ALF

Analyseur (Parser)

Bibliographie pour aujourd'hui

ALF
language crunch

Keith Cooper, Linda Torczon, *Engineering a Compiler*

- Chapitre 3
  - 3.3

Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, *Compilers: Principles, Techniques, and Tools (2nd Edition)*

- Chapitre 4
  - 4.4

Contenu

ALF
language crunch

- Les analyseurs (parsers)
- LL

Alexander Aiken

- Américain
- Stanford
- LL(*)
- MOSS
- ANTLR

Slides

Partie de slides sont écrie par

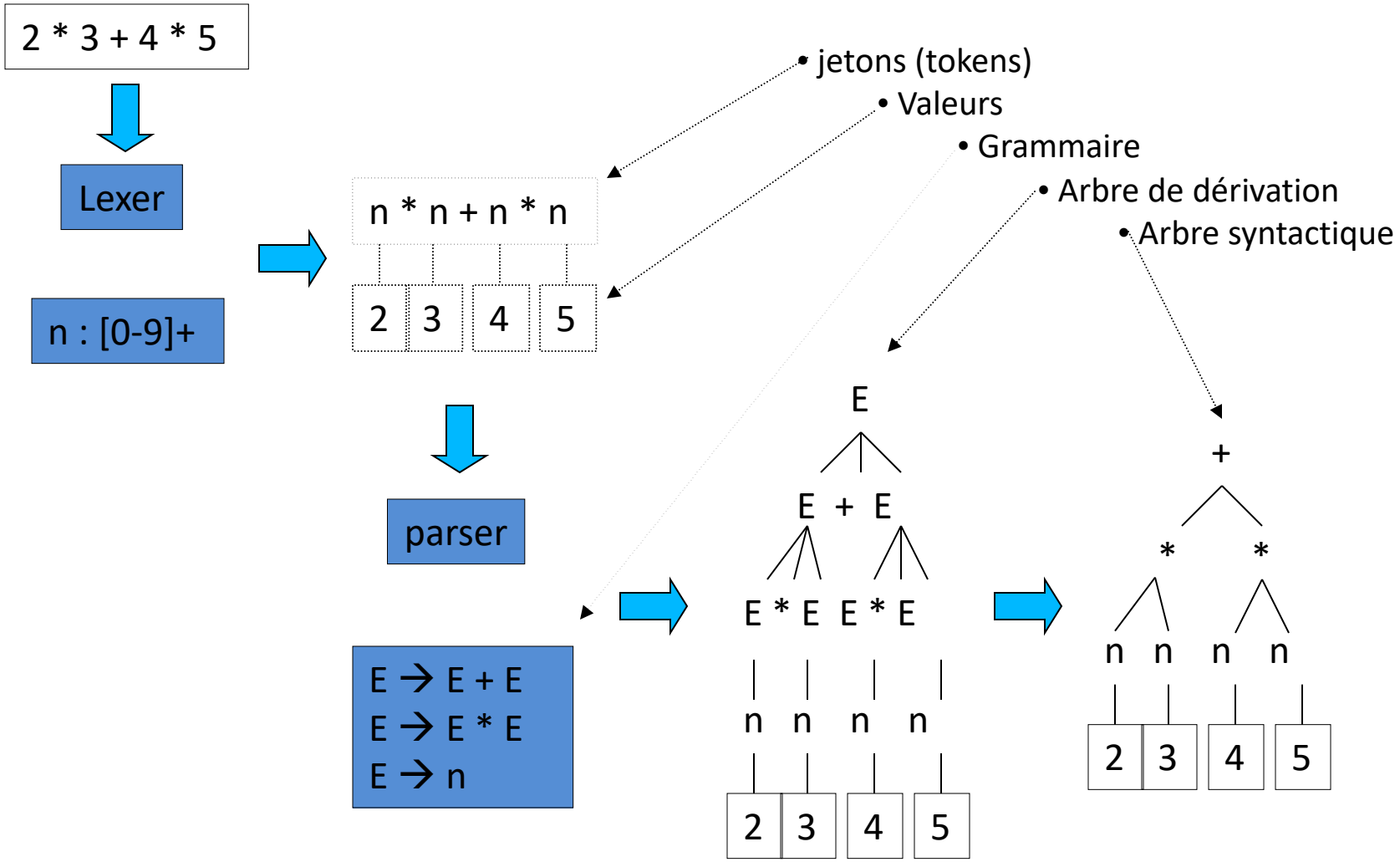
Bogdan Nitulescu

Notation BNF

ALF
language crunch

```
RFC 2616                        HTTP/1.1                        June 1999

       HTTP-date    = rfc1123-date | rfc850-date | asctime-date

       rfc1123-date = wkday "," SP date1 SP time SP "GMT“
       rfc850-date  = weekday "," SP date2 SP time SP "GMT“
       asctime-date = wkday SP date3 SP time SP 4DIGIT

       date1        = 2DIGIT SP month SP 4DIGIT
                      ; day month year (e.g., 02 Jun 1982)
       date2        = 2DIGIT "-" month "-" 2DIGIT
                      ; day-month-year (e.g., 02-Jun-82)
       date3        = month SP ( 2DIGIT | ( SP 1DIGIT ))
                      ; month day (e.g., Jun  2)
       time         = 2DIGIT ":" 2DIGIT ":" 2DIGIT
                      ; 00:00:00 - 23:59:59

       wkday        = "Mon" | "Tue" | "Wed“
                    | "Thu" | "Fri" | "Sat" | "Sun“
       weekday      = "Monday" | "Tuesday" | "Wednesday“
                    | "Thursday" | "Friday" | "Saturday" | "Sunday“
       month        = "Jan" | "Feb" | "Mar" | "Apr“
                    | "May" | "Jun" | "Jul" | "Aug"
                    | "Sep" | "Oct" | "Nov" | "Dec"
```

Arbre de dérivation / syntactique



Types d’analyse syntactique

ALF
language crunch

- Descendent (top-down)
  - Avec backtracking
  - Prédictive
    - Descendent récursive, LL avec un tableau
- Ascendant (bottom-up)
  - Avec backtracking
  - Shift-reduce
    - LR(0),SLR,LALR, LR canonique

Dérivation gauche, top down

ALF language crunch

– **Instr**	id = (id + id) ;
– id = **Expr** ;	id = (id + id) ;
– id = (**Expr**) ;	id = (id + id) ;
– id = (**Expr** + Expr) ;	id = (id + id) ;
– id = (id + **Expr**) ;	id = (id + id) ;
– id = (id + id) ;	id = (id + id) ;

- LL: La chaîne de jetons est itérée à partir du côté gauche (L)
- Le non-terminal le plus à gauche est dérivé (L)

Dérivation gauche, top down

ALF
language crunch

– **Instr**	id = (id) + (id) ;
– id = **Expr** ;	id = (id) + (id) ;
– id = **Expr** + Expr ;	id = (id) + (id) ;
– id = (**Expr**) + Expr ;	id = (id) + (id) ;
– id = (id) + **Expr** ;	id = (id) + (id) ;
– id = (id) + (**Expr**) ;	id = (id) + (id) ;
– id = (id + id) ;	id = (id) + (id) ;

- Comment choisir la production utilisée pour la dérivation?
- Backtracking?

