Projet AF4 :
Recherche dans un chier

version 1, mars 2011

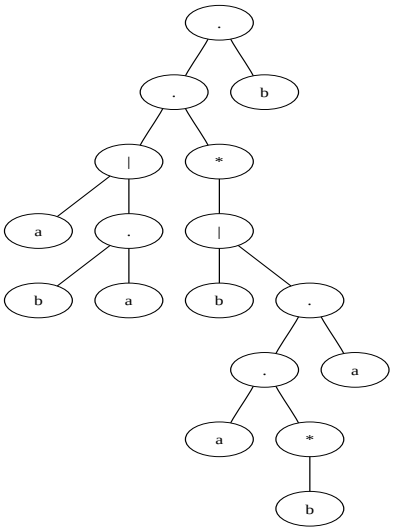
Le projet consistera en un programme Java qui prend en parametre une expression rationnelle et le nom d'un chier, et recopie sur la sortie standard toutes les lignes du chier qui contiennent, comme facteur, un mot decrit par l'expression.

Le principe

Le coeur du projet est l'implementation de l'algorithme de Glushkov pour fournir un automate ni deterministe reconnaissant le langage decrit par une expression rationnelle donnee.

Pour traiter une expression rationnelle, on la representera au moyen d'un arbre binaire.

Ainsi, l'expression suivante : $(a|ba)(b|ab^*a)^*b$ sera representee par l'arbre ci-dessous.



Note : le symbole | est utilise ici dans le même sens que +, c'est a dire pour l'union.

L'analyse syntaxique consistant a transformer l'expression sous sa forme standard parenthesee en un arbre est un peu di cile et ne constitue pas le coeur du projet. Vous utiliserez donc dans un premier temps la notation $postfixe$ (ou polonaise inversee) qui possede l'avantage considerable de ne pas necessiter de parentheses.

L'idee est la suivante, au lieu d'ecrire (a + b), on ecrit ab+. A titre d'exercice, veri ez que l'expression ci-dessous reproduit l'arbre donne en exemple :

$$a\,b\,a\,\cdot\,|\,b\,a\,b\,*\,\cdot\,a\,\cdot\,|\,*\,\cdot\,b\,\cdot$$

Les étapes principales

1. Ecrire une methode qui, a partir d'une expression rationnelle sous forme post xe, renvoie l'arbre correspondant.
2. Ecrire une methode qui, a partir d'un arbre representant une expression rationnelle, renvoie un automate non deterministe. Il s'agira d'appliquer l'algorithme de Glushkov vu en cours, au moyen des methodes rappelees dans la partie implementation.
3. Ecrire une methode qui, a partir d'un chier et d'un automate, deroule l'execution de l'automate sur le contenu du chier (qui sera donc le mot a reconna^tre) et a che chaque ligne dans laquelle l'execution fait passer l'automate par un etat terminal. Attention : l'automate construit a partir d'une expression ne sera pas utilise tel quel puisqu'on cherche un facteur et non un mot exact.
4. La construction d'un automate deterministe, soit directement par l'utilisation des tables de l'algorithme de Glushkov, soit par determinisation, est optionnelle.

Le resultat du projet comprendra, outre le code correctement commente et teste, un document expliquant les choix d'implementation e ectues et leur incidence sur l'e cacite du programme, et decrivant plusieurs tests non triviaux du projet.

Implémentation

Les arbres

Vous verrez en TP une implementation des arbres d'expression tirant parti des possibilites fournies par la programmation objet, basee sur une classe abstraite `Arbre` avec un champ `symbole` et trois classes derivees `Feuille`, `Unaire` et `Binaire`.

Vous pouvez egalement utiliser une autre structure, par exemple une unique classe `Arbre`, avec un champ `symbole` et deux champs `gauche` et `droite` de type `Arbre`, ou une feuille est un Arbre tel que `gauche == droite == null`, et si `symbole == '*'`, alors un des deux ls est `null` .

Attention : cette implementation n'est pas recommandee. Elle sera sans doute plus facile a compiler que celle proposee en TP mais (donc) plus susceptible d'attirer les bugs.

L'analyse d'une expression postfixe

Il pourra être commode d'utiliser une pile d'Arbre.

- A chaque fois qu'on lit une lettre, on rajoute sur la pile l'Arbre correspondant a une feuille etiquetee par cette lettre.
- Si on rencontre un operateur binaire ($\cdot$ ou $|$), on prend les deux premiers elements sur le dessus de la pile, on leur applique l'operateur pour creer un nouvel Arbre, qu'on repose ensuite sur le dessus de la pile.
- Si on rencontre le symbole $*$, on prend le premier element de la pile, on lui applique $*$ en creant un nouvel Arbre, qu'on repose ensuite sur le dessus.

Glushkov

Les classes d'arbres devront implementer les methodes suivantes, qui reprendront les de nitions recursives donnees en cours.

```
Set<Feuille> premier();
Set<Feuille> dernier();
Set<Feuille> suivant();
boolean contientMotVide();
```