Université De La Rochelle

Licence 3 - Informatique

Compilation

Rapport du Projet - Analyseur lexicale en Java

- Mohammed BENAOU
- Mohammed RASFA

Responsable : Pr. Viaud Jean-François

25 Février 2017

1 Introduction

Le présent rapport a pour but de décrire le déroulement de l'implémentation d'un programme de compilation en Java dans le cadre du réalisation d'un analyseur lexicale. Ce rapport contient l'ensemble des éléments du projet, le plan du projet contient trois axes principaux, dans la première partie on expliquera le fonctionnement générale du moteur AFN, ensuite d'une part on procédera à clarifier le passage d'un AFN à un AFD et d'autre part on se concentra à la génération d'un automate à partir d'une expression régulière. Finalement, nous allons présenter la stratégie adopter dans la gestion de l'avancement de notre projet ainsi les difficultés rencontrées.

2 La partie technique

2.1 Mode d'emploi

notre projet < projetCompilation> contient quatre classes :

- La classe projetCompilation : c'est tout simplement notre main ;
- La classe Automate : c'est la classe principale du projet où on trouve les fonctions fondamentales (Automate(), AfficherAutomate(), verification( ), traitement ( ), lambdafermeture(), transiter(), Determinisation(), ecrire_fichier(), genere_fichier_dot()). ;
- La classe Transition : c'est une classe contient tous les composants d'une transition l'etat(d'entré et de sortie), les alphabet(d'entrées et de sorties). ;
- La classe ExpressionReguliere : c'est la classe qui permet de générer un automate á partir d'une expression régulière.

3 Le fonctionnement du projet

Après avoir exécuté le main du programme suivant :

```
18  public class ProjetCompilation {
19
20      /**
21       * @param args the command line arguments
22       * @throws java.io.IOException
23       */
24      public static void main(String[] args) throws IOException {
25
26          String Nom_fichier="NDSL01";
27          Automate Aut= new Automate(Nom_fichier+".descr");
            Automate Aut1= new Automate();
29          ExpressionReguliere exp = new  ExpressionReguliere();
30          //HashSet <String>tab = new HashSet();
31          Scanner sc = new Scanner(System.in);
32          Scanner choix = new Scanner(System.in);
33          List<String>msg= new ArrayList();
34          String saisie = "";
35
            String c = "";
37
38          do{
39          System.out.println("1.Affichage de l'Automate.");
40          System.out.println("2.Traitement des entrees.");
41          System.out.println("3.Exporter fichier .Dot.");
42          System.out.println("4.Determinisation.");
43          System.out.println("5.créer un automate");
44          System.out.println("6.Quitter");
45          System.out.println("Validez votre choix : ");
46          c = choix.nextLine();
47          System.out.println("---------------------------------------");
```

on aura ce menu comme résultat

```
run:
1.Affichage de l'Automate.
2.Traitement des entrees.
3.Exporter fichier .Dot.
4.Determinisation.
5.créer un automate
6.Quitter
Validez votre choix :
```

- choix numéro 1 : permet d'afficher l'automate entré en paramètre

```
String Nom_fichier="NDSL01";
Automate Aut= new Automate(Nom_fichier+".descr");
Automate Aut1= new Automate();
```

Voila le résultat

```
Le nom de l'automate est : []
Meta-Caractere : null
les entrees sont : [a, b, c]
les sorties sont : []
nombre d'etats : 6
l'etat(s) initial(s) est : [0]
l'etat(s) acceptant(s) est : [5]
Les Transitions de l'automate sont :
(0,a,1,null)
(0,a,2,null)
(0,b,5,null)
(0,c,5,null)
(1,b,3,null)
(2,c,4,null)
(3,b,5,null)
(4,c,5,null)
```

- choix numéro 2 : c'est le moteur AEF où on saisi les phrases à traiter et après on reçoit bien sur le résultat de ces phrases.
- choix numéro 3 : nous transformons l'automate (AEF) en fichier .dot et qu'on peut après l'afficher sous forme de graphe é l'aide du logiciel dédié à ce fait 'graphiz'ou le site web 'graphiz en ligne'.
- choix numéro 4 : c'est le passage d'un AFND à un AFD et on aura encore un petit menu.

```
--------------------------------------
=>1.Afficher l'Automate Deterministe.
=>2.Afficher les SuperEtats.
=>3.Générer fichier Dot.
=>4.Retour.
```

- choix numéro 5 : il permet de créer un fichier '.descr 'à partir d'une expression régulière passé par paramètre.
- choix numéro 6 : pour quitter l'exécution.