Parser LL, LR

ALF
language crunch

- Nous devrions éviter backtracking
- Une grammaire qui permet le parser déterministe
  - LL(k) lit left-to-right, dérivation left
  - LR(k) lit left-to-right, dérivation right
  - K – lookahead (combien de tokens sont lus)
- LL(k) < LR(k)
- L'algorithme est indépendant du langage, la grammaire dépend du langage

Analyse descendent récursive

ALF
language crunch

- Non-terminal -> fonction
- Si le symbole apparaît dans la partie droite de production -> appel la fonction
- Si le symbole apparaît dans la partie gauche de production – la production est choisi en fonction des jetons (tokens) suivants (lookahead)

Analyse descendent récursive

rfc850-date = weekday "," SP date2 SP time SP "GMT“

Fonction pour parser le non-terminal rfc850-date

```
ParseRFC850Date() {
  ParseWeekDay();
  MatchToken(",");
  MatchToken(SP);
  ParseDate2();
  MatchToken(SP);
  ParseTime();
  MatchToken(SP);
  MatchToken("GMT“);
}
```

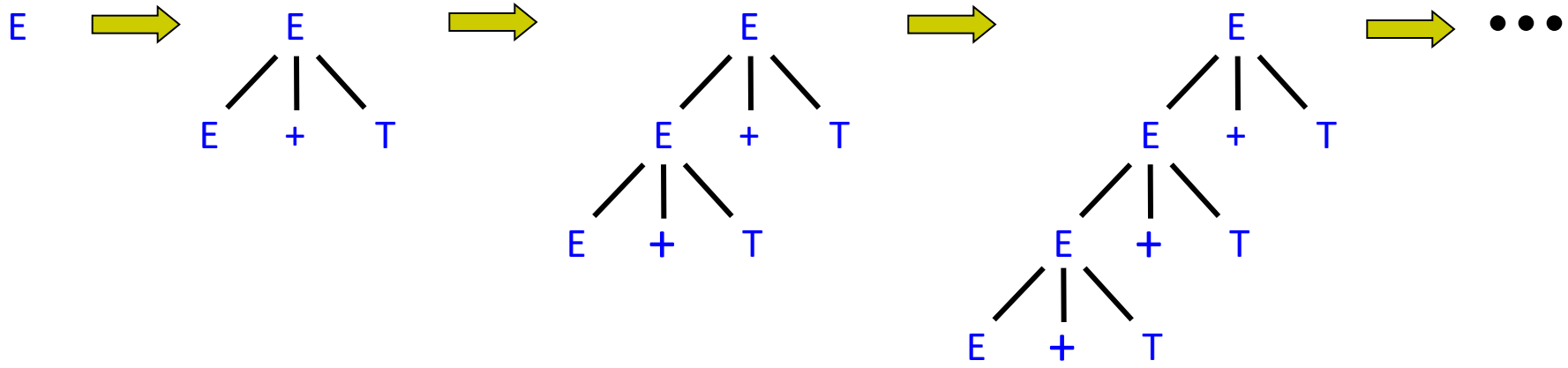
```
MatchToken (token) {
  if (lookahead != token) throw error();
  lookahead = lexer.getNextToken();
}
```

Récursivité gauche

Avec la grammaire

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid \mathbf{id}$$

Un analyseur descendant entre dans une boucle infinie lorsque vous essayez d'analyser cette grammaire



(Aho,Sethi,Ullman, pp. 176)

Récursivité gauche

Grammaire des expression

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid \mathbf{id}$$

Peut être écrive sans la récursivité gauche

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' \mid \varepsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT' \mid \varepsilon$$
$$F \rightarrow (E) \mid \mathbf{id}$$

ε – string vide

(Aho,Sethi,Ullman, pp. 176)

Exemple de analyseur récursive

ALF
language crunch

```
ParseE() {
 ParseT(); ParseE1();
}

ParseE1() {
  if (lookahead==“+”)
  {
    MatchToken(“+”);
    ParseT();
    ParseE1();
  }
}
```

```
ParseT() {
 ParseF(); ParseT1();
}

ParseT1() {
 if (lookahead==“*”)
 {
   MatchToken(“*”);
   ParseF();
   ParseT1();
 }
}
```

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid \mathbf{id}$

```
ParseF() {
 if (lookahead == “(“) {
    MatchToken(“(“); ParseE(); MatchToken(“)”);
  }
else
   MatchToken(T_ID);
}
```

Analyse descendent récursive

Comment choisir entre deux productions?

Comment pouvons-nous savoir quelles conditions de poser a if?

Lorsque nous émettons des erreurs?

```
F → ( E )
F → id
T' → *FT'
T' → ϵ
```

```
ParseT1() {
   if (lookahead==“*”) {
       MatchToken(“*”);
       ParseF();
       ParseT1();
    }
    else if (lookahead == “+”) { }
    else if (lookahead == “)”) { }
    else if (lookahead == T_EOF) { }
    else throw error();
}
```

```
ParseF() {
   if (lookahead == “(“) {
       MatchToken(“(“);
       ParseE();
       MatchToken(“)”);
    }
   else if (lookahead == T_ID)
   {
     MatchToken(T_ID);
   }
   else throw error();
}
```

Les conditions pour if

ALF
language crunch

- FIRST
  - Ensemble de terminaux-préfixées pour le non-terminal
- FOLLOW
  - Ensemble de terminaux suivantes pour le non-terminal
- NULLABLE
  - Ensemble de non-terminaux qui peut etre derive en ε

FIRST

GRAMMAIRE:

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid \mathbf{id}$

ENSEBLES:

FIRST(**id**) = {**id**}
FIRST(*) = {*}
FIRST(**+**) = {**+**}
FIRST(**(**) = {**(**}
FIRST(**)**) = {**)**}
FIRST(E') = ~~{ε}~~ {**+**, ε}
FIRST(T') = ~~{ε}~~ {*, ε}
FIRST(F) = {(, id}
FIRST(T) = FIRST(F) = {(, id}
FIRST(E) = FIRST(T) = {(, id}

FIRST (pseudocode):

1. If X is a terminal, FIRST(X) = {X}
2. If $X \rightarrow \varepsilon$, then $\varepsilon \in FIRST(X)$
3. If $X \rightarrow Y_1 Y_2 \bullet\bullet\bullet Y_k$
   and $Y_1 \bullet\bullet\bullet Y_{i-1} \stackrel{*}{\Rightarrow} \varepsilon$
   and $a \in FIRST(Y_i)$
   then $a \in FIRST(X)$
4. If $X \rightarrow Y_1 Y_2 \bullet\bullet\bullet Y_k$
   and $a \in FIRST(Y_1)$
   then $a \in FIRST(X)$

