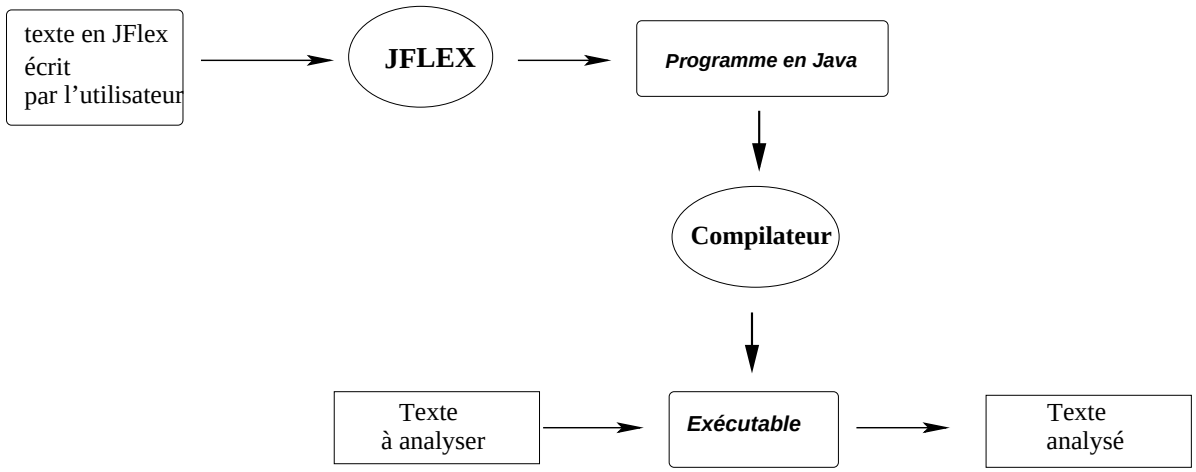
Une introduction au logiciel JFlex

Robert Cori

janvier 2009

1 Principe général

JFlex est un outil logiciel qui permet de générer des analyseurs lexicaux. Pour réaliser un tel analyseur il faut décrire d'abord l'outil à construire dans un texte qu'il est bon de terminer par `.flex` par exemple on peut l'appeler `monanalyseur.flex` , ce texte fourni à **Lex** permet d'obtenir ensuite un analyseur lexical en langage Java. Une fois obtenu cet analyseur sous la forme d'une seule classe de java il faut la compiler et l'on a alors un `.class` que l'on peut exécuter et qui effectue l'analyse lexicale d'un texte suivant les instructions données dans `monanalyseur.lex`. Ce fonctionnement peut être schématisé par la figure suivante :



Les instructions nécessaires à la réalisation sont alors les suivantes

```
java -jar lib/JFlex.jar monanalyseur.lex
javac  -classpath  /usr/share/java Lexer.java
java Lexer < texteAAnalyser > resultat
```

En fait il est plus convenable de réaliser l'ensemble de ces trois opérations au sein d'un `Makefile` comme par exemple:

```
Lexer.java : $(SRCFLEX)
        java -jar $(BJFLEX)  $(SRCFLEX)
%.class : %.java
        javac -classpath $(CLASSPATH) *java
```

```
clean:
        rm -f *.class *~ $(PARSER).java $(LEXER).java sym.java parse_prog
```

qui permet de générer un analyseur à partir des valeurs de `BJFLEX` l'emplacement où se trouve le logiciel `JFlex`, de `CLASSPATH` le chemin d'accès aux classes exécutables nécessaires et du texte `SRCFLEX` écrit en Jlex.

On se propose ici de décrire en détail comment construire un texte `monanalyseur.flex` que nous appelerons un "programme en `JFlex`".

2 Organisation d'un programme en `JFlex`

Un programme en **JFlex** se divise en trois parties séparées par des `%%`. La première contient des définitions initialisations et classes en Java, utiles pour le programme généré ; ce qui est écrit là sera inséré tel quel dans ce programme généré.

La seconde contient des options, des définitions d'expressions régulières et des déclarations.

La troisième contient les règles à appliquer lors de la lecture du texte à analyser.

2.1 Options et définitions

Le programme généré comportera une seule classe en Java dont le nom doit être indiqué par l'option :

`%class` *nom à choisir*

D'autres options sont décrites dans le manuel `JFlex`, notons celle qui indique la présence d'une analyse syntaxique (`%cup`) après l'analyse lexicale, ou pas d'analyse à la suite (`%standalone`).

On peut dans cette partie du texte JLex, donner des initialisations de variables statiques où des fonctions et méthodes en Java à insérer dans la classe, noter que celles données dans la première partie sont inséreés en dehors de la classe.

Les définitions s'effectuent en indiquant, un nom pour le langage décrit par l'expression rationnelle suivi d'un signe `=` et de la description de cette expression en utilsant les conventions décrites dans le paragraphe ci-dessous.

