Licence MIAGE

Université de Lorraine

Compilation : Analyse syntaxique étendu

Azim ROUSSANALY

Version du
28 mars 2021

Table des matières

1 Introduction

Ce document fait partie du cours de Compilation de la Licence de Mathématiques et Informatique appliquées aux Sciences humaines et sociales (MIASHS) parcours MIAGE de l'Université de Lorraine.

Il s'agit de la deuxième partie du chapitre sur les analyseurs syntaxiques dont l'objectif est d'acquérir les compétences nécessaires en vue de concevoir le module d'analyse syntaxique intégré au projet de réalisation d'un compilateur.

Ce document entend poursuivre l'étude de cas développé dans la première partie du cours. La démarche consiste à présenter progressivement les éléments à rajouter au parser de la première partie afin de l' étendre en vue de créer une structure donnée considérée comme une représentation abstraite du texte analysé. Autrement dit, le résultat attendu du processus de parsing n'est plus seulement un booléen, mais une représentation abstraite du texte en entrée (voir 1.

Il n'y a pas vraiment de représentation abstraite type. Celle-ci dépend fortement des traitements que l'on souhaite effectuer.

Dans le cas d'un compilateur, cette représentation serait évidemment l'arbre abstrait correspondant à un programme-source ainsi que la table des symboles (TDS) associé. Dans le cas de l'étude de cas sur les expression arithmétiques, la représentation abstraite pourrait être tout simplement le résultat de l'évaluation (exemple : 2+2 serait représentée par 4 [1]).

Mais, afin de se rapprocher de la problématique des compilateurs, on va plutôt complexifier un peu plus l'étude de cas en se fixant comme objectif de construire l'arbre d'évaluation de l'expression.

Ainsi si on reprend l'exemple #7 :

```
//exemple #7
```

1. C'est l'exemple que l'on voit présenté dans la plupart des tutoriaux de CUP sur le web

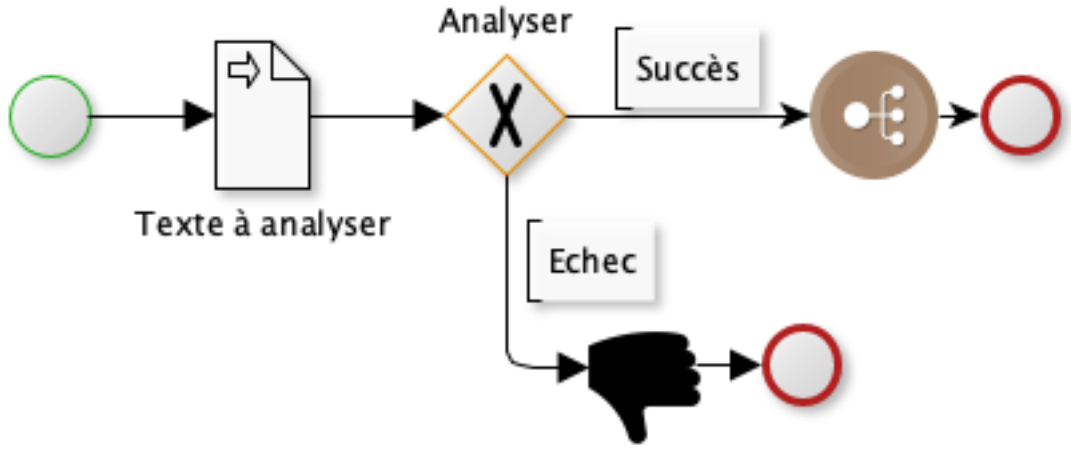
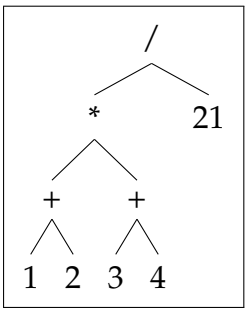


FIGURE 1 – Processus de parsing

```
((1+2)*(3+4))/21
```

on cherche à obtenir l'équivalent de l'arbre suivant :



2 Extension de la grammaire CUP

API arbre

Afin de construire l'arbre d'évaluation, on s'appuie sur l'API JAVA *arbre* (traité en TD) en se limitant aux noeuds de type : Plus, Moins, Multiplication, Division et Cons.

On doit donc importer les classes nécessaires de la manière suivante dans le programme CUP :

```
/*Grammaire CUP du projet Expression*/
package generated.fr.ul.miage.expression2;
import fr.ul.miage.arbre.*;
/* la grammaire */
...
```

Associer une classe JAVA à un non terminal

Cup permet d'associer à un non terminal, une classe JAVA.