(Aho,Sethi,Ullman, pp. 189)

FIRST(E') = {+, ε}
FIRST(T') = {* , ε}
FIRST(F) = {(, id}
FIRST(T) = {(, id}
FIRST(E) = {(, id}

FOLLOW

GRAMMAIRE:

E → TE'
E' → +TE' | ε
T → FT'
T' → *FT' | ε
F → (E) | **id**

ENSEBLES:

FOLLOW(E) = ~~{$}~~ { **)**, **$**}
FOLLOW(E') = { **)**, **$**}
FOLLOW(T) = { **)**, **$**}

FOLLOW – pseudocode:

1. If S is the start symbol, then $ ∈ FOLLOW(S)
2. If $A \to \alpha B \beta$,
   and a ∈ FIRST(β)
   and a ≠ ε
   then a ∈ FOLLOW(B)
3. If $A \to \alpha B$
   and a ∈ FOLLOW(A)
   then a ∈ FOLLOW(B)

3a. If $A \to \alpha B \beta$
   and $\beta \overset{*}{\Rightarrow} \varepsilon$
   and a ∈ FOLLOW(A)
   then a ∈ FOLLOW(B)

α et β - string de terminaux et non-terminaux
A et B – non-terminaux,
$ - fin du text

(Aho,Sethi,Ullman, pp. 189)

FOLLOW

FIRST(E') = {+, ε}
FIRST(T') = {* , ε}
FIRST(F) = {(, id}
FIRST(T) = {(, id}
FIRST(E) = {(, id}

GRAMMAIRE:

E → TE'
E' → +TE' | ε
T → FT'
T' → *FT' | ε
F → (E) | **id**

ENSEBLES:

FOLLOW(E) = {**)**, **$**}
FOLLOW(E') = { **)**, **$**}
FOLLOW(T) = { **)**, **$**} {+, **)**, **$**}

FOLLOW – règles:

1. If S is the start symbol, then $ ∈ FOLLOW(S)
2. If $A \to \alpha B \beta$,
   and a ∈ FIRST(β)
   and a ≠ ε
   then a ∈ FOLLOW(B)
3. If $A \to \alpha B$
   and a ∈ FOLLOW(A)
   then a ∈ FOLLOW(B)

3a. If $A \to \alpha B \beta$
   and $\beta \overset{*}{\Rightarrow} \varepsilon$
   and a ∈ FOLLOW(A)
   then a ∈ FOLLOW(B)

(Aho,Sethi,Ullman, pp. 189)

FIRST(E') = {+, ε}
FIRST(T') = {* , ε}
FIRST(F) = {(, id}
FIRST(T) = {(, id}
FIRST(E) = {(, id}

FOLLOW

GRAMMAIRE:

E → TE'
E' → +TE' | ε
T → FT'
T' → *FT' | ε
F → (E) | **id**

ENSEBLES:

FOLLOW(E) = {**)**, **$**}
FOLLOW(E') = { **)**, **$**}
FOLLOW(T) = {**+**, **)**, **$**}
FOLLOW(T') = {**+**, **)**, **$**}

FOLLOW – règles:

1. If S is the start symbol, then $ ∈ FOLLOW(S)
2. If $A \to \alpha B \beta$,
   and a ∈ FIRST(β)
   and a ≠ ε
   then a ∈ FOLLOW(B)
3. If $A \to \alpha B$
   and a ∈ FOLLOW(A)
   then a ∈ FOLLOW(B)

3a. If $A \to \alpha B \beta$
   and $\beta \overset{*}{\Rightarrow} \varepsilon$
   and a ∈ FOLLOW(A)
   then a ∈ FOLLOW(B)

(Aho,Sethi,Ullman, pp. 189)

FIRST(E') = {+, ε}
FIRST(T') = {* , ε}
FIRST(F) = {(, id}
FIRST(T) = {(, id}
FIRST(E) = {(, id}

FOLLOW

GRAMMAIRE:

E → TE'
E' → +TE' | ε
T → FT'
T' → *FT' | ε
F → (E) | **id**

ENSEBLES:

FOLLOW(E) = {**)**, **$**}
FOLLOW(E') = { **)**, **$**}
FOLLOW(T) = {**+**, **)**, **$**}
FOLLOW(T') = {**+**, **)**, **$**}
FOLLOW(F) = {**+**, **)**, **$**}

FOLLOW – règles:

1. If S is the start symbol, then $ ∈ FOLLOW(S)
2. If $A \to \alpha B \beta$,
   and $a \in FIRST(\beta)$
   and $a \neq \varepsilon$
   then $a \in FOLLOW(B)$
3. If $A \to \alpha B$
   and $a \in FOLLOW(A)$
   then $a \in FOLLOW(B)$

3a. If $A \to \alpha B \beta$
   and $\beta \overset{*}{\Rightarrow} \varepsilon$
   and $a \in FOLLOW(A)$
   then $a \in FOLLOW(B)$

(Aho,Sethi,Ullman, pp. 189)

FOLLOW

FIRST(E') = {+, ε}
FIRST(T') = {* , ε}
FIRST(F) = {(, id}
FIRST(T) = {(, id}
FIRST(E) = {(, id}

GRAMMAIRE:

E → TE'
E' → +TE' | ε
T → FT'
T' → *FT' | ε
F → (E) | **id**

ENSEBLES:

FOLLOW(E) = {**)**, **$**}
FOLLOW(E') = { **)**, **$**}
FOLLOW(T) = {**+**, **)**, **$**}
FOLLOW(T') = {**+**, **)**, **$**}
FOLLOW(F) = ~~{**+**, **)**, **$**}~~ {**+**, ***, **)**, **$**}

FOLLOW – règles:

1. If S is the start symbol, then $ ∈ FOLLOW(S)
2. If $A \to \alpha B \beta$,
   and $a \in FIRST(\beta)$
   and $a \neq \varepsilon$
   then $a \in FOLLOW(B)$
3. If $A \to \alpha B$
   and $a \in FOLLOW(A)$
   then $a \in FOLLOW(B)$

3a. If $A \to \alpha B \beta$
   and $\beta \overset{*}{\Rightarrow} \varepsilon$
   and $a \in FOLLOW(A)$
   then $a \in FOLLOW(B)$

(Aho,Sethi,Ullman, pp. 189)

L'algo générique récursive LL(1)

- Pour chaque non-terminal crée une fonction d'analyseur.
- Pour chaque règle A→α ajouter un test
  if (lookahead in FIRST(αFOLLOW(A)))
- Pour chaque non-terminal dans a appeler la fonction de parser.
- Pour chaque terminal dans a, vérifier le lookahead(match)

```
ParseA() {
  if (lookahead in FIRST(a B … x FOLLOW(A)) {
    MatchToken(a); ParseB(); … MatchToken(x);
  }
  else if (lookahead in FIRST(C D … y FOLLOW(A))
  {
     ParseC(); ParseD(); … MatchToken(y);
  }
  …
  else throw error();
}
```

```
A → a B … x
A → C D … y
…
```

Récursivité gauche

Quand une grammaire a au moins une forme de production
$A \rightarrow A\alpha$
nous disons qu'il est une grammaire **récursive gauche**.

