Projet de compilation - `lightGrep`

N. Delestre

1 Présentation générale

L'objectif de ce projet est de développer un grep simplifié. Pour rappel le programme `grep` permet de rechercher des occurences d'expressions rationnelles dans un fichier texte. Voici quelques exemples d'utilisation de `lightGrep` :

```
$ ./lightGrep.sh
Utilisation : expressionRationelle nomFichier
$ ./lightGrep.sh "L.*" src/fr/insarouen/iti/compilation/grep/Main.java
 * To change this license header, choose License Headers in Project Properties.
import fr.insarouen.iti.compilation.grep.automate.LettreException;
import java.util.List;
       List<int[]> positionsExpressionRationnelle;
           String ligne = fluxTexte.readLine();
               ligne = fluxTexte.readLine();
       } catch (ParseException | LettreException e) {
$ ./lightGrep.sh "L.*Exception" src/fr/insarouen/iti/compilation/grep/Main.java
import fr.insarouen.iti.compilation.grep.automate.LettreException;
       } catch (ParseException | LettreException e) {
```

FIGURE 1 – Exemple d'utilisation du programme lightGrep

Les expressions rationnelles acceptées par ce programme peuvent être constituées :

— du caractère `.` pour désigner n'importe quel caractère ;
— du caractère `*` pour désigner l'opérateur unaire suffixe pouvant répéter une expression rationnelle de 0 à n fois ;
— du caractère `+` pour désigner l'opérateur unaire suffixe pouvant répéter une expression rationnelle de 1 à n fois ;
— du caractère `?` pour désigner l'opérateur unaire suffixe pouvant répéter une expression rationnelle 0 ou 1 fois ;
— du caractère `|` (pipe) pour désigner une alternative sur des expressions rationnelles ;
— des crochets pour exprimer une alternative sur une liste de caractères, sur un intervale (utilisation du `-`), sur le complémentaire (utilisation de `^` juste après le crochet gauche) ;
— des paranthèses pour exprimer un regroupement d'expressions rationnelles ;
— des caractères `^` ou `$` pour exprimer le début ou le fin de la chaine de caractères ;
— n'importe quel caractère, avec utilisation de l'antislash (\) pour déspécialiser l'un des caractères précédents.

2 Analyse

L'application `ligthGrep` fonctionne de la façon suivante : pour chaque ligne du fichier texte, `lightGrep` recherche la présence, ou pas, d'une suite de caractères qui correspond à l'expression rationnelle donnée. Si cette recherche est positive, la ligne en question est écrite sur la sortie standard.

Nous avons vu dans le cours, qu'une expression rationnelle dénote un langage rationnel, tout comme les grammaires linéaires (à gauche ou à droite) ou les automates à états finis. Nous avons aussi vu qu'une expression rationnelle est surtout utilisée pour exprimer un langage, alors que les automates à états finis sont plutôt utilisés pour tester l'appartenance ou pas d'un mot à ce langage. Enfin nous avons vu qu'une

expression rationnelle peut être « traduite » en un automate à états finis avec epsilon transitions et que ce dernier peut être « traduit » en un automate sans epsilon transition non déterministe puis en automate déterministe.

`lightGrep` utilise donc un compilateur qui prend en entrée une expression rationnelle et qui produit en sortie un automate à états finis déterministe. Il faut donc d'une part déterminer la grammaire de ce langage et d'autre part modéliser le programme.

La grammaire des expressions rationnelles

L'ensemble des expressions rationnelles représentent un langage. Sachant que la description de ce langage, présentée précédemment, contient le fait qu'il est possible d'utiliser des parenthèses et crochets équilibrés, comme nous l'avons vu dans le cours, cela signifie que ce langage est algébrique.