Dans le cas qui nous intéresse, les non terminaux *expression*, *facteur* et *atome* sont des instances de la classe *Noeud*.

```
/* 2) non terminaux */
non terminal Noeud expression, facteur, atome;
```

Cela impose de déterminer, pour chacune des règles ayant comme membre droit l'un de ces non terminaux, l'instance de classe *Noeud* "retournée" par celle-ci. Pour cela, on utilise la pseudo-variable *RESULT*.

Pour le non terminal *expression*, cela donne :

```
expression ::= expression:e ADD facteur:f {:
                        RESULT = new Plus();
                        ((Plus)RESULT).setFilsGauche(e);
                        ((Plus)RESULT).setFilsDroit(f);
                              :}
    | expression:e SUB facteur:f {:
                              RESULT = new Moins();
                              ((Moins)RESULT).setFilsGauche(e);
                              ((Moins)RESULT).setFilsDroit(f);
                              :}
    | facteur:f               {: RESULT = f; :}
    ;
```

Explications :

— On peut insérer dans chaque règle, du code JAVA encadré par { : et :} appelé **action**. L'action est exécutée en parallèle de l'analyse syntaxique. Chaque action est activée lorsque le parser arrive au point d'analyse correspondant à son emplacement.

— La pseudo-variable CUP *RESULT* sert à déterminer l'instance associé au non terminal lorsque la règle est vérifiée.
— On peut faire référence, dans les actions, aux objets "retournés" par les différents constituants de la partie droite de la règle. Pour ce faire, on les associe à une variable précédé d'un *:* (exemple : *expression :e*, ici *e* est l'objet correspondant).

En raisonnant de la même façon, pour le non terminal *facteur*, on obtient :

```
facteur ::= facteur:f MUL atome:a  {:
                                    RESULT = new Multiplication();
                                    ((Multiplication)RESULT).
                                        setFilsGauche(f);
                                              ((Multiplication)RESULT).
                                                  setFilsDroit(a);
                                              :}
    | facteur:f DIV atome:a                   {:
                                              RESULT = new Division();
                                              ((Division)RESULT).
                                                  setFilsGauche(f);
                                              ((Division)RESULT).
                                                  setFilsDroit(a);
                                              :}
    | atome:a                                 {: RESULT = a; :}
    ;
```

Associer une classe JAVA à un terminal

Il est aussi possible d'associer une classe à un terminal.
Du côté CUP, on procède de la manière suivante :

```
/* 1) terminaux */
terminal ADD, SUB, MUL, DIV, PO, PF;
terminal Integer NUM;
```

Cette séquence indique que terminal *NUM* est de type *Integer*. On peut l'utiliser comme tel dans les actions.

```
atome ::= NUM:n                           {: RESULT = new Const(n); :}

    | PO expression:e PF                  {: RESULT = e; :}
    ;
```

En conséquence, c'est du côté de JLEX, qu'on "calcule" l'objet *Integer* "retournée" par le terminal *NUM*. Ainsi, dans le programme JLEX, on utilisera un autre constructeur *Symbol(int, Object)* où *Object* représente l'instance associé au terminal.

```
{NUM}          { return new Symbol(Sym.NUM, new Integer(yytext()));}
```

Récupérer l'arbre d'évaluation

On rappelle que CUP génère une classe JAVA, qui dans notre projet s'appelle *ParserCup*. Il est possible d'enrichir cette classe en y rajoutant ses propres variables et methodes. C'est l'objectif de la commande *parse code* que l'on utilise comme suit :

```
/*Grammaire CUP du projet Expression*/
package generated.fr.ul.miage.expression2;
import fr.ul.miage.arbre.*;

 /*code java*/
 parser code {:
     public Noeud resultat = null;
:}
/*-------------------------------------------------------------*/
```

Le champ *resultat* étant public, il devient aisé de récupérer son contenu après le parsing.

Dans le fichier CUP, on affecte la valeur de la variable *resultat* :

```
/*-------------------------------------------------------------*/
/*4) regles de production */
langage ::= expression:e                    {: resultat = e; :}

     ;
```

Voici un exemple d'utilisation, qui s'appuie sur la classe *Afficheur* de l'API *arbre* dans la classe *Main*.

```
     public static void main(String[] args) {
          String filename = null;
```

```
        //1) r cup rer le nom du fichier    traiter
        if (args.length != 1) {
            LOG.severe("Usage : commande nom-de-fichier");
            System.exit(1);
        } else {
            filename = args[0];
        }
       //dmarrer le traitement
       try {
            ParserCup pc = new ParserCup(new Yylex(new FileReader(
                filename)));
            pc.parse();
            Afficheur.afficher(pc.resultat);
            System.out.printf("OK%n");
        } catch (FileNotFoundException e) {
            System.out.printf("pas OK%n");
            LOG.severe(e.getLocalizedMessage());
        } catch (Exception e) {
            System.out.printf("pas OK%n");
            LOG.severe(e.getLocalizedMessage());
            //e.printStackTrace();
        }
        LOG.info("Termin ");
    }
```