Les analyseurs descendent ne fonctionnent pas (sans backtracking) sur les grammaires récursives gauche.

Récursivité peut ne pas être immédiat
$A \rightarrow B\alpha$
$B \rightarrow A\ \beta$

(Aho,Sethi,Ullman, pp. 176)

Elimination récursivité gauche

- Cela se fait par la réécriture de la grammaire

$List \rightarrow List\ Item \mid Item$

$List \rightarrow Item\ List'$
$List' \rightarrow Item\ List' \mid \varepsilon$

$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid \mathbf{id}$

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid \mathbf{id}$

(Aho,Sethi,Ullman, pp. 176)

Elimination récursivité gauche

$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid \mathbf{id}$

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid \mathbf{id}$

Cas général (récursivité immédiat):

$A \rightarrow A\beta_1 \mid A\beta_2 \mid \ldots \mid A\beta_m \mid \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$

$A \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_n A'$
$A' \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_m A' \mid \varepsilon$

Factorisation gauche

- Pour une instruction if:

```
if_statement ->  IF expression THEN statement ENDIF |
                 IF expression THEN statement ELSE statement ENDIF
```

- Pour parser avec LL, elle doit être factorise:

```
if_statement -> IF expression THEN statement close_if
    close_if  -> ENDIF | ELSE statement ENDIF
```

```
void ParseIfStatement()
{
   MatchToken(T_IF);
   ParseExpression();
   MatchToken(T_THEN);
   ParseStatement();
   ParseCloseIf();
}
```

```
void ParseCloseIf()
{
   if (lookahead == T_ENDIF)
      lookahead = yylex();
   else {
      MatchToken(T_ELSE);
      ParseStatement();
      MatchToken(T_ENDIF);
   }
}
```

Factorisation gauche

- Cas général:

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \delta$$

- Factorise:

$$A \rightarrow \alpha A' \mid \delta$$
$$A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

Elimination des ambiguïtés

ALF
language crunch

- Ambigu: $E \rightarrow E + E \mid E * E \mid a \mid (E)$

1. $E \rightarrow E + T \mid T$
   $T \rightarrow T * F \mid F$
   $F \rightarrow a \mid (E)$

2. $E \rightarrow T + E \mid T$
   $T \rightarrow F * T \mid F$
   $F \rightarrow a \mid (E)$

- La précédence des operateurs
- La associativité gauche ou droite

Elimination des ambiguïtés

ALF
language crunch

- Productions qui peuvent produire l'ambiguïté:
  $X \rightarrow aAbAc$

- Cas général:
  $A \rightarrow A\ B\ A \mid \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$

- Désambiguïsât:
  $A \rightarrow A'\ B\ A \mid A`$
  $A' \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$

Analyseur automatique

ALF
language crunch

- Automate push-down
- Le analyseur est fait avec un automate est un tableau
- Langage LL(1) si il n'a pas de conflits dans le tableau

Exemple d'analyseur LL

Grammaire:

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' \mid \varepsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT' \mid \varepsilon$$
$$F \rightarrow (E) \mid \mathbf{id}$$

Tableau de Parsing:

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	E→TE′			E→TE′		
E′		E′→+TE′			E′→ε	E′→ε
T	T→FT′			T→FT′		
T′		T′→ε	T′→*FT′		T′→ε	T′→ε
F	F→**id**			F→(E)		

(Aho,Sethi,Ullman, pp. 188)

Exemple d'analyseur LL

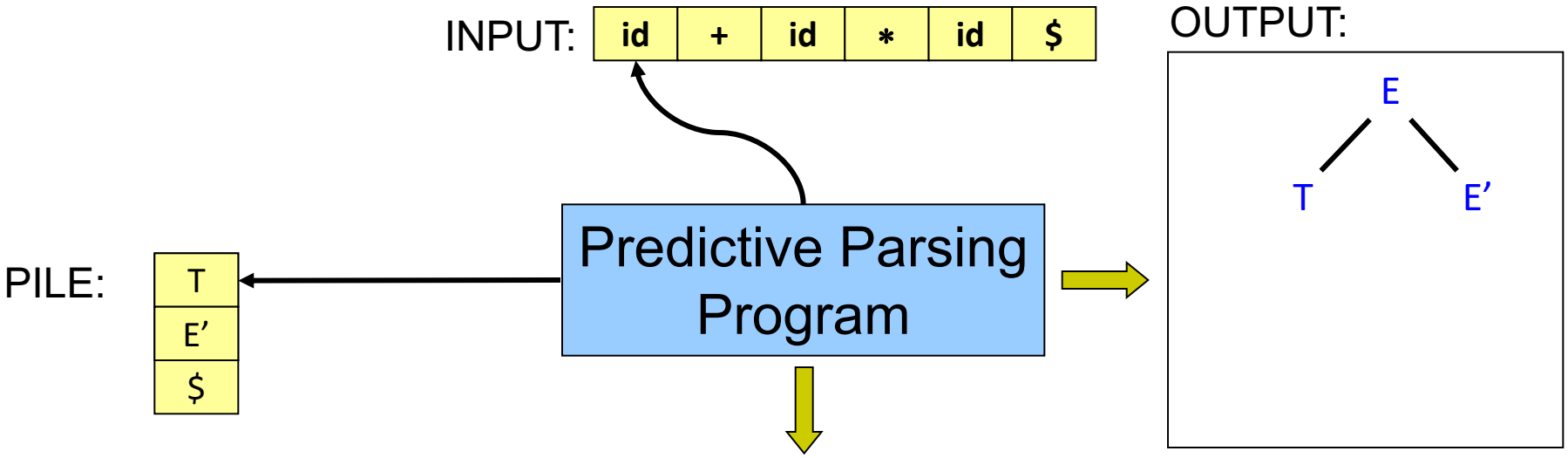
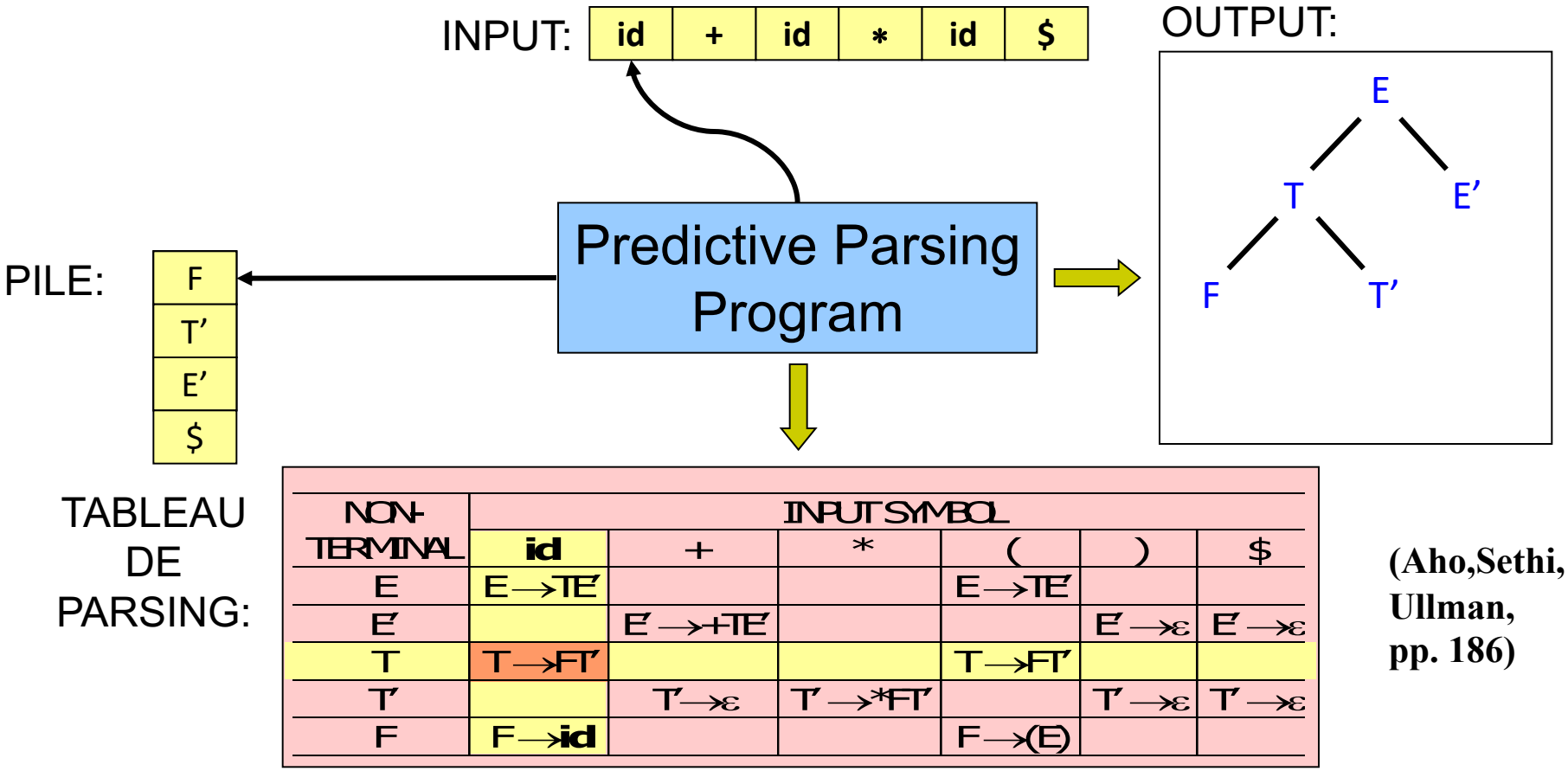