Voici un exemple de grammaire (inspirée de `https://www2.cs.sfu.ca/~cameron/Teaching/384/99-3/regexp-plg.html`) en notation BNF pour les expressions rationnelles ($\langle er \rangle$ est l'axiome de la grammaire) :

$\langle er \rangle$: := $\langle er \rangle$ '`|`' $\langle erc \rangle$ | $\langle erc \rangle$

$\langle erc \rangle$: := $\langle erc \rangle$ $\langle erb \rangle$ | $\langle erb \rangle$

$\langle erb \rangle$: := $\langle ere \rangle$ | $\langle ere \rangle$ '`*`' | $\langle ere \rangle$ '`?`' | $\langle ere \rangle$ '`+`'

$\langle ere \rangle$: := '`(`' $\langle er \rangle$ '`)`' | $\langle lettre \rangle$ | '`.`' | '`[`' $\langle ens\text{-}lettres \rangle$ '`]`' | '`[`' '`^`' $\langle ens\text{-}lettres \rangle$ '`]`'

$\langle ens\text{-}lettres \rangle$: := $\langle ens\text{-}lettres \rangle$ $\langle element\text{-}ens\text{-}lettres \rangle$ | $\langle element\text{-}ens\text{-}lettres \rangle$

$\langle element\text{-}ens\text{-}lettre \rangle$: := $\langle lettre \rangle$ | $\langle lettre \rangle$ '`-`' $\langle lettre \rangle$

$\langle lettre \rangle$: := caracteres-alphanumériques | '`\`' caracteres-echapes

tel que :

$\langle er \rangle$ est le non terminal représentant les unions d'expressions rationnelles

$\langle erc \rangle$ est le non terminal représentant les concaténations d'expressions rationnelles

$\langle erb \rangle$ est le non terminal représentant les expressions rationnelles basiques

$\langle ere \rangle$ est le non terminal représentant les expressions rationnelles élémentaires

$\langle ens - lettres \rangle$ est le non terminal représentant un ensemble de lettres

$\langle element - ens - lettre \rangle$ est le non terminal représentant un élément d'un ensemble de lettres

$\langle lettre \rangle$ est le non terminal représentant les caractères utilisables dans les expressions rationnelles

Les terminaux quant à eux sont tous les caractères entre simple côte ('`|`', '`*`', '`+`', etc.), leurs versions échappées par un antislash ('`\|`'), ('`\*`'), ('`\+`'),etc.), et les caractères alphanumériques.

Modélisation du programme

Le diagramme de classes de la figure 2 présente l'analyse de ce projet.

Le programme principal référence une instance de `ExpressionRationnelle` qui est créée à partir de la chaine de caractères qui représente l'expression rationnelle donnée. À l'aide de la méthode `presences`, le programme principal est capable de savoir si des occurrences de l'expression rationnelle est contenu dans une chaine de caractères. Pour réaliser cela, la classe `ExpressionRationnelle` utilise un `AnalyseurSyntaxique` pour obtenir l'arbre syntaxique abstrait (instances de classes du package `arbreSyntaxiqueAsbtrait`) représentant l'expression rationnelle donnée, qu'il transforme alors en `Automate`.

3 Conception

Concernant la grammaire, comme nous allons utiliser un compilateur de compilateur de type LL1 n'acceptant pas les mots vides, il est donc nécessaire de dérécursiver la grammaire précédente (cf. le cours).

Concernant le programme, comme nous l'avons en cours, nous allons utiliser le patron de conception Visiteur. On a besoin d'une part que l'AST généré par l'analyseur syntaxique soit `Visitable`. Et d'autre