3.1 Moteur AEF

On va commencer par l'affichage de l'automate à partir son fichier '.descr 'en utilisant les expressions régulières.
On prend par le premier exemple :

```
Pattern m =Pattern.compile("^M\\s*(#)$");
```

ˆ indique le début dans m en suite on trouve M comme premier caractère, s* pour plusieurs espace entre M et le prochain caractère, $ signifie la fin de la phrase.

```
Pattern v =Pattern.compile("^V\\s*\"([0-1a-zA-Z]+)\"$");
```

Il faut ajouter les guillemets par ce que dans les fichiers '.descr 'où on trouve les alphabets d ?entrées sont toujours déclaré entre guillemet il peut y avoir des chiffres entre 0 et 1 et entre les lettres et a et z minuscule et majuscule après on ajoute (+) pour préciser qu'il faut trouver ça au moins une seule fois. En ce qui concerne la lecture du le 'BufferReader'voilà un simple exemple :

```
// on test si le premier caractere est M et
        if(ligne.split(" ")[0].equals("M")){
            Matcher match = m.matcher(ligne);
           if(match.lookingAt())
            this.lambda = match.group(1);


        }
```

On verifie si le premier caractère est M, et on prend le pattern 'm'qui contient l'expression régulière de M après on trouve une condition pour tester si le match est non vide dans ce cas on prend tous ce qu'on a reçu dans l ?attribut correspondant. On fait la même chose pour le reste (ventres, sorties, nombre d'états ?). Pour l'affichage de l ?automate on a utilisé juste des 'system.out.println'des éléments de l'automate stocké dans la fonction précédente.

```
public void AfficherAutomate() throws IOException{ // methode de l'affichage de l'automate

        Transition transition; //variable temporaire

            System.out.println("Le nom de l'automate est : "+this.Nom.toString());
            System.out.println("Meta-Caractere : "+this.lambda); // affichage du metacarac
            System.out.println("les entrees sont : "+this.ventres.toString()); // affichag
            System.out.println("les sorties sont : "+this.sorties.toString());// affichage
            System.out.println("nombre d'etats : "+this.nbretats);   // affichage de E nom
            System.out.println("l'etat(s) initial(s) est : "+this.etatinit); // affichage
            System.out.println("l'etat(s) acceptant(s) est : "+this.etataccp); // affichag
            System.out.println("Les Transitions de l'automate sont :");
            // affichage de l'ensemble des Transitions separer avec ',' à l'aide de la bou
            for (int i=0; i<this.transition.size(); i++){
                transition = this.transition.get(i);
                    System.out.print( "(" +
                        transition.getEtatdebut() +","+
                        transition.getEntree() +","+
                        transition.getEtataccp() +","+
                        transition.getSortie() +")");
                    System.out.print("\n");
    }
}
methode pour verifier si les inputs saisie sont les mêmes que les alphabets d'entrée
```

Relativement à la fonction vérification, un 'boolean'qui renvoie true ou false selon à ce qu'on a saisie au clavier, si la lecture est nulle alors bool renvoie true sinon alors c'est le cas de la lecture non nulle, on stocke les variables d'entrées au clavier puis on compare avec les variables d'entrées de cet automate.

```
// methode pour verifier si les inputs saisie sont les mêmes que les alphabets d'entrée
   public boolean verification(String lecture){
       Transition tran = new Transition();
       boolean bool = false;

       if(lecture==null)
           bool=true;
       if(lecture != null){
       for(int i=0;i<lecture.length();i++)
       {
               char caractere=lecture.charAt(i);
                   for(int j=0;j<transition.size();j++){
                           tran = transition.get(j);
                           // après qu'on boucle sur la lecture on test si la lecture est égal
                           if (caractere == tran.getEntree().charAt(0) || caractere == '#'){
                              bool = true;
                              break;
                           }
                           else
                              bool = false;
                   }
           if(bool == false)
               break;
       }
       }
       return bool;
   }
```