TABLEAU DE PARSING:

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	E → TE'			E → TE'		
E'		E' → +TE'			E' → ε	E' → ε
T	T → FT'			T → FT'		
T'		T' → ε	T' → *FT'		T' → ε	T' → ε
F	F → **id**			F → (E)		

Exemple d'analyseur LL



(Aho,Sethi, Ullman, pp. 186)

Exemple d'analyseur LL

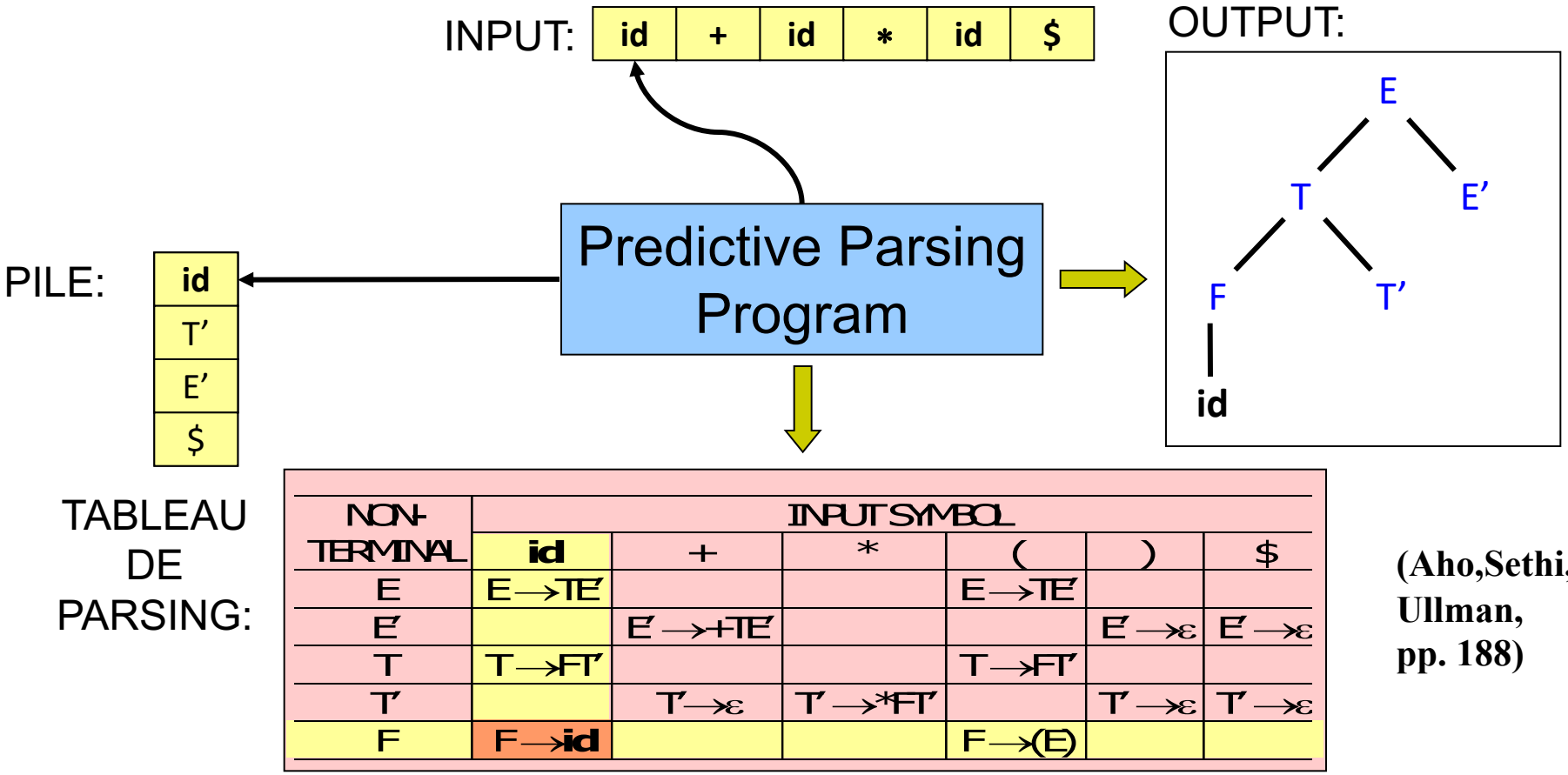


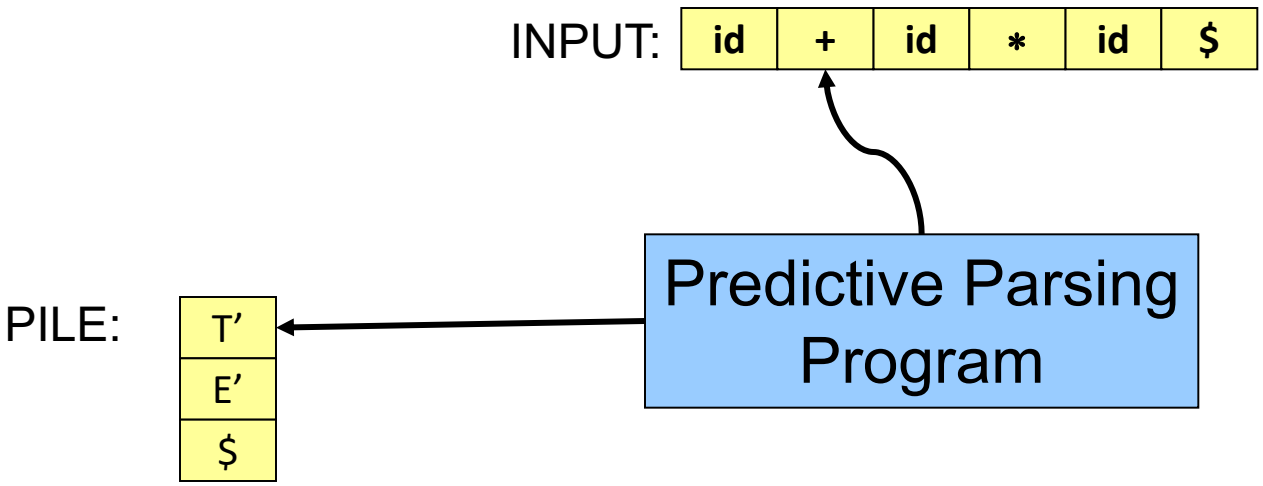
TABLEAU DE PARSING:

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	E → TE'			E → TE'		
E'		E' → +TE'			E' → ε	E' → ε
T	T → FT'			T → FT'		
T'		T' → ε	T' → *FT'		T' → ε	T' → ε
F	F → id			F → (E)		

