Projet de Compilation des Langages (PCL)

Présentation du module

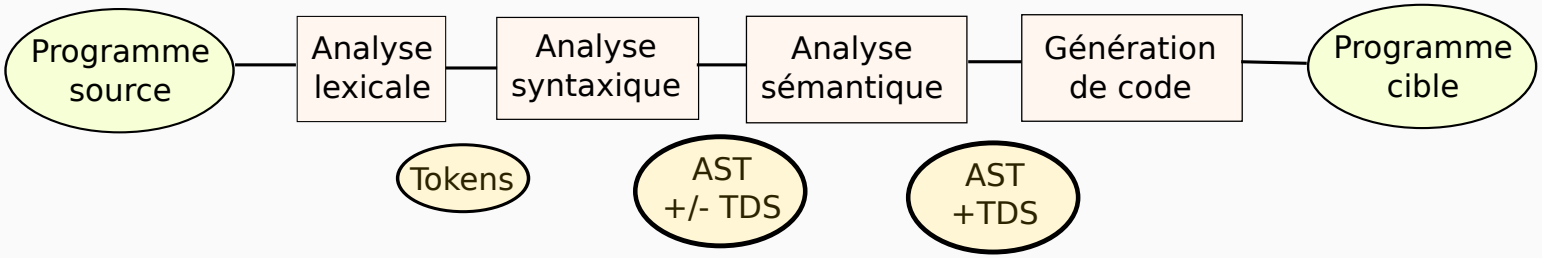
Suzanne Collin, Sébastien Da Silva, Pierre Monnin

2017 – 2018

TELECOM Nancy

Objectifs du module

Objectifs du module



Programmer un compilateur : de la définition de la grammaire à la génération du code assembleur

Première séance : prise en main d'ANTLR

Fichiers pour cette séance

Les *.jar, Expr.g et Test.java sont sous

- `/home/depot/2A/PROJET_COMPIL`
- https://github.com/pmonnin/compilation-lab

ANTLR

- ANTLR est un *générateur* d'analyseur lexical et syntaxique

  Entrée Fichier *.g (grammaire du langage)

  Sortie Deux classes Java réalisant l'analyse lexicale et syntaxique.

- Fichiers générés à intégrer au projet

- ANTLR à utiliser en version 3.3
- Antlrworks pour visualiser l'arbre de parsing et l'AST

Fichier grammaire .i

```
1  grammar Expr; // Nom identique au fichier
2
3  options
4  {
5      // Voir la doc
6  }
7
8  @header {
9  import java.util.HashMap;
10 }
11 @members {
12 HashMap<String,Integer> memory = new HashMap<String,Integer>();
13 }
14
15 // Non-terminaux en minuscules
16 prog:   stat+ ;
17
18 stat:   expr NEWLINE {System.out.println($expr.value);}
19     |   ID '=' expr NEWLINE {memory.put($ID.text, new Integer($expr.value
       ));}
20     |   NEWLINE
```

Fichier grammaire ii

```
21      ;
22
23  // [...]
24
25  multExpr returns [int value]
26      :   e=atom {$value = $e.value;} ('*' e=atom {$value *= $e.value;})*
27      ;
28
29  // [...]
30
31  // Terminaux en majuscules
32  ID  :   ('a'..'z'|'A'..'Z')+ ;
33  INT :   '0'..'9'+ ;
34  NEWLINE:'\r'? '\n' ;
35  WS  :   (' '|'\t')+ {$channel=HIDDEN;} ;
```

Intégration des classes générées dans votre projet

Fonction main de votre programme :

```java
public static void main(String[] args) throws Exception {
    ANTLRInputStream input = new ANTLRInputStream(System.in);

    /*
    Arbre_ListeLexer: ANTLR-generated class for the lexical analysis
    Arbre_ListeParser: ANTLR-generated class for the syntax analysis
    */
    Arbre_ListeLexer lexer = new Arbre_ListeLexer(input);
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    Arbre_ListeParser parser = new Arbre_ListeParser(tokens);

    parser.arbre(); // arbre is the name of the axiom rule

    /*
    Here: Java code for semantic analysis and code generation
    Get the AST from the ANTLR-generated classes and work with it
    */
}
```

Pour le projet

Consignes pour la grammaire

La grammaire doit être LL(1)

```
1 options
2 {
3     ...
4     k = 1;
5     ...
6 }
```

Ce qui signifie que le backtracking est interdit :

```
1 options
2 {
3     ...
4     backtracking = true; // INTERDIT
5     ...
6 }
```

Génération AST / TDS

- L'AST peut être généré :
  - Par les fichiers générés par ANTLR (souvenez-vous de la syntaxe de réécriture avec $\rightarrow$)
  - Par vous en parcourant le *parsed tree*... (déconseillé)

Génération AST / TDS

- L'AST peut être généré :
  - Par les fichiers générés par ANTLR (souvenez-vous de la syntaxe de réécriture avec $\rightarrow$)
  - Par vous en parcourant le *parsed tree*... (déconseillé)
- La TDS peut être générée :
  - Grâce à des fonctions sémantiques dans la grammaire
    - Se fera en même temps que l'analyse syntaxique
    - Résultat à récupérer en sortie des fichiers générés par ANTLR
  - En parcourant l'AST produit par l'analyse syntaxique

Groupes de projet & évaluations

- Groupes de 4 personnes
- Mélange des approfondissements possible
- A définir pour la prochaine séance

Groupes de projet & évaluations

- Groupes de 4 personnes
- Mélange des approfondissements possible
- A définir pour la prochaine séance
- Les notes pourront être individualisées

Groupes de projet & évaluations

- Groupes de 4 personnes
- Mélange des approfondissements possible
- A définir pour la prochaine séance
- Les notes pourront être individualisées

- Soutenances intermédiaires
- Soutenance finale
- Rapport de projet

Groupes de projet & évaluations

- Groupes de 4 personnes
- Mélange des approfondissements possible
- **A définir pour la prochaine séance**
- Les notes pourront être individualisées

- Soutenances intermédiaires
- Soutenance finale
- Rapport de projet
- Prévoyez des fichiers de tests **en quantité suffisante** pour chaque évaluation

Dépôts Git

Pour vos dépôts Git:

- `https://gitlab.telecomnancy.univ-lorraine.fr`
- Ajouter Suzanne Collin, Sébastien Da Silva et Pierre Monnin en *Master*
- Les dépôts seront consultés avec analyses automatiques

Dépôts Git

- Tests semi-automatiques de votre code depuis vos dépôts
- Respecter le template *projet* sur `https://github.com/pmonnin/compilation-lab`
- En particulier :
  - Un Makefile à la racine de votre dépôt
  - Une cible `build`
  - Une cible `run`
  - `System.exit(-1);` en cas d'erreur dans la compilation
  - `System.exit(0);` pour un fichier correct

Suzanne Collin

suzanne.collin@loria.fr

Sébastien Da Silva

sebastien.da-silva@loria.fr

Pierre Monnin

pierre.monnin@loria.fr