

Analyseurs II

INFO010 – Théorie des langages – Partie pratique

Sébastien Collette

Gilles Geeraerts



Problèmes de prévision

- Lors du précédent TP, on avait vu qu'il n'était pas toujours facile de choisir l'action à effectuer dans l'analyseur.
 - Analyseur gauche : quelle règle choisir pour faire une expansion ?
 - Analyseur droit : faut-il faire un *shift* ou un *reduce* ?
- Une solution : essayer **toutes** les possibilités (*backtracking*)...
- Plus intelligemment : tenter de **prévoir** la règle à appliquer en regardant «**en avant**» dans l'*input*.

Notion de *First*

- Soit une forme phrasée $\alpha \in (V \cup T)^*$. On définit $\text{First}^k(\alpha)$ comme l'ensemble des k premiers caractères des *strings* que l'on peut produire à partir de α .
- Formellement :

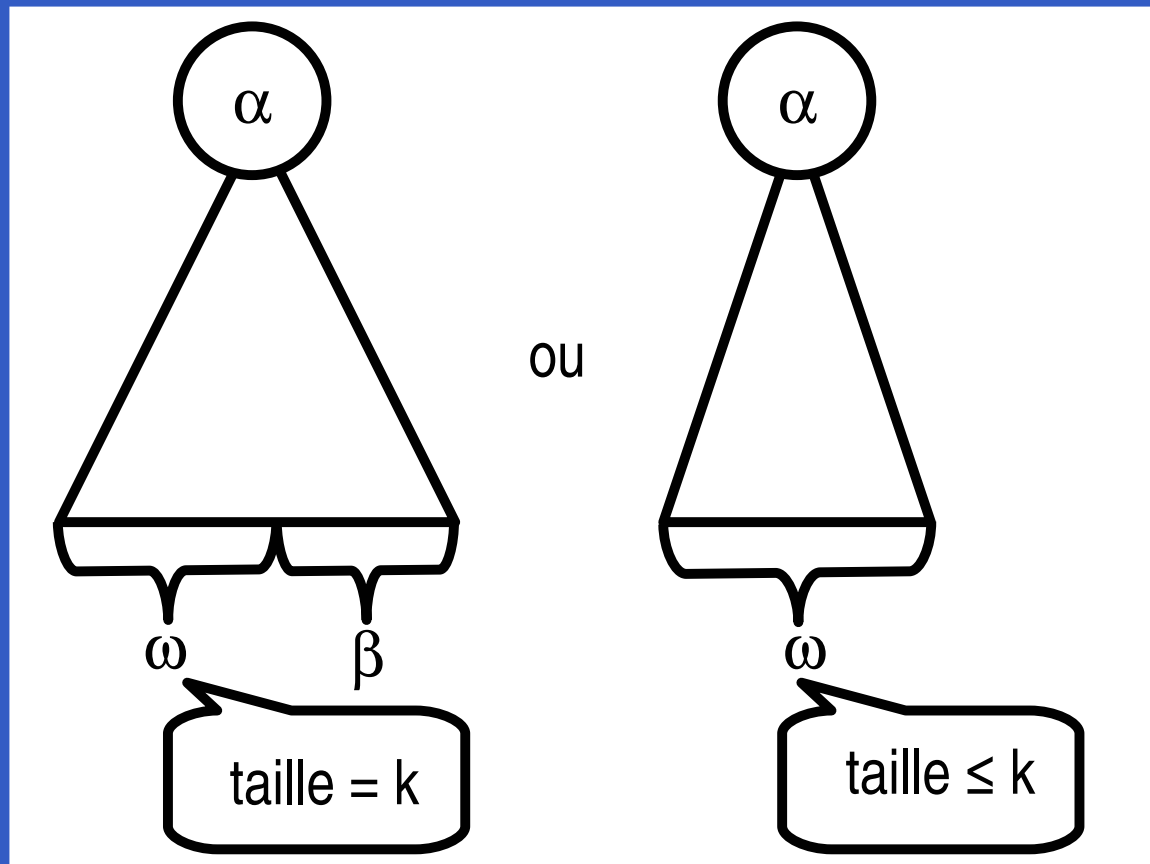
$$\forall \alpha \in (V \cup T)^* : \text{First}^k(\alpha) =$$