(Aho,Sethi, Ullman, pp. 188)

Exemple d'analyseur LL

Quand l'action c'est Top(Pile) = input ≠ $: 'Pop' de la pile, avance la bande de input.



OUTPUT:

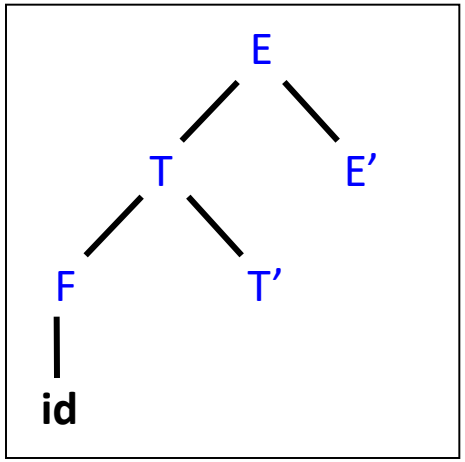


TABLEAU DE PARSING:

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→**id**			F→(E)		

(Aho,Sethi, Ullman, pp. 188)

Exemple d'analyseur LL

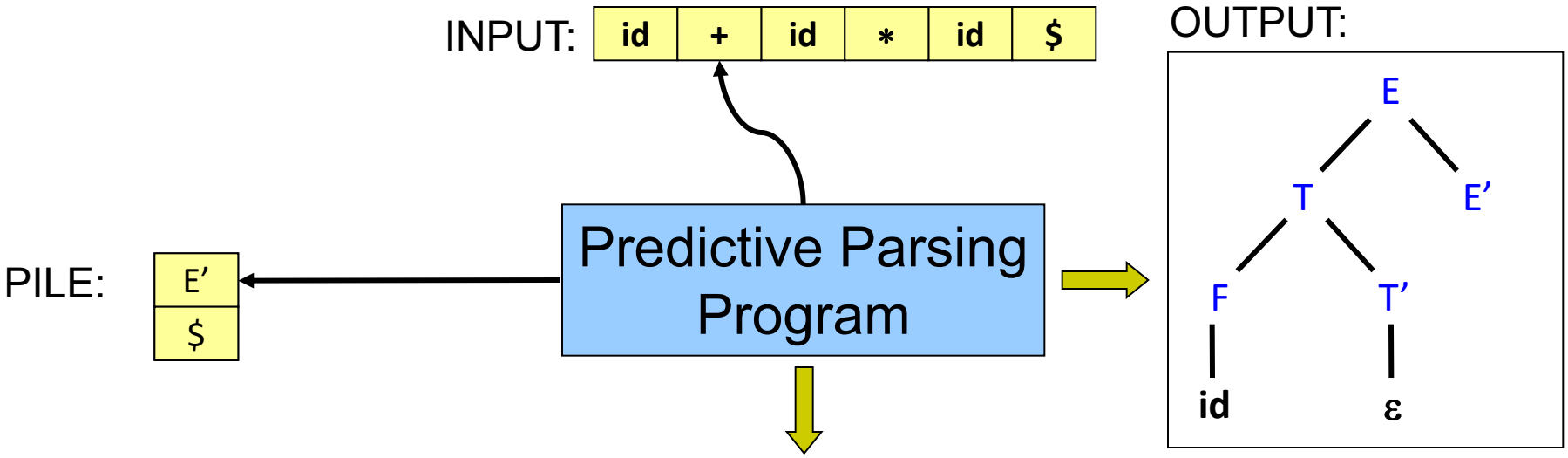


TABLEAU DE PARSING:

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→**id**			F→(E)		

(Aho,Sethi, Ullman, pp. 188)

Exemple d'analyseur LL

Et ainsi, il construit l'arbre de dérivation:

