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

TP-Projet IF - Parser des grammaires avec ANTLR

1 Premiers pas avec ANTLR

Vous pouvez passer cette section si vous avez déjà fait la première partie du TP-projet.

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.

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 Expressions régulières

Créez une grammaire `ExpReg` qui reconnait les expressions régulières (expressions rationnelles) sur l'alphabet qui contient les lettres minuscules, majuscules et les chiffres.

Pour définir les tokens (les variables définies en bas du fichier .g4), on peut utiliser `[a-zA-Z0-9]` pour représenter les symboles de l'alphabet, et `[*]` pour représenter l'étoile `*`.

3 Exécution de petits programmes en C

3.1 Récupérer et tester la grammaire

Téléchargez le dossier `ProgrammeC` depuis Moodle et mettez-le dans votre dossier `/projetIF`.

Compilez la grammaire et testez la sur les fichiers `exemple1.txt` et `exemple2.txt`.

Remarquez que la grammaire ne fait marcher que certaines fonctionnalités simples du langage C : les variables sont toutes de type `int`, il n'y a ni boucles, ni fonctions, ni classes. On va ajouter quelques fonctionnalités mais notre grammaire ne fonctionnera pas exactement comme un programme C (notamment parce qu'on n'a pas de fonction main, mais surtout parce que de toute façon c'est trop compliqué de compiler du C).

3.2 Améliorer la grammaire

On va essayer d'améliorer un peu cette grammaire :

1. Ajoutez la possibilité de mettre un block de proramme entre accolades : `{ instruction }` (on se contente de faire fonctionner les accolades comme les parenthèses).
2. Vous avez peut-être remarqué que le `if` n'est pas écrit comme en C. Modifiez la grammaire pour qu'elle reconnaisse la construction du `if` du langage C :

   ```
   if ( condition )
       instruction
   ```

   Modifiez également le fichier `exemple2.txt`.
3. Dans le `if`, ajoutez la possibilité d'avoir un `else` :

```
if ( condition )
    instruction
else
    instruction
```

4. Ajoutez des opérateurs `<` et `>` pour comparer deux entiers. Le type du résultat doit être `int`, donc on représente le booléen `true` par l'entier `1` et le booléen `false` par l'entier `0` (ça correspond au fonctionnement du langage C).
5. Ajoutez la fonctionnalité de la boucle `while` :

```
while ( condition )
    instruction
```

6. Créez un fichier `exemple3.txt` pour tester toutes les fonctionnalités ajoutées.