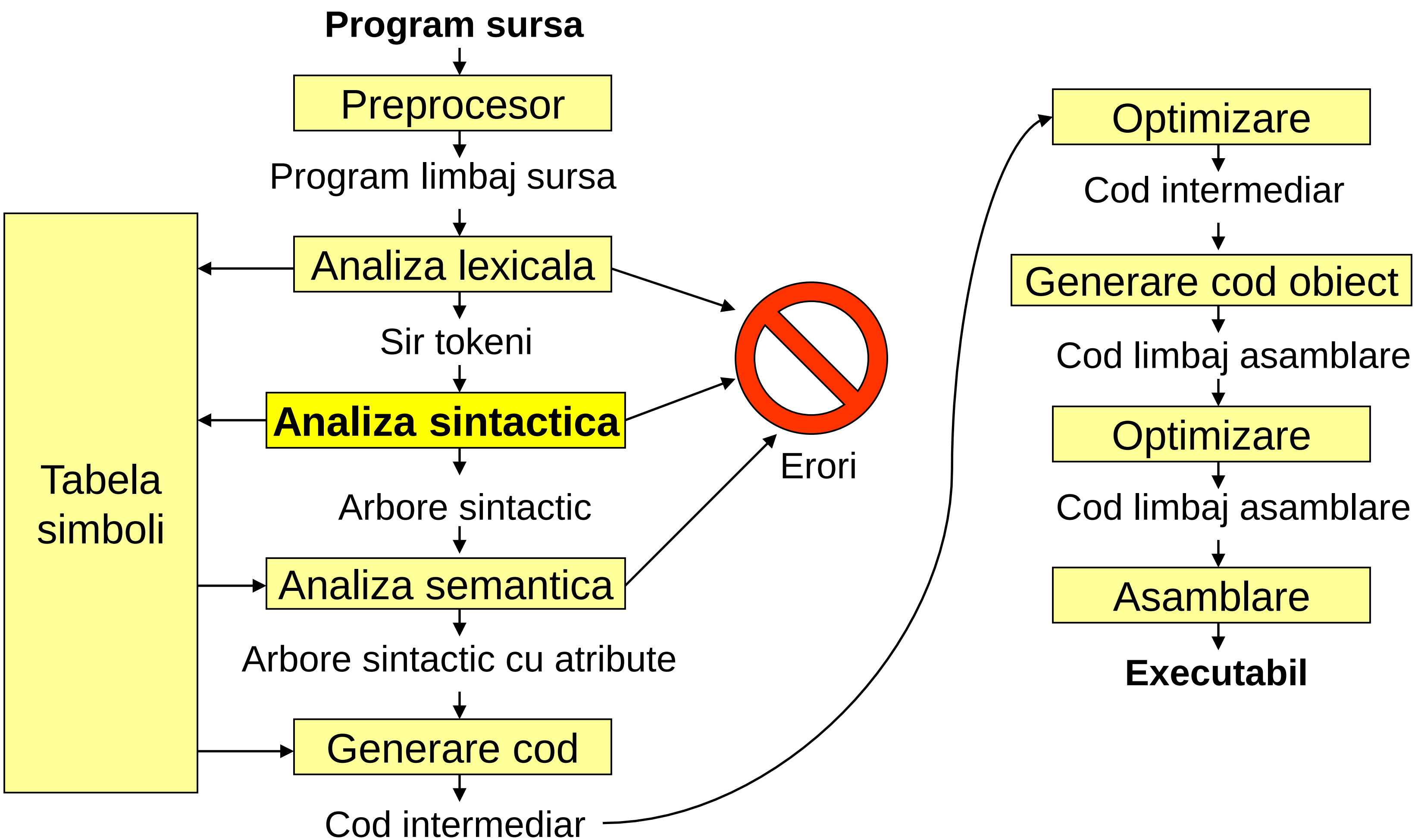
Compilatoare

Analiza Sintactică
Parsere LL

CPL
IR crunch

Structura detaliata



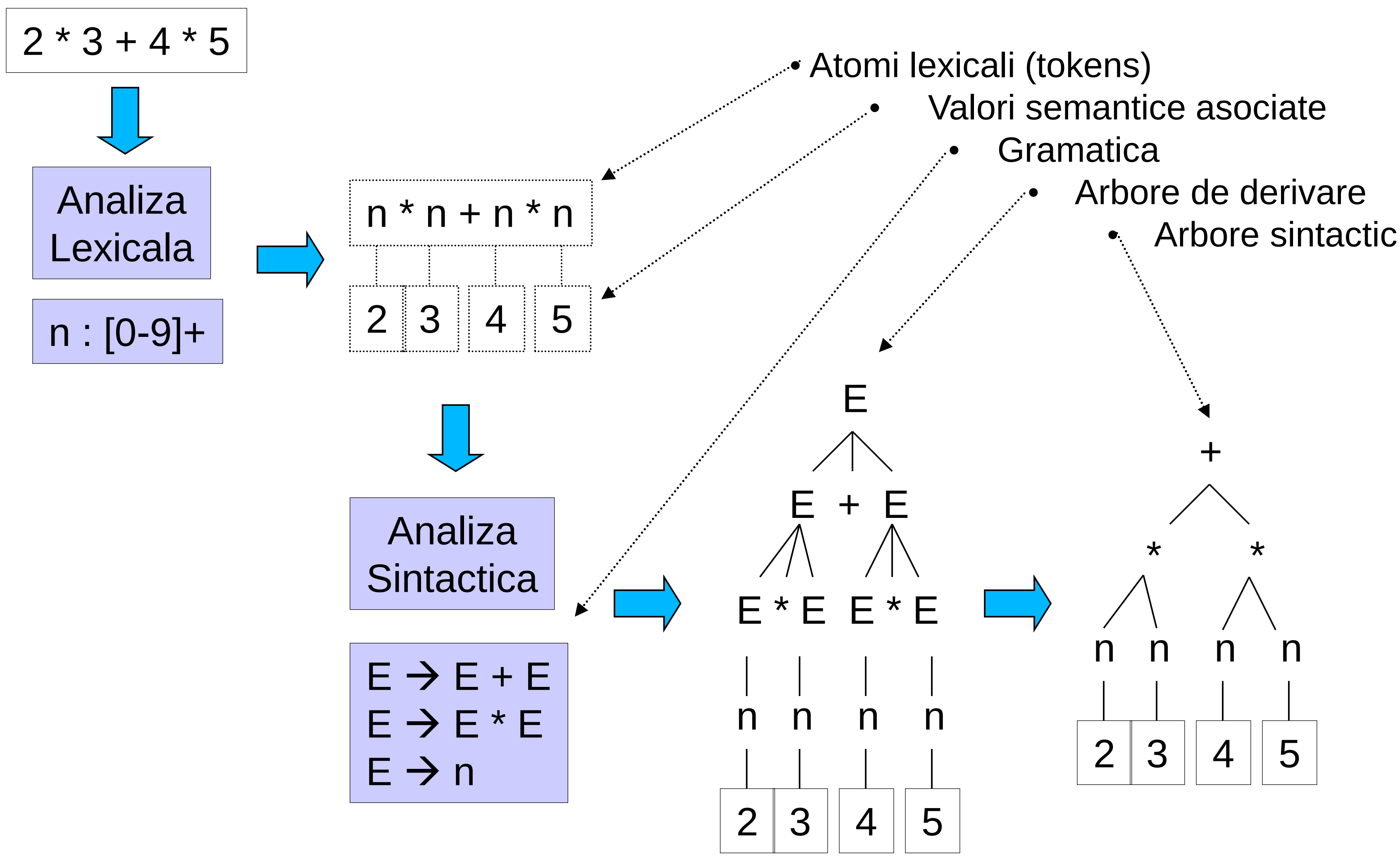
Analiza sintactica

- Verifica formarea corecta (cf. gramaticii) a constructiilor din limbaj
  - Analiza lexicala – “cuvinte”
  - Analiza sintactica – “propozitii”
- Primeste un sir de atomi lexicali, construieste un arbore de derivare
  - Structura utila in final este un arbore sintactic
- Folosita in front-end-ul unui interpretor / compilator
  - Dar si de catre IDE: syntax highlight, navigare prin cod, refactoring