$E' \rightarrow +TE'$

$T \rightarrow FT'$

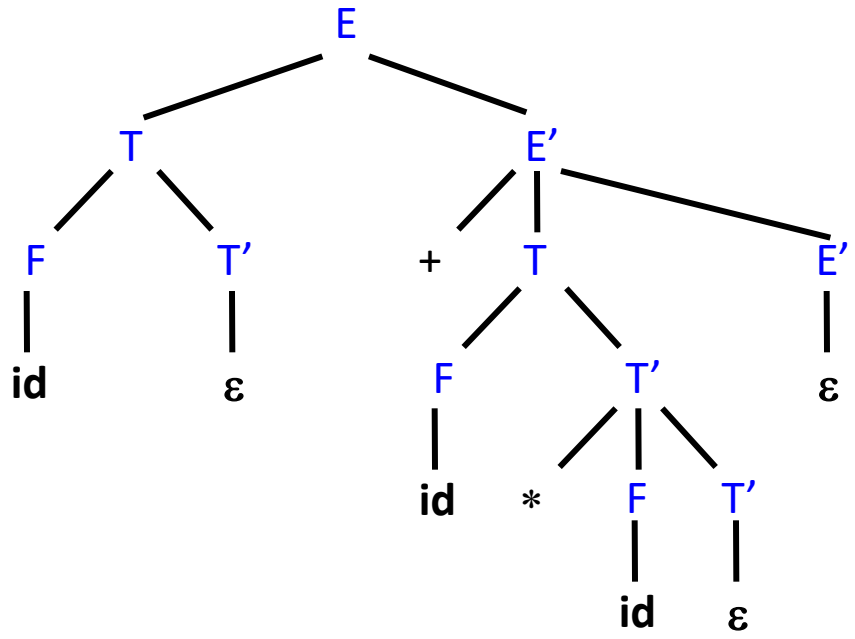
$F \rightarrow \mathbf{id}$

$T' \rightarrow * FT'$

$F \rightarrow \mathbf{id}$

$T' \rightarrow \varepsilon$

$E' \rightarrow \varepsilon$



Quand Top(Pile) = input = $

Le parser arrêt et accepte l'input

(Aho,Sethi, Ullman, pp. 188)

Remplir de tableau

ALF
language crunch

- FIRST
  - Ensemble de terminaux-préfixées pour le non-terminal
- FOLLOW
  - Ensemble de terminaux suivantes pour le non-terminal
- NULLABLE
  - Ensemble de non-terminaux qui peut etre derive en ε

GRAMMAIRE:

E → TE'
E' → +TE' | ε
T → FT'
T' → *FT' | ε
F → (E) | **id**

FIRST SETS:

FIRST(E') = {+, ε}
FIRST(T') = {* , ε}
FIRST(F) = {(, id}
FIRST(T) = {(, id}
FIRST(E) = {(, id}

FOLLOW SETS:

FOLLOW(E) = {), $}
FOLLOW(E') = {), $}
FOLLOW(T) = {+,), $}
FOLLOW(T') = {+,), $}
FOLLOW(F) = {+, *,), $}

1. If $A \rightarrow \alpha$:
   if $a \in FIRST(\alpha)$, add $A \rightarrow \alpha$ to M[A, a]

TABLEAU DE PARSING:

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→**id**			F→(E)		

Règles de construction de la table d'analyse

(Aho,Sethi,Ullman, pp. 190)

GRAMMAIRE:

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid \mathbf{id}$

FIRST SETS:

FIRST(E') = {+, ε}
FIRST(T') = {* , ε}
FIRST(F) = {(, id}
FIRST(T) = {(, id}
FIRST(E) = {(, id}

FOLLOW SETS:

FOLLOW(E) = {), $}
FOLLOW(E') = {), $}
FOLLOW(T) = {+,), $}
FOLLOW(T') = {+,), $}
FOLLOW(F) = {+, *,), $}

1. If $A \rightarrow \alpha$:
   if $a \in FIRST(\alpha)$, add $A \rightarrow \alpha$ to M[A, a]

TABLEAU DE PARSING:

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \mathbf{id}$			$F \rightarrow (E)$		

Règles de construction de la table d'analyse

(Aho,Sethi,Ullman, pp. 190)

GRAMMAIRE:

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid \mathbf{id}$

FIRST SETS:

FIRST(E') = {+, ε}
FIRST(T') = {* , ε}
FIRST(F) = {(, id}
FIRST(T) = {(, id}
FIRST(E) = {(, id}

FOLLOW SETS:

FOLLOW(E) = {), $}
FOLLOW(E') = {), $}
FOLLOW(T) = {+,), $}
FOLLOW(T') = {+,), $}
FOLLOW(F) = {+, *,), $}

1. If $A \rightarrow \alpha$:
   if $a \in FIRST(\alpha)$, add $A \rightarrow \alpha$ to M[A, a]

TABLEAU DE PARSING:

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \mathbf{id}$			$F \rightarrow (E)$		

Règles de construction de la table d'analyse

(Aho,Sethi,Ullman, pp. 190)

GRAMMAIRE:

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid \mathbf{id}$

FIRST SETS:

FIRST(E') = {+, ε}
FIRST(T') = {* , ε}
FIRST(F) = {(, id}
FIRST(T) = {(, id}
FIRST(E) = {(, id}

FOLLOW SETS:

FOLLOW(E) = {), $}
FOLLOW(E') = {), $}
FOLLOW(T) = {+,), $}
FOLLOW(T') = {+,), $}
FOLLOW(F) = {+, *,), $}

1. If $A \rightarrow \alpha$:
   if $a \in FIRST(\alpha)$, add $A \rightarrow \alpha$ to M[A, a]

TABLEAU DE PARSING:

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \mathbf{id}$			$F \rightarrow (E)$		

Règles de construction de la table d'analyse

(Aho,Sethi,Ullman, pp. 190)

GRAMMAIRE:

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid \mathbf{id}$

FIRST SETS:

FIRST(E') = {+, ε}
FIRST(T') = {* , ε}
FIRST(F) = {(, id}
FIRST(T) = {(, id}
FIRST(E) = {(, id}

FOLLOW SETS:

FOLLOW(E) = {), $}
FOLLOW(E') = {), $}
FOLLOW(T) = {+,), $}
FOLLOW(T') = {+,), $}
FOLLOW(F) = {+, *,), $}

1. If $A \rightarrow \alpha$:
   if $a \in FIRST(\alpha)$, add $A \rightarrow \alpha$ to M[A, a]

TABLEAU DE PARSING:

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \mathbf{id}$			$F \rightarrow (E)$		

Règles de construction de la table d'analyse

(Aho,Sethi,Ullman, pp. 190)

GRAMMAIRE:

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid \mathbf{id}$

FIRST SETS:

FIRST(E') = {+, ε}
FIRST(T') = {* , ε}
FIRST(F) = {(, id}
FIRST(T) = {(, id}
FIRST(E) = {(, id}

FOLLOW SETS:

FOLLOW(E) = {), $}
FOLLOW(E') = {), $}
FOLLOW(T) = {+,), $}
FOLLOW(T') = {+,), $}
FOLLOW(F) = {+, *,), $}

1. If $A \rightarrow \alpha$:
   if $a \in FIRST(\alpha)$, add $A \rightarrow \alpha$ to M[A, a]
2. If $A \rightarrow \alpha$:
   if $\varepsilon \in FIRST(\alpha)$, add $A \rightarrow \alpha$ to M[A, b]
   for each terminal $b \in FOLLOW(A)$,