Compléter le fichier pom.xml

Le projet utilisant *Maven*, on rajoute la dépendance concerant l'API *artbre* :

```
        <!-- Utiliser API ARBRE -->
        <dependency>
            <groupId>fr.ul.miage</groupId>
            <artifactId>arbre</artifactId>
            <version>1.1</version>
        </dependency>
```

Récapitulation

Le fichier *Parser.cup*

```
/*Grammaire CUP du projet Expression*/
package generated.fr.ul.miage.expression2;
import fr.ul.miage.arbre.*;

/*code java*/
```

```
 parser code {:
     public Noeud resultat = null;
:}
/*-------------------------------------*/
/* la grammaire */
/* 1) terminaux */
terminal ADD, SUB, MUL, DIV, PO, PF;
terminal Integer NUM;
/* 2) non terminaux */
non terminal langage;
non terminal Noeud expression, facteur, atome;
/* 3) Axiome/Start */
start with langage;
/*-------------------------------------*/
/* 4) regles de production */
langage ::= expression:e                    {: resultat = e; :}
      ;
expression ::= expression:e ADD facteur:f     {:
                                                RESULT = new Plus();
                                                ((Plus)RESULT).setFilsGauche(e);
                                                ((Plus)RESULT).setFilsDroit(f);
                                              :}
      | expression:e SUB facteur:f            {:
                                                RESULT = new Moins();
                                                ((Moins)RESULT).setFilsGauche(e);
                                                ((Moins)RESULT).setFilsDroit(f);
                                              :}
      | facteur:f                             {: RESULT = f; :}
      ;
facteur ::= facteur:f MUL atome:a             {:
                                                RESULT = new Multiplication();
                                                ((Multiplication)RESULT).setFilsGauche(f);
                                                ((Multiplication)RESULT).setFilsDroit(a);
                                              :}
      | facteur:f DIV atome:a                 {:
                                                RESULT = new Division();
                                                ((Division)RESULT).setFilsGauche(f);
                                                ((Division)RESULT).setFilsDroit(a);
                                              :}
      | atome:a                               {: RESULT = a; :}
      ;
atome ::= NUM:n                               {: RESULT = new Const(n); :}
      | PO expression:e PF                    {: RESULT = e; :}
      ;
```

Le fichier *Scannee.lex*

```
/*
Scanner pour le parser d'expression de constantes
TD Compil L3 Miage
Azim Roussanaly
(c) Universit de Lorraine
2020
*/
package generated.fr.ul.miage.expression2;
import java_cup.runtime.Symbol;

%%
/* options */
%line
%public
%cupsym Sym
%cup
/* macros */
```

```
SEP      =    [ \n\r]
NUM      =    [0-9]+
COM        =      "//".*\n

%%
/* r gles */
"+"             { return new Symbol(Sym.ADD);}
"-"             { return new Symbol(Sym.SUB);}
"*"             { return new Symbol(Sym.MUL);}
"/"             { return new Symbol(Sym.DIV);}
"("             { return new Symbol(Sym.PO);}
")"             { return new Symbol(Sym.PF);}
{NUM}           { return new Symbol(Sym.NUM, new Integer(yytext()));}
{SEP}           {;}
{COM}           {;}
<<EOF>>         {return new Symbol(Sym.EOF);}
```

Le fichier *Main.java*

```
package fr.ul.miage.expression2;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.logging.Logger;

import fr.ul.miage.arbre.Afficheur;
import generated.fr.ul.miage.expression2.ParserCup;
import generated.fr.ul.miage.expression2.Yylex;

/**
 * Classe principale
 * Utilisation du parser d'expression de constantes
 * - argument unique et obligatoire : le nom du fichier    analyser
 * - r sultat : affichage d'un message
 * @author azim roussanaly
 */
public class Main {

     //Logger pour afficher les messsages de log
     private static final Logger LOG = Logger.getLogger(Main.class.
         getName());

     public static void main(String[] args) {
          String filename = null;
          //1) r cup rer le nom du fichier    traiter
          if (args.length != 1) {
               LOG.severe("Usage : commande nom-de-fichier");
               System.exit(1);
          } else {
```

```
30                filename = args[0];
            }
          //dmarrer le traitement
          try {
                ParserCup pc = new ParserCup(new Yylex(new FileReader(
                    filename)));
35              pc.parse();
                Afficheur.afficher(pc.resultat);
                System.out.printf("OK%n");
            } catch (FileNotFoundException e) {
                System.out.printf("pas OK%n");
40              LOG.severe(e.getLocalizedMessage());
            } catch (Exception e) {
                System.out.printf("pas OK%n");
                LOG.severe(e.getLocalizedMessage());
                //e.printStackTrace();
45          }
            LOG.info("Termin ");
      }
}
```