$$\{\omega \in T^* \mid \alpha \xRightarrow{*} \omega\beta\}$$

$$\wedge ((|\omega| = k \wedge \beta \in T^*) \vee (|\omega| < k \wedge \beta = \varepsilon))\}$$

First – Illustration intuitive

Pour le $\text{First}^k(\alpha)$:



Notion de *First* – Algorithme de calcul

begin

pour chaque $a \in T$ **faire** $F_0(a) \leftarrow \{a\}$;

pour chaque $A \in V$ **faire**

$F_0(A) \leftarrow \{\omega \in T^* \mid A \rightarrow \omega\beta \wedge ((|\omega| = k \wedge \beta \in T^*) \vee (|\omega| < k \wedge \beta = \varepsilon))\}$;

$i \leftarrow 0$;

répéter

pour chaque $A \in V$ **faire**

$F_i(A) \leftarrow F_{i-1}(A) \cup \{\omega \in T^* \mid A \rightarrow X_1X_2 \cdots X_n \wedge \omega \in (F_{i-1}(X_1) \oplus^k F_{i-1}(X_2) \oplus^k \cdots \oplus^k F_{i-1}(X_n))\}$;

$i \leftarrow i + 1$;

jusqu'à $F_i(A) = F_{i-1}(A)$;

Notion de *Follow*

- Soit une variable A . On définit $\text{Follow}^k(A)$ comme l'ensemble des k premiers caractères des *strings* qui peuvent suivre la production de A .
- Comment on le calcule :

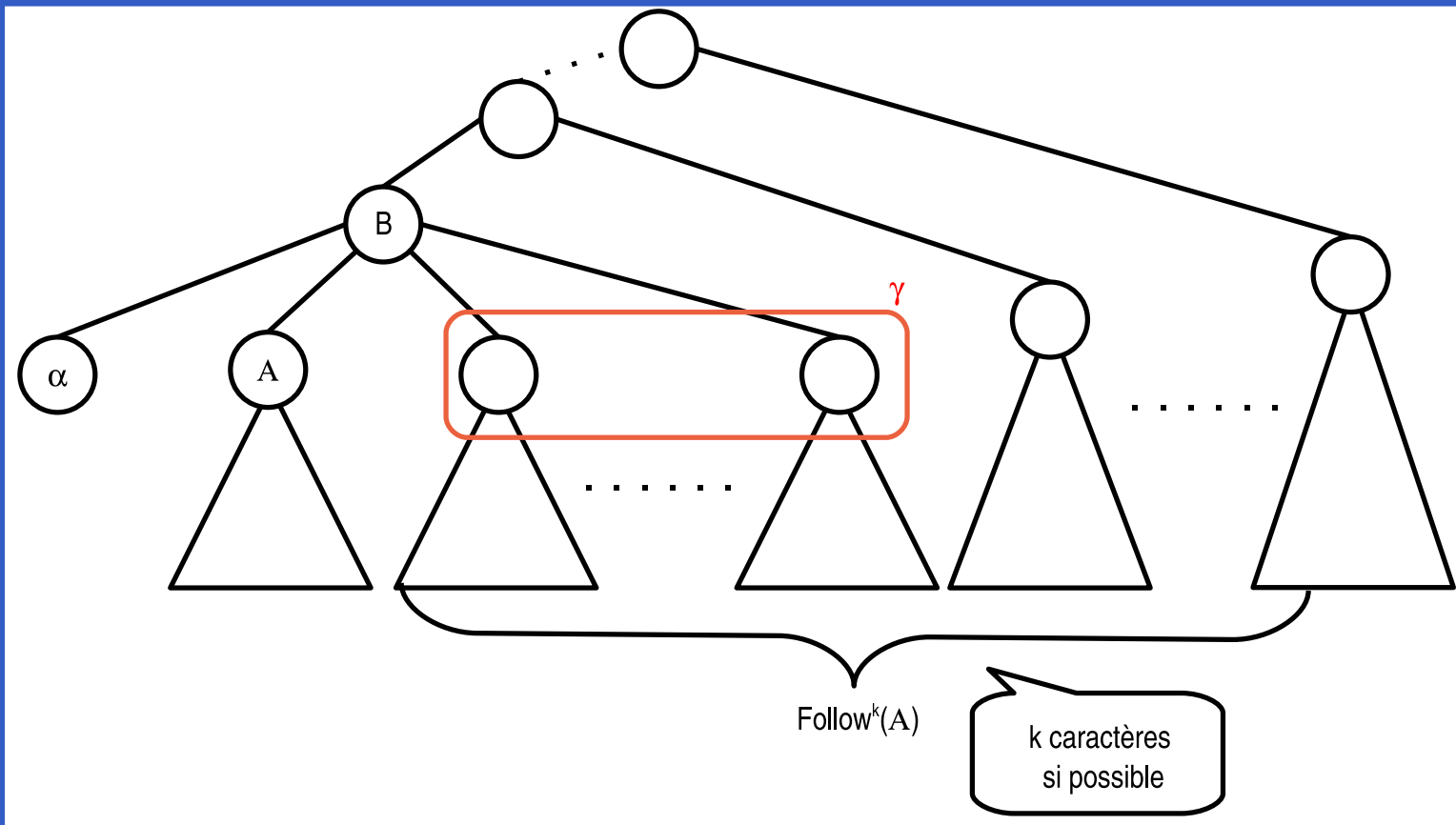
$$\forall A \in V \setminus \{S\} : \text{Follow}^k(A) =$$

