Département IMA / 4A (S8)
Informatique Fondamentale 2020/2021

TP-Projet IF - Parser des grammaires avec ANTLR

1 Premiers pas avec ANTLR

1.1 Setup ANTLR

Pour commencer on crée un dossier /projetIF avec un sous-dossier /lib et on télécharge antlr dedans, pour faire ça on peut lancer les commandes suivantes dans un terminal :

```
mkdir ~/projetIF ~/projetIF/lib
cd ~/projetIF/lib
wget https://www.antlr.org/download/antlr-4.9.2-complete.jar
```

Ensuite on va modifier le fichier /.bashrc, pour faire ça on peut lancer la commande :

```
gedit ~/.bashrc
```

Dans ce fichier on va ajouter les lignes suivantes :

```
export CLASSPATH=".:~/projetIF/lib/antlr-4.9.2-complete.jar:$CLASSPATH"
alias antlr4='java -Xmx500M -cp "~/projetIF/lib/antlr-4.9.2-complete.jar:$CLASSPATH" org.antlr.v4.Tool'
alias grun='java -Xmx500M -cp "~/projetIF/lib/antlr-4.9.2-complete.jar:$CLASSPATH" org.antlr.v4.gui.TestRig'
```

N'oubliez pas de sauvegarder avant de fermer le fichier. Ensuite il faut sourcer le fichier, pour ça on utilise la commande :

```
source ~/.bashrc
```

Le setup est terminé, vous pouvez vérifier que antlr est bien setup en lançant la commande :

```
java org.antlr.v4.Tool
```

Le terminal devrait vous renvoyer quelquechose qui commence par :

```
ANTLR Parser Generator  Version 4.9.2
-o ___              specify output directory where all output is generated
-lib ___            specify location of grammars, tokens files
...
```

1.2 Un premier exemple : la grammaire $\{a^n b^n \mid n \in \mathbb{N}\}$

Créez un sous dossier `/anbn` dans votre dossier `/projetIF`, récupérez le fichier `AnBn.g4` depuis moodle et placez le dans votre dossier `/anbn`.
Ouvrez un terminal dans ce dossier et lancez les commandes suivantes.
La première commande permet de créer les fichiers java qui permettent de parser la grammaire :

```
antlr4 AnBn.g4
```

Ensuite il faut compiler les fichiers java :

```
javac AnBn*.java
```

Enfin vous pouvez lancer le programme sur une chaîne de caractères dont vous voulez savoir si elle est reconnue par la grammaire. Lancez la commande :

```
grun AnBn prog -gui
```

Alors le programme attend qui vous lui écriviez une chaîne de caractères, vous pouvez lui écrire `aabb`.
Attention, pour valider votre chaîne de caractères il faut faire la combinaison de touches 'Ctrl' + 'd', cette combinaison permet d'écrire le caractère End Of File (en programmation on le note `EOF`). Le caractère Entrée est un caractère à part entière qui peut faire partie de votre grammaire (en programmation on le note `\n`).

Le programme doit alors vous afficher un arbre de dérivation de la grammaire pour la chaîne de caractères que vous lui avez donné. Vous pouvez expérimenter en testant des mots qui ne sont pas reconnus par la grammaire, par exemple `aaab` ou `hello`.

Il y a plusieurs manières de lancer la grammaire, par exemple on peut tester une chaîne de caractère dans un fichier. Récupérez le fichier `test.txt` sur moodle et placez le dans votre dossier `/anbn`. Pour lancer la grammaire sur le fichier on utilise une redirection, lancez la commande :

```
grun AnBn prog -gui < test.txt
```

Vous pouvez aussi lancer la grammaire pour obtenir l'arbre dans un format textuel :

```
grun AnBn prog -tree
```

2 Écrire des grammaire dans ANTLR

2.1 Comment écrire une grammaire dans ANTLR

Les grammaires en ANTLR sont définies dans des fichiers au format `.g4`. Inspectez le contenu du fichier `AnBn.g4` pour voir comment est définie la grammaire.
On remarque une distinction entre les non-terminaux de la grammaire, `prog` et `s`, et les tokens, `A` et `B`. Les noms de non-terminaux sont toujours en minuscule. Les tokens sont toujours en majuscule. La différence entre les deux est que les tokens se transforment directement en chaînes de caractères alors qu'on peut appliquer plusieurs règles consécutives aux non-terminaux.

On remarque aussi que le non-terminal initial n'est pas spécifié dans la grammaire. En fait on peut choisir le non-terminal de départ à chaque fois qu'on lance la grammaire sur une chaîne de caractères. La procédure pour lancer une grammaire nommée `Grammar` sur le non-terminal `initial` est :

```
antlr4 Grammar.g4
javac Grammar*.java
grun Grammar initial -gui
```

2.2 Exercices : quelques grammaires à écrire

Je vous conseille de créer un dossier de travail pour chaque exercice. Nommez vos grammaires comme indiqué dans chaque question. Les fichiers `.g4` sont à rendre à la fin du projet.

Exercice 1 *Programmez les grammaires suivantes (pensez à les vérifier sur des chaînes de caractères exemples) :*

- *La grammaire* `abbc.g4` *du langage* $L = a^*bb^*c$ *:* $G = (\{S, A\}, \{a, b, c\}, P, S)$ *avec* $P = \{S \to aS|bA, A \to bA|c\}$
  (Attention, dans le `.g4`*, les non-terminaux doivent être en minuscule)*
- *La grammaire* `a_pair.g4` *du langage* $L = (aa)^*$ *:* $G = (\{S\}, \{a\}, P, S)$ *avec* $P = \{S \to aaS|\varepsilon\}$.

Exercice 2 *Programmez des grammaires pour exprimer les langages suivants (ce sont les grammaires de l'exercice 6 du TD3) :*

- *Les mots sur l'alphabet* $\{a, b\}$ *qui sont égaux à leur mot miroir, grammaire* `miroir.g4`.
- *Les mots sur* $\{a, b\}$ *contenant autant de a que de b, grammaire* `ab_equal.g4`.
- *Les mots sur* $\{a, b, c\}$ *contenant autant de a que de b, grammaire* `abc_equal.g4`.
- $\{a^n b^m c^m d^n, n, m \geq 0\}$*, grammaire* `abcd.g4`.

3 Expressions arithmétiques

Le but de cette partie est d'écrire une grammaire pour reconnaître des expressions arithmétiques et qui calcule le résultat de ces expressions. Pour ça on va utiliser des morceaux de code java dans notre grammaire.

3.1 Premier exemple : compter des lignes

Récupérez le fichier `CountLines.g4` sur moodle.
Ce programme permet de compter le nombre de retour à la ligne dans une chaîne de caractères.

C'est un exemple d'utilisation de code java pour enrichir la grammaire.

3.2 Expressions arithmétiques

Récupérez le fichier `ArithExpr.g4` sur moodle.
Inspectez le contenu du fichier et testez le sur des exemples d'expressions arithmétiques (Attention le symbole pour la multiplication est x et pas *).

Exercice 3 *En modifiant le fichier* `ArithExpr.g4`*, ajoutez les calculs de la soustraction avec le caractère* `-` *et de la division avec le caractère* `/` *(testez sur des exemples).*

3.3 Problèmes de priorité

Dans les expressions arithmétiques, la priorité entre les différentes opérations (+, -, x, /) est prévue pour que l'expression $5 + 2$ x $9 - 2/2$ donne le résultat 22. Quel résultat obtenez vous avec votre grammaire ?

Observez l'arbre de dérivation et expliquez pourquoi vous obtenez votre résultat.