La fonction de traitement permet de traiter ce qu'on a saisie au clavier et de vérifier si la phrase est acceptée par l'automate ou non. Cette fonction reçoit comme argument la liste des entées, premièrement on a été amené à faire une boucle sur cette liste 'list'et puis une autre boucle sur chaque mot 'lecture'puis une troisième sur les transitions afin de faire des comparaisons.
Deuxièmement on compare si l'état courant est équivalent à celui de l'état début des transitions apèrs on comparer chaque alphabet de saisie au clavier avec les alphabets entré dans les transitions s'ils sont identiques alors on affiche l'état courant et l'alphabet saisie au clavier et puis si dans les transitions on trouve une sortie on l'affiche, si les alphabets ne sont pas identiques affiche l'état courant et l'alphabet entré et que la transition est non trouvé. Apèrs on fait un dernier teste si l'état acceptant dans la transition contient l'état courant genre l'état courant égal à l'état final.

```
----------------------------------------
Ecrivez votre message :
00
01
01010
###
Etat courant : 0, Entree : 0
Sortie : a, Transition trouvee
Etat courant : 0, Entree : 0
Sortie : a, Transition trouvee
Entree acceptante.
--->La sortie de cette phrase est : aa
---Fin de phrase---
-Fin de traitement-
Etat courant : 0, Entree : 0
Sortie : a, Transition trouvee
Etat courant : 0, Entree : 1
Transition trouvee
Entree non acceptante.
--->La sortie de cette phrase est : a
---Fin de phrase---
-Fin de traitement-
Etat courant : 0, Entree : 0
Sortie : a, Transition trouvee
Etat courant : 0, Entree : 1
Transition trouvee
Etat courant : 1, Entree : 0
Sortie : b, Transition trouvee
Etat courant : 0, Entree : 1
Transition trouvee
Etat courant : 1, Entree : 0
Sortie : b, Transition trouvee
Entree acceptante.
--->La sortie de cette phrase est : abb
---Fin de phrase---
```

3.2 Determinisation

Dans cette partie de determinisation on a travaillé avec trois fonctions fondamentales

```java
// Automate NonDéterministe =>Lambda fermeture (Utilisation de l'algorithme  du cours)
public ArrayList<Integer> lambdafermeture ( HashSet<Integer> T ) { //  Ensemble d'etat
        ArrayList<Integer> pile = new ArrayList<Integer>() ;
        ArrayList<Integer> listsupp = new ArrayList<Integer>(); // l'ensemble des états à supprim
       ArrayList<Integer> listajout = new ArrayList<Integer>(); // l'ensemble des états à ajouter

       for ( Integer e : T ){ // on ajoute tous les etats de T dans la pile
            pile.add( e );
        }
        ArrayList<Integer> F = new ArrayList<Integer>();
        while ( ! pile.isEmpty() ){
            for ( Integer t : pile ){  // t est le sommet de la pile si t n'est pas dans la liste
                if ( ! F.contains( t ) ){
                    F.add( t );

                    for ( Transition transition : this.transition ){ // on empile chaque u tel qu
                        if ( transition.getEtatdebut() == t && transition.getEntree().equals("#")
                            listajout.add ( transition.getEtataccp() );
                    }
                }
                listsupp.add( t );
            }
            pile.addAll( listajout); // Empiler u sur p
            pile.removeAll( listsupp );
        }
        Collections.sort(F);
        return F;
    }//Fin Lambdafermeture
```