Le fichier *pom.xml*

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
          apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
5     <groupId>fr.ul.miage</groupId>
      <artifactId>expression2</artifactId>
      <version>0.0.1-SNAPSHOT</version>
      <name>parse expression</name>
      <description>Exemple de parsing  tendu  d'une expression</
          description>
10    <properties>
           <!-- On utilise Java8 -->
           <maven.compiler.source>1.8</maven.compiler.source>
           <maven.compiler.target>1.8</maven.compiler.target>
           <project.build.sourceEncoding>UTF8</project.build.
               sourceEncoding>
15         <!-- O   stocker la distribution binaire -->
           <project.bindist.dir>${project.basedir}/bindist</project.
               bindist.dir>
           <!-- Nom de la classe principale -->
           <project.main.classname>${project.groupId}.${project.artifactId
               }.Main</project.main.classname>
           <!-- Nom de la classe principale -->
20         <project.bin.appname>parse</project.bin.appname>
      </properties>
```

```
    <!-- Junit -->
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
        <!-- Utiliser CUP 0.11b -->
        <dependency>
            <groupId>com.github.vbmacher</groupId>
            <artifactId>java-cup-runtime</artifactId>
            <version>11b-20160615</version>
        </dependency>
        <!-- Utiliser API ARBRE -->
        <dependency>
            <groupId>fr.ul.miage</groupId>
            <artifactId>arbre</artifactId>
            <version>1.1</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <!-- Utiliser le plugin jflex -->
            <!-- usage: mvn compile -->
            <!-- Convention : - le fichier .lex se trouve dans le
                dossier src/main/cuplex
                - Il se nomme Scanner.lex - Il utilise une interface
                    nomm e Sym pour coder
                les terminaux - Le fichier g n r est Yylex.java
                    -->
            <plugin>
                <groupId>de.jflex</groupId>
                <artifactId>jflex-maven-plugin</artifactId>
                <version>1.7.0</version>
                <executions>
                    <execution>
                        <phase>generate-sources</phase>
                        <goals>
                            <goal>generate</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <outputDirectory>src/main/java</outputDirectory>
                    <lexDefinitions>
                        <lexDefinition>src/main/cuplex</
                            lexDefinition>
                    </lexDefinitions>
                </configuration>
            </plugin>
            <plugin>
```

```
                <!-- Utiliser le plugin cup usage: mvn compile -->
                <!-- Conventions : - le fichier - le fichier .cup se
                    trouve dans le dossier
                     src/main/cuplex - il se nomme Parser.cup - il
                         g n re une interface nomm e
                     Sym -->
                <groupId>com.github.vbmacher</groupId>
                <artifactId>cup-maven-plugin</artifactId>
                <version>11b-20160615</version>
                <executions>
                     <execution>
                          <phase>generate-sources</phase>
                          <goals>
                               <goal>generate</goal>
                          </goals>
                     </execution>
                </executions>
                <configuration>
                     <cupDefinition>src/main/cuplex/Parser.cup</
                         cupDefinition>
                     <className>ParserCup</className>
                     <symbolsName>Sym</symbolsName>
                     <outputDirectory>src/main/java</outputDirectory>
                </configuration>
           </plugin>
           <plugin>
                <!-- create a distribution archive -->
                <!-- mvn install ou assembly:single -->
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>3.1.0</version>
                <configuration>
                     <descriptorRefs>
                          <descriptorRef>project</descriptorRef>
                     </descriptorRefs>
                </configuration>
                <executions>
                     <execution>
                          <id>dist</id>
                          <phase>install</phase>
                          <goals>
                               <goal>single</goal>
                          </goals>
                     </execution>
                </executions>
           </plugin>
           <plugin>
                <!-- create scripts for apps -->
                <!-- mvn package | appassembler:assemble -->
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>appassembler-maven-plugin</artifactId>
```

```
                    <version>1.10</version>
                    <configuration>
                        <assembleDirectory>${project.bindist.dir}</
                            assembleDirectory>
                        <programs>
                            <program>
                                <mainClass>${project.main.classname}</
                                    mainClass>
                                <id>${project.bin.appname}</id>
                            </program>
                        </programs>
                    </configuration>
                    <executions>
                        <execution>
                            <phase>package</phase>
                            <goals>
                                <goal>assemble</goal>
                            </goals>
                        </execution>
                    </executions>
                </plugin>
            </plugins>
        </build>
    </project>
```

3 Exercices

Exercice 1

Écrire un parser qui construit une objet JAVA de type *ArrayList<String>* en lisant un fichier de mots séparés par une virgule.

IDMC
Pole Herbert Simon
13, rue M. Ney
54000 Nancy