

Analyse Lexicale

Analyse Lexicale

CS410 - Langages et Compilation

Julien Henry
Catherine Oriat

Grenoble-INP Esisar

2012-2013

Lexicographie : décrit la forme des mots du langage.

Analyse Lexicale : transforme une suite de *caractères* en une suite de *mots*.

Les mots peuvent être décrit par des *Langages réguliers*.

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 1 / 23 >

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 2 / 23 >

programme source (suite de caractères)
exemple : `val := val + 1 ;`

Analyse lexicale

Erreurs lexicales

suite de mots

exemple : `idf(val) affect idf(val) plus num(1) pv`

Analyse syntaxique

Erreurs syntaxiques

arbre abstrait

```
graph TD
    Affect --> Idf1["Idf (val)"]
    Affect --> Plus
    Plus --> Idf2["Idf (val)"]
    Plus --> Entier["Entier 1"]
```

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 3 / 23 >

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 4 / 23 >

Rôle de l'Analyse Lexicale

- Supprime les espaces, retours à la ligne, tabulations, commentaires
- identifie les *mots réservés* et les *identificateurs*.

Ces mots sont appelés les *lexèmes* (ou *token*).

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 3 / 23 >

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 4 / 23 >

Exemple

Un exemple de programme en Jcas :

```
delta := b*b - 4*a*c;
```

```
idf(delta) affect idf(b) mult idf(b)
moins num(4) mult idf(a) mult idf(c)
```

Lexèmes : `idf`, `affect`, `mult`, `moins`, `num`

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 5 / 23 >

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 6 / 23 >

Vocabulaire Terminal

L'ensemble des lexèmes forme le vocabulaire terminal du langage.

Exemple :

```
Exp  →  Exp plus Terme
       | Exp moins Terme
       | Exp mult Terme
       | Terme
Terme →  idf | num | ( Exp )
```

Identificateurs en Jcas

Lettre = { 'a', 'b', ..., 'z', 'A', ..., 'Z' }
Chiffre = { '0', '1', ..., '9' }
Idf = Lettre (Lettre + Chiffre + '_')*

On utilise des langages *réguliers* : cela permet d'obtenir des algorithmes de reconnaissance efficaces.

Reconnaisseur de Langages Réguliers

Definition (Reconnaisseur)

Un reconnaisseur pour L est un programme prenant en entrée une chaîne c et qui retourne $c \in L$.

Pour écrire un reconnaisseur, on utilise un automate *déterministe*, si possible *minimal*.

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 7 / 23 >

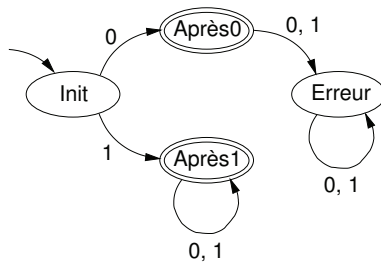
Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 8 / 23 >

Exemple

Automate reconnaisseur de l'expression régulière : $0 + 1(0 + 1)^*$.



Algorithme reconnaisseur

Il existe 3 méthodes automatiques pour programmer un algorithme reconnaisseur à partir d'un automate :

- utilisation d'une variable d'état, codage des transitions
- codage des transitions par un tableau
- codage avec des étiquettes et des branchements

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 9 / 23 >

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 10 / 23 >

Variable d'état, codage des transitions

- On définit un type Etat de l'automate.
- On utilise une variable de type état pour définir l'état courant.
- On code les transitions de l'automate avec un grand "switch" sur la valeur de l'état courant.

Variable d'état, codage des transitions

```
public enum Etat {Init, Apres0, Apres1, Erreur;}
public reconnaisseur {
    Etat E = Init; char C = lire_car();
    while (C != '#') {
        switch (Etat) {
            case Init:
                if (C == '0') E = Apres0; else E = Apres1;
                break;
            case Apres0:
                E = Erreur;
            default: break;
        }
        C = lire_car();
    }
    if (!(E == Apres0 || E == Apres1))
        {System.out.println("ERREUR");}
}
```

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 11 / 23 >

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 12 / 23 >

Codage par label et branchements

- Chaque état de l'automate est codé par un label.
- Chaque transition de l'automate est codée par une instruction `goto`

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 13 / 23 >

Codage par label et branchements

```
void reconnaisseur () {
    char c;
    bool ok;

    Init:
        c = lire_car();
        if (c=='0') goto Apres0;
        if (c=='1') goto Apres1;
        ok = false; goto Fin;

    Apres0:
        c = lire_car();
        if (c=='#') {ok = true; goto Fin};
        goto Erreur;

    Apres1:
        ...

    Fin:
        if (ok) printf("chaîne reconnue")
        else   printf("chaîne non reconnue")
}
}
```

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 14 / 23 >

Ecriture d'un analyseur lexical

Definition (analyseur lexical)

Un analyseur lexical est un programme qui lit des caractères en entrée et fournit des lexèmes (token).