$$\bigcup_{B \rightarrow \alpha A \gamma} \{\text{First}^k(\gamma) \oplus^k \text{Follow}^k(B)\}$$

$$\bigcup (\text{if } S \stackrel{+}{\Rightarrow} \alpha A \text{ then } \{\varepsilon\} \text{ else } \emptyset)$$

Follow – Illustration intuitive

Pour le $\text{Follow}^k(A)$:



Notion de *Follow* – Algorithme

begin

pour chaque $A \in V$ **faire** $\text{Follow}^k(A) \leftarrow \emptyset$;

répéter

si $B \rightarrow \alpha A \beta \in P$ **alors**

$\text{Follow}^k(A) \leftarrow$

$\text{Follow}^k(A) \cup \{\text{First}^k(\beta) \oplus^k \text{Follow}^k(B)\}$;

jusqu'à *Stabilité*;

end

Exercice 1

| | | |
|----|------------------|--|
| 1 | <program> | → begin <statement list> end |
| 2 | <statement list> | → <statement> <statement tail> |
| 3 | <statement tail> | → <statement> <statement tail> |
| 4 | <statement tail> | → λ |
| 5 | <statement> | → ID := <expression> ; |
| 6 | <statement> | → read (<id list>) ; |
| 7 | <statement> | → write (<expr list>) ; |
| 8 | <id list> | → ID <id tail> |
| 9 | <id tail> | → , ID <id tail> |
| 10 | <id tail> | → λ |
| 11 | <expr list> | → <expression> <expr tail> |
| 12 | <expr tail> | → , <expression> <expr tail> |
| 13 | <expr tail> | → λ |
| 14 | <expression> | → <primary> <primary tail> |
| 15 | <primary tail> | → <add op> <primary> <primary tail> |
| 16 | <primary tail> | → λ |
| 17 | <primary> | → (<expression>) |
| 18 | <primary> | → ID |
| 19 | <primary> | → INTLIT |
| 20 | <add op> | → + |
| 21 | <add op> | → - |
| 22 | <system goal> | → <program> \$ |

Donnez le $\text{First}^1(A)$ et le $\text{Follow}^1(A)$ pour tout $A \in V$.