2.2 Règles et actions

C'est le cœur du programme **JFlex**, il s'agit ici d'indiquer une suite de couples formés d'une exprssion régulière et d'une action. Lorsqu'un mot contiendra une expression donnée dans cette suite l'analyseur accomplira l'action se trouvant sur la ligne définissant cette expression.

2.3 Un premier exemple

Un exemple simple est le suivant, c'est un analyseur qui lit une suite de `0` et de `1` ; il transforme `0` en `a` s'il est précédé d'un nombre pair de `1` et en `b` s'il est précédé d'un nombre impair de `1` de même il remplace les `1` en `c` ou `d` s'ils sont précédés d'un nombre pair ou impair de `0`.

```
mport java.io.*;



%%
```

```
%class Lexer
%line
%column
%standalone

%{
public static int nb0 = 0, nb1 = 0;

%}
%%
0 {if (nb1 %2  == 0) System.out.print ("a"); else System.out.print("b"); nb0++ ;}
1 {if (nb0 %2  == 0) System.out.print ("c"); else System.out.print("d"); nb1++ ;}
```

3 Le fonctionnement du filtrage

Si le cœur d'un programme `JFlex` contient une suite de lignes comportant une expression rationnelle et une action sous la forme suivante :

R1	A1
R2	A2
..	..
Ri	Ai
..	..
Rn	An

L'analyseur considère le texte à analyser comme un mot `f` il cherche le facteur gauche le plus long de ce mot qui appartient à l'un des `Ri`, on a ainsi $f = gh, g \in R_i$ il effectue alors l'action `Ai` et recommence le procédé avec h. Si le facteur gauche g appartient à deux langages `Ri` et `Rj` c'est l'action correspondante au plus petit des deux indices `i, j` qui est effectuée. Si aucun facteur gauche non vide appartient à l'un des `Ri` alors une lettre de f est lue puis recopiée en sortie. Le processus est ensuite répété jusqu'à la fin du texte à analyser.

Remarque : *Aussi bien dans la description des actions que dans le programme en Java,* `yytext()` *de type* `String` *représente le mot reconnu*

4 Expressions rationnelles en Lex

```
x
     filtre le  caractère ‘x’


.
     tout  caractère (byte) sauf newline

[xyz]
     un  "ensemble de caractères"; ici, l’expression  filtre ou bien un
     ‘x’, ou un  ‘y’,ou un  ‘z’

[abj-oZ]
     un  "ensemble da caractères" contenant un intervalle; filtre un ‘a’, un ‘b’, ou une lett
```

```
[^A-Z]
     le complément d'un "ensemble de caractères", i.e., un caractère
     qui n'est pas dans l'ensembe entre []. Ici tout  caractère
     SAUF une majuscule.

[^A-Z\n]
     tout  caractère  sauf une majuscule ou un  newline

R*
     zero une ou plusieurs  R, où R est une  expression rationnelle

R+
     une ou plusieurs R

R?
     zero ou une R

     exactement  4 R

{NAME}
     l'expression  "NAME" definie dans la première partie

"[xyz]\"foo"
     la chaîne de caractères: `[xyz]"foo'

\X
     si  X est un  `a', `b', `f', `n', `r', `t', or `v', alors
     l'interprétation  ANSI de  \X. Sinon, un literal `X' (utilisé pour des
     operateurs comme `*')

\123
     le caractère de valeur  octale  123

\x2a
     le caractère de valeur  hexadécimale  `2a'


(R)
     filtre un R; les parentheses servent à ne pas tenir compte des
     règles de précédence.

RS
     l'expression R suivie par l'expression S; c'est la  "concaténation"

R|S
     une expression R ou  S; c'est l'union"

R/S
     une  R si et seulement si elle est suivie par l'expression S.
```

```
     Le texte filtré par S est inclus pour déterminer quelle est la plus
     longue règle applicable, mais est repris pour l'etape de reconnaissance
     suivante.

^R
     une R, si elle  est au début d'une ligne (i.e., en tout début du texte
    à analyser, ou juste après un newline ).

~R   filtre tout jusqu'à trouver pour la première fois une expression
dans R.
```

5 Exemples

5.1 Suppression des lignes blanches

```
\n/\n  {}
```

5.2 Code à délai non borné

On souhaite décoder une suite de 0 et de 1 provenant du codage de a en 0, b en 01 et c en 11. On note que le décodage s'effectue en examinant le nombre de 1 entre deux 0 : si ce nombre est pair c'est que le 0 provient d'un a, si ce nombre est impair il faut décoder le 01 par b soit en JFlex :

```
%
11 {System.out.print("c");}
01/(11)* {System.out.print("b");}
0/(11)*  {System.out.print("a");}
0/((11)*0)  {System.out.print("a");}
01/((11)*0) {System.out.print("b");}
```