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 15 / 23 >

Propriétés de l'analyseur

- On veut reconnaître le préfixe le plus long possible :
d, *de*, *del*, *delt* sont des identificateurs possibles, mais on veut obtenir l'identificateur *delta*.
- Enchaînement de plusieurs unités lexicales :
La fin d'une unité lexicale est détectée soit par un séparateur (espace), soit par le début de l'unité lexicale suivante.
Exemple : $b*b - 4*a*c$
- Détection des mots réservés :
Il faut reconnaître les mots réservés (*begin*, *end*, etc.). On construit une table contenant tous ces mots ainsi que son code associé, et on recherche si le mot lu appartient à cette table.

Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 16 / 23 >

Lexème

L'analyse lexicale fournit le type de lexème (*begin*, *end*, *idf*, etc), mais aussi des informations supplémentaires.

```
public enum Code_lex {Begin_lex, Affect, Num, ...}

public class Lexeme {
    // code du lexeme
    Code_lex code;
    // chaîne correspondant à l'unité lexicale
    String chaîne;
    // numeros de ligne et colonne du lexeme
    int num_ligne, num_colonne;
}
```

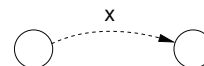
Grenoble-INP Esisar

Analyse Lexicale

2012-2013 < 17 / 23 >

Extension des automates

Pour traiter l'enchaînement de plusieurs unités lexicales sans séparateur, on introduit des transitions dans l'automate qui ne consomment pas le symbole courant.



Grenoble-INP Esisar

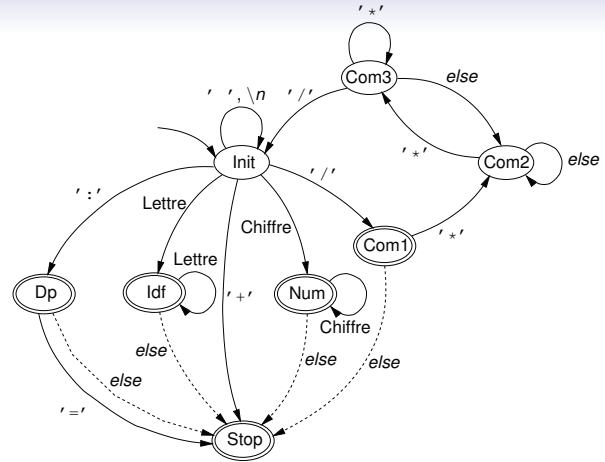
Analyse Lexicale

2012-2013 < 18 / 23 >

Exemple

On considère le langage de lexicographie :

- les commentaires commencent par `/*` et finissent par `*/`
- on a les signes `:`, `:=`, `/` et `+`
- les séparateurs sont l'espace, le retour à la ligne et les commentaires
- les identificateurs sont définis par $Idf = Lettre^+$
- les entiers sont définis par $Num = Chiffre^+$



Exemple (Suite)

unité lexicale : chemin entre l'état initial *Init* et un état acceptant.

L'état acceptant est *Stop*, mais aussi *Dp*, *Idf*, *Num* et *Com1* si il n'y a pas de caractère suivant (fin du fichier).

On ajoute un lexème d'erreur, qui est renvoyé si aucun lexème n'est reconnu.

```
int Num_LC, Num_CC; char C; Lexeme LC;
public void lex_suiv() {
    boolean Passer_au_suivant; Etat cour = Init;
    while (cour != Stop && cour != Erreur && !EOF()) {
        switch (Etat_cour) {
            case Init: ...
            case Dp: ...
            case Com1: ... Mise a jour de cour
            case Com2: ... selon la valeur de C
            case Com3: ...
            case Idf: ...
            case Num: ...
            default : break;
        }
        if (Passer_au_suivant) {Car_Suiv();}
    }
    if (EOF()) {
        if (cour==Init) {LC.code = Eof}
        else if (cour==Com2||cour==Com3) {LC.code=Erreur}
    }
}
```

En pratique

En pratique, de nombreux outils existent pour générer automatiquement des analyseurs lexicaux.

Exemple :

- C : flex
- Java : jflex
- Ocaml : ocamllex
- ...

```
%{
#include <stdlib.h>
#include "global.h"
%}
blancs    [ \t ]+
chiffre   [0-9]
entier    {chiffre}+
exposant  [eE][+-]?{entier}
reel      {entier}("."{entier})?{exposant}?
%%
{blancs}  { /* On ignore */ }
{reel}    { yylval=atof(yytext);
             return(NOMBRE); }
"+"       return(PLUS);
"-"       return(MOINS);
"*"       return(FOIS);
"/"       return(DIVISE);
"^"       return(POUISSANCE);
"("       return(PARENTHESE_GAUCHE);
")"       return(PARENTHESE_DROITE);
"\n"      return(FIN);
```