La première c'est fonction lambda fermeture, cette fonction reçoit comme argument une liste des états, on a utilisé deux listes pour mettre les états empiler et dépiler et une dernière ou on va stocker les états résultant (F).
Dans un premier temps c'est qu'on va ajouté tout les états de T dans la pile et puis on teste si le sommet de la pile n'est pas dans la liste résultante (F) alors on ajoute le sommet dans F, puis on boucle sur les transitions afin de trouver une transition avec un état début le sommet et une entrée lambda (#) si on la trouve on ajoute cette transition dans la liste des ajouts et on ajoute le sommet dans la 'listsupp', après avoir boucler sur toutes les éléments on ajoute les éléments de listajout dans la pile et on supprime les éléments de 'listsupp'de la pile et retourne (F).

```java
// Fonction Transiter le meme algorithle que celle du cours
public HashSet<Integer> transiter( ArrayList<Integer> T , String a ){
        HashSet<Integer> F = new HashSet<Integer>();
            for ( Integer t : T ){ // t etat de T
                for ( Transition transition : this.transition ){
                    // on boucle sur chaque u tel que t ->u avec a comme transition
                    if ( transition.getEtatdebut() == t && transition.getEntree().compareTo( a )
                        if ( ! F.contains( transition.getEtataccp() ) )
                            F.add ( transition.getEtataccp() ); // ajouter u dans F
                        }
                    }
                }
        return F;
            }
```

la deuxième c'est la fonction Transiter, cette fonction reçoit comme argument une liste d'état T et une Entrée , on a commencé par déclaré une liste F qui est la liste résultante, puis on a bouclé sur tous les états de (T) ainsi que sur tous les transitions afin de trouver une transition qui mène de l'état début à un état de T et comme entrée le caractère d'entrée donné dans l'argument de la fonction,on vérifie si l'état n'est pas dans liste (F), si c'est le cas alors on ajoute l'état dans (F) et on retourne (F). la dernière fonction c'est 'Determinisation', c'est la fonction principale dans cette partie, on a commencé par déclarer une liste pour les super états, une liste pour des Etats, deux listes pour une liste d'état (etat), une autre liste des nouvelles états acceptants et une dernière liste U pour stocker. On a commencé par l'état initial s'il est vide donc ajoute 0 puis on ajoute dans la liste des liste de super états le résultat de lambda fermeture de l'états initial comme on a vu en TD, et on ajoute cet état dans la liste Etat, et puis on boucle et on ajoute les états dans liste (Etat) des états traité s'il n'est pas dans cette dernière. Après on combine lambda fermeture et transiter et on stocke le résultat dans la liste U, et puis on vérifie si U est vide alors l?état fin est -1 (c'est l'état puit) sinon on vérifie si la liste Etat contient ce qu'on a dans U sinon on ajoute U dans Etat et puis l'état fin sera l?indice de U selon l'ordre. Après on met les états traités résultants dans la liste des (newetat) et on boucle sur les états acceptants et les états traités et on ajoute les états acceptants qui sont dans les états traités dans Newetataccp, après on défini les entrées et les sorties et l'état initial et le nombre d'état et on retourne T l'automate final. On prend par exemple le fichier de test 'NDSL0'qui est un AFND.

```
----------------------------------------
Le nom de l'automate est : []
Meta-Caractere : null
les entrees sont : [a, b, c]
les sorties sont : []
nombre d'etats : 6
l'etat(s) initial(s) est : [0]
l'etat(s) acceptant(s) est : [5]
Les Transitions de l'automate sont :
(0,a,1,null)
(0,a,2,null)
(0,b,5,null)
(0,c,5,null)
(1,b,3,null)
(2,c,4,null)
(3,b,5,null)
(4,c,5,null)
1.Affichage de l'Automate.
2.Traitement des entrees.
3.Exporter fichier .Dot.
4.Determinisation.
5.créer un automate
6.Quitter
Validez votre choix :
```

On va essayer de déterminiser cet automate alors on valide le choix 4

```
Le nom de l'automate est : []
Meta-Caractere : null
les entrees sont : [a, b, c]
les sorties sont : []
nombre d'etats : 5
l'etat(s) initial(s) est : [0]
l'etat(s) acceptant(s) est : [2]
Les Transitions de l'automate sont :
(0,a,1,null)
(0,b,2,null)
(0,c,2,null)
(1,a,-1,null)
(1,b,3,null)
(1,c,4,null)
(2,a,-1,null)
(2,b,-1,null)
(2,c,-1,null)
(3,a,-1,null)
(3,b,2,null)
(3,c,-1,null)
(4,a,-1,null)
(4,b,-1,null)
(4,c,2,null)
```