Donnez le $\text{First}^2(\langle \text{expression} \rangle)$ et le $\text{Follow}^2(\langle \text{expression} \rangle)$.

Correction de l'exercice 1

| Symbol | First() | Follow() |
|----------------|--------------------------|-------------------|
| system goal | begin | |
| program | begin | \$ |
| statement list | ID read write | end |
| statement tail | ID read write ϵ | end |
| statement | ID read write | ID read write end |
| id list | ID |) |
| id tail | , ϵ |) |

Correction de l'exercice 1 – 2

| Symbol | First() | Follow() |
|--------------|----------------|------------------|
| expr list | (ID INTLIT |) |
| expr tail | , ϵ |) |
| expression | (ID INTLIT | <i>i</i> ,) |
| primary tail | + - ϵ | <i>i</i> ,) |
| primary | (ID INTLIT | + - <i>i</i> ,) |
| add op | + - | (ID INTLIT |

Correction de l'exercice 1 – 3

- $\text{First}^2(\text{expression}) = \{ ((, (\text{ID}, (\text{INTLIT}, \text{ID}+, \text{ID}-, \text{INTLIT}+, \text{INTLIT}-, \text{ID}, \text{INTLIT} \}$
- $\text{Follow}^2(\text{expression}) = \{ , (, , \text{ID}, , \text{INTLIT},) ; ,)+,)-,) , ,)), \text{i read}, \text{i write}, \text{i ID}, \text{i end} \}$

Grammaires LL(1)

- Une grammaire LL(1) est une grammaire qui peut être reconnue par un **analyseur gauche** avec **un seul symbole de prévision**.
- Plus formellement :

Une grammaire G est LL(1) si et seulement si :
Pour toute production $A \rightarrow \alpha_1$ et $A \rightarrow \alpha_2$ ($\alpha_1 \neq \alpha_2$)
on a :

$$\text{First}^1(\alpha_1 \text{Follow}^1(A)) \cap \text{First}^1(\alpha_2 \text{Follow}^1(A)) = \emptyset$$

Table des actions

- Avec une grammaire LL(1), on peut donc facilement construire une **table des actions**.
- Celle-ci donne l'action à effectuer en fonction du symbole au sommet du *stack* et du caractère de prévision.

$$(1) \quad S \rightarrow aS$$

$$(2) \quad \rightarrow b$$

| | a | b | c | ... |
|---|----|----|---|-----|
| S | P1 | P2 | × | ... |
| a | M | × | × | ... |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

Table des actions – Algorithme

begin

$M \leftarrow \times$;

pour chaque $A \rightarrow \alpha$ **faire**

pour chaque $a \in First^1(\alpha)$ **faire**

$M[A, a] \leftarrow M[A, a] \cup Produce(A \rightarrow \alpha)$;

si $\varepsilon \in First^1(\alpha)$ **alors**

pour chaque $a \in Follow^1(A)$ **faire**

$M[A, a] \leftarrow M[A, a] \cup Produce(A \rightarrow \alpha)$;

pour chaque $a \in T$ **faire** $M[a, a] \leftarrow Match$;

$M[\$, \varepsilon] \leftarrow Accept$;

end

Exercice 2

Lesquelles de ces grammaires sont LL(1) ?