TABLEAU DE PARSING:

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \mathbf{id}$			$F \rightarrow (E)$		

Règles de construction de la table d'analyse

(Aho,Sethi,Ullman, pp. 190)

GRAMMAIRE:

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid \mathbf{id}$

FIRST SETS:

FIRST(E') = {+, ε}
FIRST(T') = {* , ε}
FIRST(F) = {(, id}
FIRST(T) = {(, id}
FIRST(E) = {(, id}

FOLLOW SETS:

FOLLOW(E) = {), $}
FOLLOW(E') = {), $}
FOLLOW(T) = {+,), $}
FOLLOW(T') = {+,), $}
FOLLOW(F) = {+, *,), $}

1. If $A \rightarrow \alpha$:
   if $a \in FIRST(\alpha)$, add $A \rightarrow \alpha$ to M[A, a]
2. If $A \rightarrow \alpha$:
   if $\varepsilon \in FIRST(\alpha)$, add $A \rightarrow \alpha$ to M[A, b]
   for each terminal $b \in FOLLOW(A)$,

TABLEAU DE PARSING:

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \mathbf{id}$			$F \rightarrow (E)$		

Règles de construction de la table d'analyse

(Aho,Sethi,Ullman, pp. 190)

GRAMMAIRE:

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid \mathbf{id}$

FIRST SETS:

FIRST(E') = {+, ε}
FIRST(T') = {* , ε}
FIRST(F) = {(, id}
FIRST(T) = {(, id}
FIRST(E) = {(, id}

FOLLOW SETS:

FOLLOW(E) = {), $}
FOLLOW(E') = {), $}
FOLLOW(T) = {+,), $}
FOLLOW(T') = {+,), $}
FOLLOW(F) = {+, *,), $}

1. If $A \rightarrow \alpha$:
   if $a \in FIRST(\alpha)$, add $A \rightarrow \alpha$ to M[A, a]
2. If $A \rightarrow \alpha$:
   if $\varepsilon \in FIRST(\alpha)$, add $A \rightarrow \alpha$ to M[A, b]
   for each terminal $b \in FOLLOW(A)$,
3. If $A \rightarrow \alpha$:
   if $\varepsilon \in FIRST(\alpha)$, and $\$ \in FOLLOW(A)$,
   add $A \rightarrow \alpha$ to M[A, $]

TABLEAU DE PARSING:

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \mathbf{id}$			$F \rightarrow (E)$		

Règles de construction de la table d'analyse

(Aho,Sethi,Ullman, pp. 190)

Utilisation d'analyseur LL(1)

ALF
language crunch

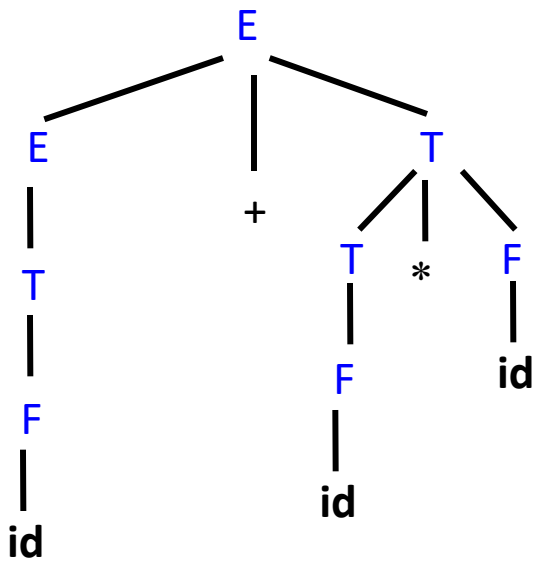
- Grammaires
  - Non ambigu
  - Factorise
  - Non récursive a gauche
- On peut montrer que la grammaire G est LL (1) si et seulement si pour deux productions de la forme

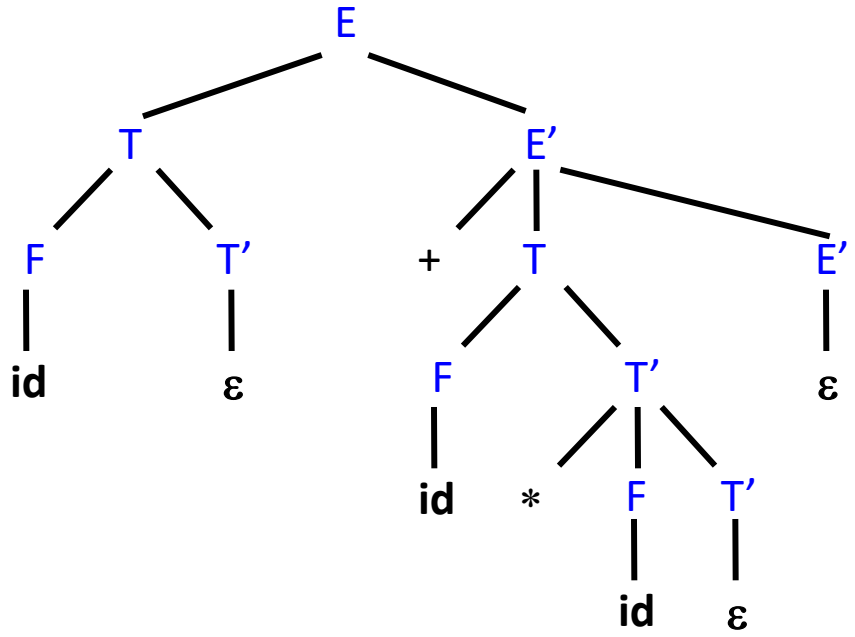
  $A \rightarrow \alpha$, $A \rightarrow \beta$, avec $\alpha \neq \beta$ les conditions suivantes sont satisfaites:
  - $FIRST(\alpha) \cap FIRST(\beta) = \varnothing$
  - Si $\beta \Rightarrow^* \varepsilon$ alors $FIRST(\alpha) \cap FOLLOW(A) = \varnothing$ et si $\alpha \Rightarrow^* \varepsilon$ alors $FIRST(\beta) \cap FOLLOW(A) = \varnothing$.

Avantage/désavantage LL(1)

- Facile de écrive ‘aux main’
- Vite, facile de comprendre

- La grammaire doit être transforme
- L’arbre de dérivation et diffèrent de l’arbre sémantique





Parser LL

ALF
language crunch

- ANTLR
  - Java
  - LL (*)
  - Factorisation

Règles EBNF

ALF
language crunch

Something?	SomethingQ -> ε | Something
Something*	SomethingStar -> ε | Something SomethingStar
Something+	SomethingPlus -> Something SomethingStar

Sujets

ALF
language crunch

- Les analyseurs (parses)
- LL
  - Eviter l'ambiguïté
  - Factorisation
  - Eviter la récursivité gauche
- Algorithme général récursive LL

Questions

ALF
language crunch