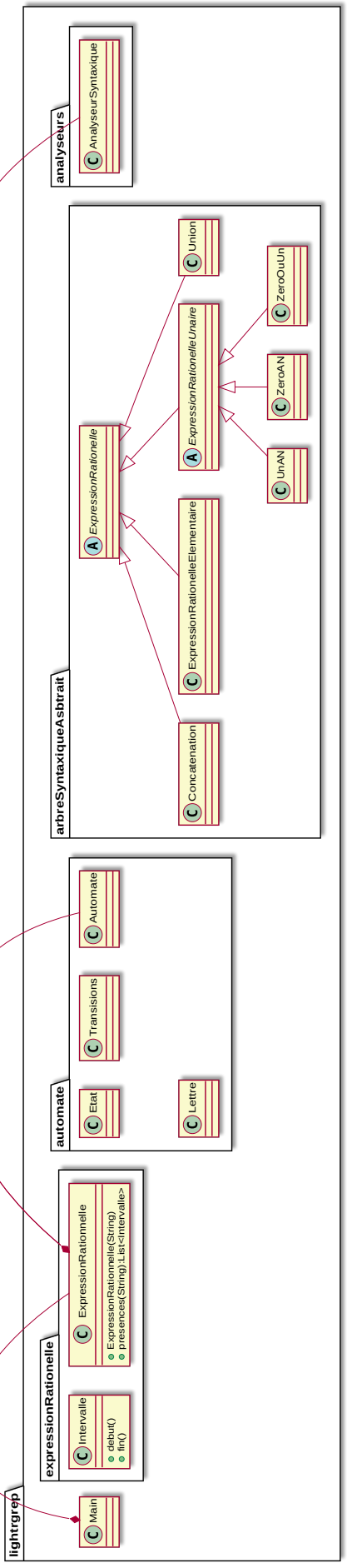

FIGURE 2 – Diagramme de classes de l'analyse

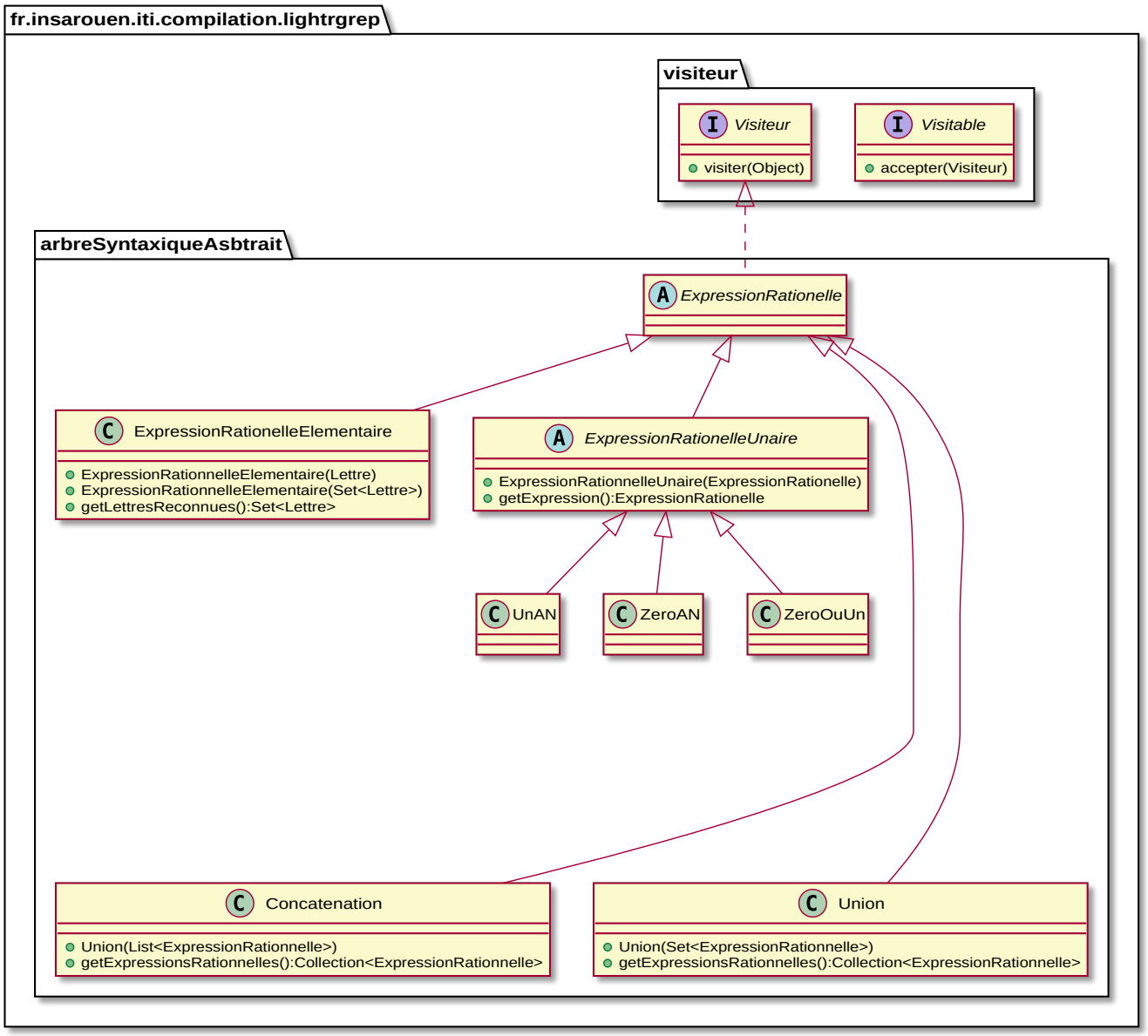

FIGURE 3 – Diagramme de classes des classes de l'AST

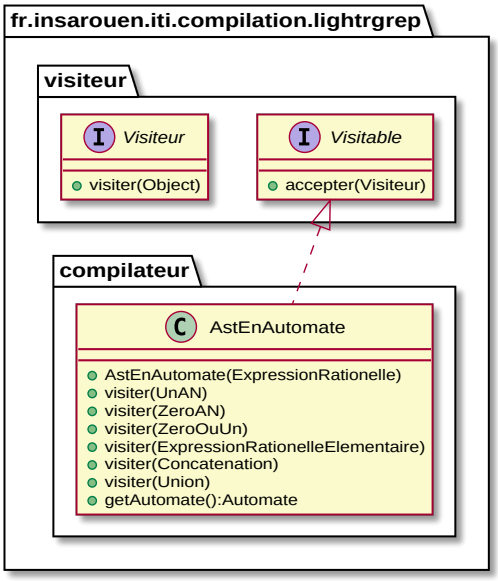

FIGURE 4 – Diagramme de classes du compilateur

part, on a besoin d'un compilateur, pour créer un automate à partir d'un AST, qui doit être un `Visiteur` : c'est le rôle de la classe `ASTEnAutomate`. Les figures 3 et4 présentent ces classes.

Enfin les classes `Automate` et `OutilPourAutomate` proposent toutes les méthodes permettant de construire un automate, de combiner des automates, de déterminiser un automate ou de supprimer les ϵ-transitions. La figure 5 présente cette classe.

Travail à réaliser

Dérécursivez la grammaire de l'analyse et supprimez l'utilisation des mots vides en utilisant l'opérateur de Kleene.

4 Développement

Excepté le fichier `src/fr/insarouen/compilation/lightgrep/analyseurs/AnalyseurSyntaxique.jj`, tout le code Java du projet vous est donné.

Le `Makefile` donné permet de compiler le fichier `AnalyseurSyntaxique.jj` à l'aide de `javacc` et de compiler l'ensemble des classes Java à l'aide de `javac` (au moins la version 14 du jdk). Ces deux programmes devront êtres accessibles (leurs chemins devront être présents dans le `PATH`).

Vous pouvez tester votre code à l'aide de tests unitaires qui peuvent être lancés à l'aide du script `tests.sh` :

```
$ ./tests.sh
JUnit version 4.13.2
..........
Time: 0,024

OK (10 tests)
```

Conseil développez itérativement votre code javacc, par exemple :

1. reconnaître un seul caractère non spécialisé ;
2. reconnaître la concaténation de caractères non spécialisés ;
3. reconnaître les opérateurs `*`, `+` et `?` ;
4. reconnaître les ensembles de caractères ;
5. etc.