1

$$S \rightarrow ABBA$$

$$A \rightarrow a|\varepsilon$$

$$B \rightarrow b|\varepsilon$$

2

$$S \rightarrow aSe|B$$

$$B \rightarrow bBe|C$$

$$C \rightarrow cCe|d$$

3

$$S \rightarrow ABc$$

$$A \rightarrow a|\varepsilon$$

$$B \rightarrow b|\varepsilon$$

4

$$S \rightarrow Ab$$

$$A \rightarrow a|B|\varepsilon$$

$$B \rightarrow b|\varepsilon$$

Exercice 3

Construisez la table des actions pour :

- 0 S \rightarrow Expr\$
- 1 Expr \rightarrow - Expr
- 2 Expr \rightarrow (Expr)
- 3 Expr \rightarrow Var ExprTail
- 4 ExprTail \rightarrow - Expr
- 5 ExprTail \rightarrow ε
- 6 Var \rightarrow ID VarTail
- 7 VarTail \rightarrow (Expr)
- 8 VarTail \rightarrow ε

Correction de l'exercice 2

- 1 : pas LL(1) car $a \in \text{Follow}^1(A)$ et $a \in \text{First}^1(A)$.
Donc $M[A, a] = \{\text{Produce 2, Produce 3}\}$.
- 2 : LL(1).
- 3 : LL(1).
- 4 : pas LL(1) car $b \in \text{First}^1(A)$ et $b \in \text{Follow}^1(B)$.
Donc $M[A, b] = \{\text{Produce 3, Produce 4}\}$.

Correction de l'exercice 3

| | - | (|) | ID | \$ |
|----------|---|---|---|----|----|
| S | 0 | 0 | × | 0 | × |
| Expr | 1 | 2 | × | 3 | × |
| ExprTail | 4 | × | 5 | × | 5 |
| Var | × | × | × | 6 | × |
| VarTail | 8 | 7 | 8 | × | 8 |

Descente récursive

Comment programmer un analyseur gauche ?

- On crée une fonction par variable.
- Chacune de ces fonctions fait appel à `next_token()` pour lire sur l'*input*.
- En fonction du *token* lu (= prévision), elle choisit la suite d'actions à effectuer :
 - Appel(s) à `match(token)...`
 - Appel(s) à une fonction correspondant à une autre règle (= *produce*).
- Une fonction `syntax_error()` est aussi disponible.

Descente récursive – Exemple

```
statement → id := expr ; | read (id list) ; | write (expr list) ;  
void statement(void) {  
    token tok = next_token() ;  
    switch(tok) {  
    case ID:  
        match(ID) ; match(ASSIGNOP) ; expression() ; match(SEMICOLON) ;  
        break ;  
    case READ:  
        match(READ) ; match(LPAREN) ; id_list() ; match(RPAREN) ;  
        match(SEMICOLON) ; break ;  
    case WRITE:  
        match(WRITE) ; match(LPAREN) ; expr_list() ; match(RPAREN) ;  
        match(SEMICOLON) ; break ;  
    default:  
        syntax_error(tok) ; break ;  
    }  
}
```

Exercice 4

| | | |
|----|------------------|--|
| 1 | <program> | → begin <statement list> end |
| 2 | <statement list> | → <statement> <statement tail> |
| 3 | <statement tail> | → <statement> <statement tail> |
| 4 | <statement tail> | → λ |
| 5 | <statement> | → ID := <expression> ; |
| 6 | <statement> | → read (<id list>) ; |
| 7 | <statement> | → write (<expr list>) ; |
| 8 | <id list> | → ID <id tail> |
| 9 | <id tail> | → , ID <id tail> |
| 10 | <id tail> | → λ |
| 11 | <expr list> | → <expression> <expr tail> |
| 12 | <expr tail> | → , <expression> <expr tail> |
| 13 | <expr tail> | → λ |
| 14 | <expression> | → <primary> <primary tail> |
| 15 | <primary tail> | → <add op> <primary> <primary tail> |
| 16 | <primary tail> | → λ |
| 17 | <primary> | → (<expression>) |
| 18 | <primary> | → ID |
| 19 | <primary> | → INTLIT |
| 20 | <add op> | → + |
| 21 | <add op> | → - |
| 22 | <system goal> | → <program> \$ |

Écrivez un analyseur en descente récursive pour les règles 14 à 21 de cette grammaire.