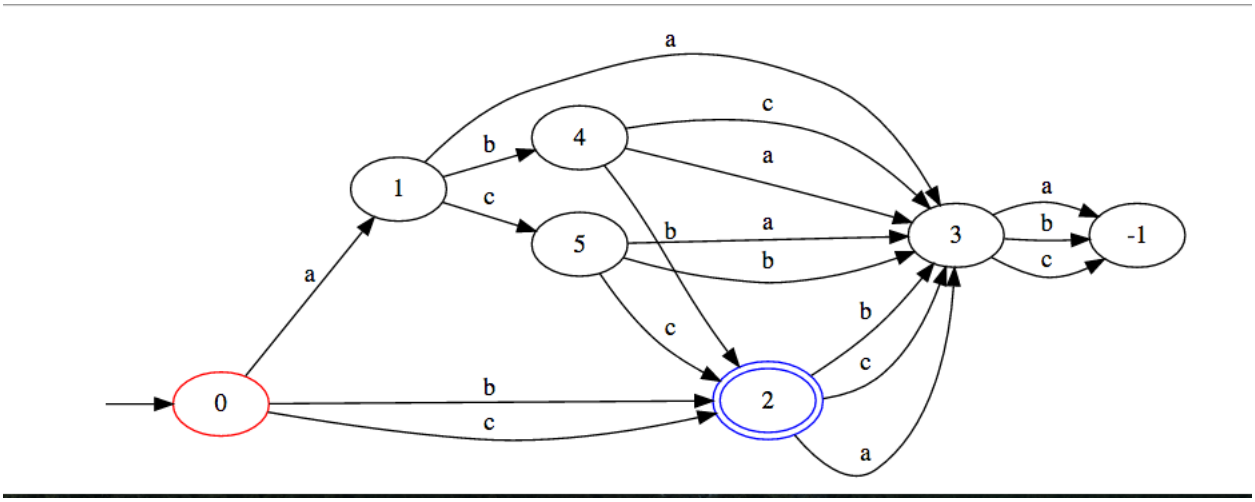
Les supers états (Superetat)

```
--------------------
[0]<==>0
[1, 2]<==>1
[5]<==>2
[3]<==>3
[4]<==>4
```

Pour montrer l'efficacité de notre implémentation, nous avons crée nos propres fichiers de tests qui représentent des automates détérministes assurant les traitements des phrases, et d'autre part on a vérifier le bon fonctionnement du programme de détérminisation en utilisant le fichier 'Simo.descr 'ci-joint dans les répértoires du code.

3.3 l'importation du fichier '.dot'

Dans cette fonction on va afficher l'automate comme un graphe en notation DOT ainsi que le fichier '.dot 'va prendre le même nom de fichier '.descr'et pour l'implémentation on a travaillé avec la fonction 'ecrire_fichier 'avec sa méthode write qui nous permet d'écrire dans le fichier, puis on a suivi la description donné pour l'affichage d'automate en fichier Dot. Le graphe de l'automate de test 'NDSL01'après la 'déterminisation'.



On a évité d'afficher les alphabets de sortie quand ils sont nuls.

3.4 Expression régulière

comme on a déjà indiqué dans le menu (choix 5), on génère un automate à partir d'une expression régulière passé par paramètre

```
public class ExpressionReguliere extends Automate{
    private ArrayList<Character>valeur;
    public ExpressionReguliere(){
       Automate aut = new Automate();
    }
     public String nom="";
    public ExpressionReguliere (String expression){
       super();
       this.expression= "(a+b)((a)*+(ba)";
       nom="(a+b)((a)*+(ba)";
       this.valeur = new ArrayList<Character>();
    }
```

On a crée la fonction 'generer_fichier()'pour générer un 'fichier .descr', la même chose on travail toujours avec la méthode 'write'comme la fonction précédente(ecrire_fichier) aprés le traitement des fonction (detecter et zero) pour qu'on puisse avoir le nombre d'états et les transitions entre eux.

4 Conclusion

Dans la fin, il fallait penser aux objectifs et aux méthodes qu'on a suivi pour atteindre le but de ce travail. Sans oublier que nous avons rencontré quelques difficultés au niveau de l'implémentation du programme de chaque fonction et les erreurs associés.
En revanche la création d'un analyseur lexicale en java nous a permis de découvrire plus profondément plusieurs aspects concernant les automates et la compilation en générale.