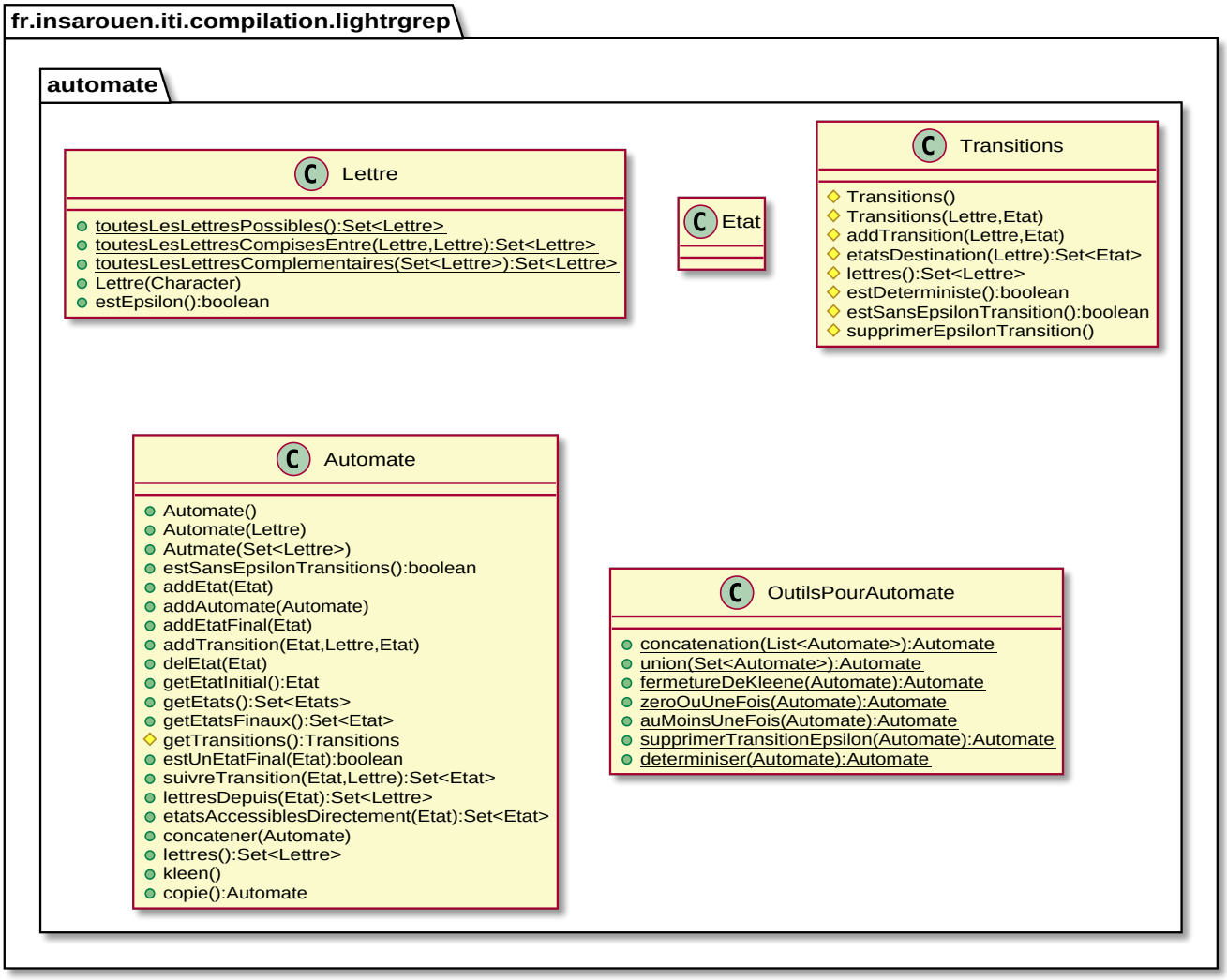

FIGURE 5 – Diagramme de classes des classes pour gérer les automates

5 Livraison

Vous ne changerez aucun nom (répertoires, fichiers et archive). Rien de vos fichiers ne doit permettre de vous identifier, l'évaluation doit être anonyme. Vous déposerez l'archive `lightgrep.zip` dans l'atelier moodle « Réalisation du projet et évaluation par les pairs ».