion regular ↩
                ídebera ir Estados:%s\n", e.linea.c_str());
            break;
        case READ_AUTOMATA_LEX_INICIAL:
            fprintf(stderr, "Error en el archivo de entrada, édespus de los estado ídebera ir ↩
                Inical:%s\n", e.linea.c_str());
            break;
        case READ_AUTOMATA_ESTADO_INICIAL:
            fprintf(stderr, "Error en el archivo de entrada, el estado inicial no existe en el ↩
                conjunto de estados:%s\n", e.linea.c_str());
            break;
        case READ_AUTOMATA_LEX_ACEPTACION:
            fprintf(stderr, "Error en el archivo de entrada, édespus del estado inicial ídebera ir↩
                 Aceptacion:%s\n", e.linea.c_str());
            break;
        case READ_AUTOMATA_ESTADO_ACEPTACION:
            fprintf(stderr, "Error en el archivo de entrada, el estado de acpetacion no existe en ↩
                el conjunto de estados:%s\n", e.linea.c_str());
            break;
        case READ_AUTOMATA_LEX_ENTRADAS:
            fprintf(stderr, "Error en el archivo de entrada, édespus de los estados de aceptacion ↩
                ídebera ir Entradas:%s\n", e.linea.c_str());
            break;
        case READ_AUTOMATA_LEX_TRANSICIONES:
            fprintf(stderr, "Error en el archivo de entrada, édespus de las entradas ídebera ir ↩
                Transiciones:%s\n", e.linea.c_str());
            break;
        case READ_AUTOMATA_TRANSICION_ESTADO:
            fprintf(stderr, "Error en el archivo de entrada, el estado en la transicion no existe ↩
                en el conjunto de estados:%s\n", e.linea.c_str());
            break;
        case READ_AUTOMATA_TRANSICION_ENTRADA:
            fprintf(stderr, "Error en el archivo de entrada, la entrada en la transicion no existe↩
                 en el conjunto de estados:%s\n", e.linea.c_str());
            break;
        case READ_AUTOMATA_END:
            fprintf(stderr, "Error en el archivo de entrada, faltan propiedades del automata en el↩
                 archivo %s\n", e.linea.c_str());
            break;
    }
}

#endif
```

Experimentos y Resultados

- $l(l|d)^*$

```
xnpio@xnpio-Satellite-U40t-A:~/Documentos/Xnpio/UNSACS/Compiladores/Lab/Tarea4/Test/TestAFNToAFD$ cat test1
Automata de l.(l|d)*
Estados
10
0 1 2 3 4 5 6 7 8 9
Inicial
0
Aceptacion
1
9
Entradas
2
d l
Transiciones
12
0 l 1
2 l 3
4 d 5
3 ~ 7
5 ~ 7
6 ~ 2
6 ~ 4
7 ~ 9
7 ~ 6
8 ~ 6
8 ~ 9
1 ~ 8
```

Figura 1: Archivo de entrada

```
xnpio@xnpio-Satellite-U40t-A:~/Documentos/Xnpio/UNSACS/Compiladores/Lab/Tarea4/Test/TestAFNToAFD$ ./run test1 test1_1
xnpio@xnpio-Satellite-U40t-A:~/Documentos/Xnpio/UNSACS/Compiladores/Lab/Tarea4/Test/TestAFNToAFD$ cat test1_1
Automata de l.(l|d)*
Estados
4
0 [0]
1 [1 2 4 6 8 9]
2 [2 4 5 6 7 9]
3 [2 3 4 6 7 9]
Inicial
0
Aceptacion
3
1 2 3
Entradas
2
d l
Transiciones
7
0 l 1
1 d 2
1 l 3
3 d 2
3 l 3
2 d 2
2 l 3
xnpio@xnpio-Satellite-U40t-A:~/Documentos/Xnpio/UNSACS/Compiladores/Lab/Tarea4/Test/TestAFNToAFD$
```

Figura 2: Archivo de salida

- $(b|bc)^+$

```
xnpio@xnpio-Satellite-U40t-A:~/Documentos/Xnpio/UNSACS/Compiladores/Lab/Tarea4/Test/TestAFNToAFD$ cat test2
Automata de (b|b.c)+
Estados
10
0 1 2 3 4 5 6 7 8 9
Inicial
8
Aceptacion
1
9
Entradas
2
b c
Transiciones
11
0 b 1
2 b 3
4 c 5
3 ~ 4
1 ~ 7
5 ~ 7
6 ~ 0
6 ~ 2
7 ~ 9
7 ~ 6
8 ~ 6
xnpio@xnpio-Satellite-U40t-A:~/Documentos/Xnpio/UNSACS/Compiladores/Lab/Tarea4/Test/TestAFNToAFD$
```

Figura 3: Archivo de entrada

```
xnpio@xnpio-Satellite-U40t-A:~/Documentos/Xnpio/UNSACS/Compiladores/Lab/Tarea4/Test/TestAFNToAFD$ ./run test2 test2_2
xnpio@xnpio-Satellite-U40t-A:~/Documentos/Xnpio/UNSACS/Compiladores/Lab/Tarea4/Test/TestAFNToAFD$ cat test2_2
Automata de (b|b.c)+
Estados
3
0 [0 2 6 8]
1 [0 1 2 3 4 6 7 9]
2 [0 2 5 6 7 9]
Inicial
0
Aceptacion
2
1 2
Entradas
2
b c
Transiciones
4
0 b 1
1 b 1
1 c 2
2 b 1
xnpio@xnpio-Satellite-U40t-A:~/Documentos/Xnpio/UNSACS/Compiladores/Lab/Tarea4/Test/TestAFNToAFD$
```

Figura 4: Archivo de salida

- $(abc)*$

```
xnpio@xnpio-Satellite-U40t-A:~/Documentos/Xnpio/UNSACS/Compiladores/Lab/Tarea4/Test/TestAFNToAFD$ cat test3
Automata de (a.b.c)*
Estados
8
0 1 2 3 4 5 6 7
Inicial
6
Aceptacion
1
7
Entradas
3
a b c
Transiciones
9
0 a 1
2 b 3
1 ~ 2
4 c 5
3 ~ 4
5 ~ 7
5 ~ 0
6 ~ 0
6 ~ 7
xnpio@xnpio-Satellite-U40t-A:~/Documentos/Xnpio/UNSACS/Compiladores/Lab/Tarea4/Test/TestAFNToAFD$
```

Figura 5: Archivo de entrada

```
xnpio@xnpio-Satellite-U40t-A:~/Documentos/Xnpio/UNSACS/Compiladores/Lab/Tarea4/Test/TestAFNToAFD$ ./run test3 test3_3
xnpio@xnpio-Satellite-U40t-A:~/Documentos/Xnpio/UNSACS/Compiladores/Lab/Tarea4/Test/TestAFNToAFD$ cat test3_3
Automata de (a.b.c)*
Estados
4
0 [0 6 7]
1 [1 2]
2 [3 4]
3 [0 5 7]
Inicial
0
Aceptacion
2
0 3
Entradas
3
a b c
Transiciones
4
0 a 1
1 b 2
2 c 3
3 a 1
xnpio@xnpio-Satellite-U40t-A:~/Documentos/Xnpio/UNSACS/Compiladores/Lab/Tarea4/Test/TestAFNToAFD$
```

Figura 6: Archivo de salida