5.3 Un programme de comptage

Ce programme compte le nombre de lettres majuscules, miniscules et autres caractères d'un texte. Le résultat est affiché lorsque la lecture du texte est terminée. On utilise pour indiquer des actions en fin de lecture l'écriture d'instructions en java entre `%eof{` et `%eof}`.

```
import java.io.*;

%%
%class Lexer
%standalone
%{
static int minis = 0, majus = 0,  autres =0, blancs =0;
%}
```

```
%eof{
System.out.println( minis + " " + majus + " "+ autres
  +  "  " + blancs);
%eof}

%%
[a-z] {minis++;}
[A-Z] {majus++;}
[\ ] {blancs++;}
\n {}
. {autres++;}
```

5.4 Analyseur Lexical Java

Voici un analyseur lexical très simplifié de java.

```
CHIFFRE = [0-9]
LETTRE = [a-z]|[A-z]
MOTRESERVE = abstract|double|int|static|boolean|else|interface|super|break|
extends|long|switchbyte|final|native|synchronized|case|finally|new|this|
catch|float|null|import|return|throw|char|for|package|throws|class|
goto|volatile|do|private|transient|const|if|protected|try|continue|
implements|public|void|default|instanceof|short|while
OPERATEUR = \<|\=|\+|<=|\*
SEPARATEUR = [\{\}\[\];,.)]

%%
(\/\*)({LETTRE}|\ )*(\*\/) {System.out.print("\n COMMENTAIRE \n");}
{MOTRESERVE}               {System.out.print("MR" +yytext() );}
{OPERATEUR}                {System.out.print("OP" +yytext());}
{SEPARATEUR}               {System.out.print("SEP" +yytext());}
{LETTRE}({LETTRE}|{CHIFFRE})*\(
                           {System.out.print(" id-fonction" +yytext());}
{LETTRE}({LETTRE}|{CHIFFRE})*
                           { System.out.print( "id ");}
{CHIFFRE}*                 {System.out.print( "CONST-INT ");}
\"{LETTRE}*\"              {System.out.print( "CONST-STRING ");}
```

Un programme Java

```
class Factorielle {
   /* affiche la suite des factorielles  */
  public static void main(String [] args) {
    long x = 1;
    int n1 = 0;
    int max = 30 ;
    while (n1 < max) {
      System.out.println(n1 + "vaut" +x) ;
      n1 = n1 +  1;
      x = x * n1;
    }
    System.out.println(max + "vaut"+ x) ;
  }
}
```

Transformation du programme par l'analyseur précédent

```
MRclass id  SEP{


 COMMENTAIRE


  MRpublic MRstatic MRvoid  id-fonctionmain(id  id  id SEP) SEP{

    MRlong id  OP= CONST-INT SEP;
    MRint id  OP= CONST-INT SEP;
    MRint id  OP= CONST-INT  SEP;

    MRwhile (id  OP< id SEP) SEP{
      id SEP.id SEP. id-fonctionprintln(id  OP+ " --> " OP+id SEP) SEP;
      id  OP= id  OP+  CONST-INT SEP;
      id  OP= id  OP* id SEP;
      SEP}

    id SEP.id SEP. id-fonctionprintln(id  OP+  CONST-STRING OP+ id SEP) SEP;
  SEP}

SEP}

 w
```

6 Automates en JFlex

On peut faire effectuer par JFlex un comportement de type automate. Il faut pour cela définir des états et dire que certaines actions ne sont à réaliser que si un certain état est atteint. Dans le manuel les états sont appelés *Start conditions*.

Pour définir un état on indique dans la partie définition :

```
%state NomEtat
```

Noter que l'état YYINITIAL n'a pas besoin d'être déclaré, c'est l'état dans lequel se trouve l'automate au début du programme.

Pour mettre le programme dans l'état NomEtat on utilise la fonction :

```
yybegin(NomEtat)
```

.

Pour dire qu'une action ne s'applique que si le programme se trouve dans l'état NomEtat on indique devant une règle le nom de l´etat entre <>. Par exemple le 1 est transformé en 0 si on est dans l'état S1, s´écrit :

```
<S1>1 {System.out.print('0');}
```

Un exemple d'automate qui effectue la même transformation que dans l'exemple donné page 2 est le suivant

```
%state PairImpair ImpairPair ImpairImpair


%%
<YYINITIAL> 0        {System.out.print ("a"); yybegin(ImpairPair);}
<YYINITIAL> 1        {System.out.print ("c"); yybegin(PairImpair);}
<ImpairPair> 0     {System.out.print ("a"); yybegin(YYINITIAL);}
<ImpairPair> 1     {System.out.print ("d"); yybegin(ImpairImpair);}
<PairImpair> 0     {System.out.print ("b"); yybegin(ImpairImpair);}
<PairImpair> 1     {System.out.print ("c"); yybegin(YYINITIAL);}
<ImpairImpair> 0   {System.out.print ("b"); yybegin(PairImpair);}
<ImpairImpair> 1   {System.out.print ("d"); yybegin(ImpairPair);}
```

7 Compléments

Il est conseillé de lire le manuel JFlex pour des informations sur les performances du programme d'analyse créé : rapidité, taille de l'automate obtenu.