Feuille de TP 6 : Algorithme de Glushkov

Rappel : L'algorithme de Glushkov prend en entree une expression rationnelle et renvoie un automate reconnaissant le langage decrit par cette expression. Cet algorithme consiste en trois etapes :

1. Lineariser l'expression, c'est-a-dire remplacer les lettres par des symboles di erents ;
2. Calculer les ensembles *premiers, derniers* et la fonction *successeurs* ;
3. Creer un etat pour chaque \lettre" de l'expression linearisee et un etat initial. Les etats appartenant a *derniers* seront les etats terminaux. Ajouter des transitions de l'etat initial vers les etats de *premiers* et de chaque etat vers ses successeurs. L'etiquette d'une transition est la lettre (dans l'expression originale) qui correspond a son etat destination.

Vous avez au moins deux possibilites pour coder \lettres" de l'expression linearisee :

{ par des entiers stockes dans des champs de type `char`
{ par des instances de la classe `Feuille`.

Les exercices du present sujet proposent d'utiliser la premiere solution. La deuxieme peut donner lieu a une solution plus elegante mais peut-être plus difcile si vous avez eu du mal a manipuler les structures de Java. Des indications sont donnees en section 4.

Il est demande, pour chaque exercice, d'ecrire des tests dans des methodes `main` pour chacune des classes.

1 Linearisation

Exercice 1 :

Creer une classe `Glushkov` avec un champ `Vector<Character> positions`. Ce vecteur permettra de retrouver la lettre de l'expression originale qui correspond a une lettre de l'expression linearisee. Ecrire un constructeur `Glushkov()` pour cette classe.

Exercice 2 :

Ecrire une methode `Arbre lirePostfix(String expr)` qui renvoie l'arbre de la linearisee de l'expression `expr`. Cette methode est similaire a la methode `static Arbre lirePostfix(String expr)` de la classe `Arbre`, sauf que les feuilles sont etiquetees par `(char) 1`, `(char) 2`, etc. et qu'on met a jour le champ `positions` chaque fois une feuille est inseree dans l'arbre.

2 Premiers, derniers, successeurs

Exercice 3 :

Ajouter une methode `abstract boolean contientMotVide()()` dans la classe `Arbre` et l'implementer dans les classe `Feuille`, `Unaire`, `Binaire`.

Exercice 4 :

Ajouter une methode `abstract Set<Character> premiers()` dans la classe `Arbre` et l'implementer dans les classe `Feuille`, `Unaire`, `Binaire`.

Exercice 5 :

Ajouter une methode `abstract Set<Character> derniers()` dans la classe `Arbre` et l'implementer dans les classe `Feuille`, `Unaire`, `Binaire`.

Exercice 6 :

Ajouter une methode `abstract Map<Character, Set<Character>> succ()` dans la classe `Arbre` et l'implementer dans les classe `Feuille`, `Unaire`, `Binaire`.

3 Construction de l'automate

Exercice 7 :

Ecrire dans la classe `Glushkov` une methode `AutomateND glushkovND(String expr)` qui lit une expression sous forme post x et renvoie un automate non-deterministe qui la decrit. Cette methode :

- { cree l'arbre de l'expression linearisee, puis calcule les ensembles *premiers*, *derniers* et la fonction *successeurs*
- { cree un tableau d'`EtatND` de taille `n = positions.size() + 1`
- { cree pour chaque case du tableau un `EtatND` portant comme identi ant son index dans le tableau. L'etat de la case 0 est initial, les etats dont l'index apparait dans *derniers* sont terminaux.
- { ajoute des transitions de l'etat 0 vers les etats de *premiers*, c-a-d les etats dont l'index est dans *premiers*, puis ajoute des transitions de chaque etat vers ses successeurs. On trouvera l'etiquette d'une transition a l'aide du vecteur `positions`.

Payer attention a la gestion des indices : l'element i de `positions` a comme identi ant i + 1.

4 Implementation alternative

Si vous choisissez de coder les lettres de l'expression linearisee au moyen des instances de la classe `Feuille`, les etapes sont les suivantes :

1. La section 1 est inutile ;
2. Dans la section 2, les ensembles renvoyes sont des ensembles de feuilles et non de caracteres (et pour le dernier exercice, le Map associera un ensemble de feuilles a chaque feuille) ;
3. Pour la construction de l'automate, on utilisera un `Map <Feuille, EtatND>` et on parcourra l'ensemble des feuilles de l'arbre avec une boucle `for`.