Exemplu de specificatie
Notatia BNF

```
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"
```

Arbore de derivare / sintactic



Tipuri de analiza sintactica

- Descendenta (top-down)
  - Cu backtracking
  - Predictiva
    - Descendent recursiva, LL pe baza de tabel
- Ascendenta (bottom-up)
  - Cu backtracking
  - Shift-reduce
    - LR(0),SLR,LALR, LR canonica

Analiza LL, LR

- Vrem sa evitam backtrackingul
- O clasă de gramatici independente de context care permit o analiza deterministă.
  - Alg. LL(k) analizeaza left-to-right, derivare stanga
  - Alg. LR(k) analizeaza left-to-right, derivare dreapta
  - K – lookahead (cati tokeni sunt cititi)
- LL(k) <LR(k)
- Algoritmul folosit nu depinde de limbaj, gramatica da.

Analiza descendent recursiva

- Fiecare neterminal are o functie care il parseaza
- Daca simbolul apare in partea dreapta a productiei -> functia se va apela recursiv
- Daca un neterminal apare in partea stanga a mai multor productii – se alege una din ele in functie de urmatorii atomi lexicali (lookahead)

Analiza descendent recursiva

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

Functia de parsat nonterminalul rfc850-date

```
ParseRFC850Date() {
  ParseWeekDay();
  MatchToken(T_COMMA);
  MatchToken(T_SPACE);
  ParseDate2();
  MatchToken(T_SPACE);
  ParseTime();
  MatchToken(T_SPACE);
  MatchToken(T_GMT);
  }
```

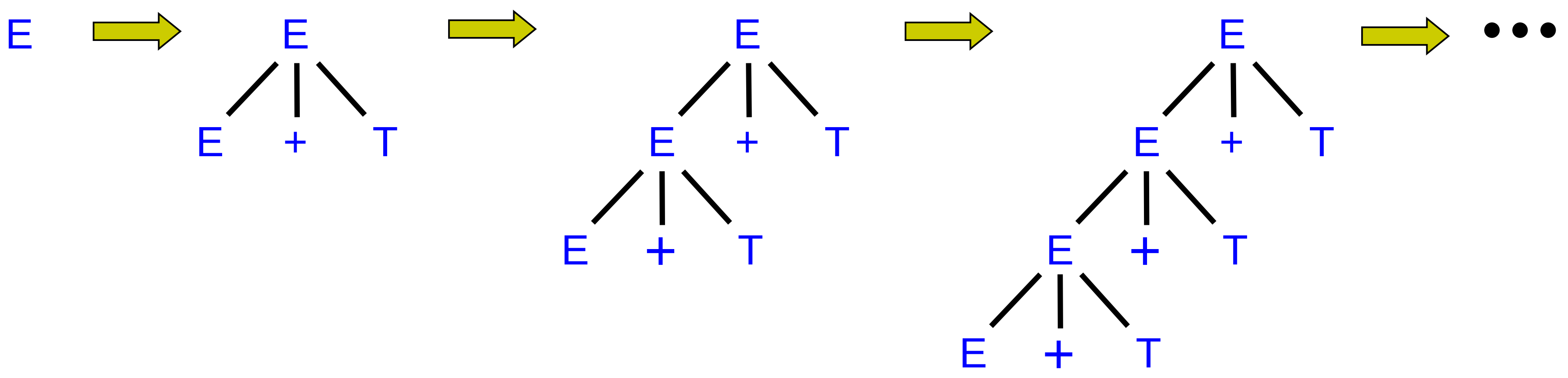
```
MatchToken (token) {
if (lookahead != token) throw error();
lookahead = lexer.getNextToken();
}
```

Recursivitatea stanga

Sa luam gramatica:

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

Un parser descendent intra in bucla infinita cand incearca sa parseze aceasta gramatica



(Aho,Sethi,Ullman, pp. 176)

Recursivitatea stanga

Gramatica expresiilor aritmetice:

$$
\begin{aligned}
E &\rightarrow E + T \mid T \\
T &\rightarrow T * F \mid F \\
F &\rightarrow (E) \mid \textbf{id}
\end{aligned}
$$

Poate fi rescrisa cu eliminarea recursivitatii stanga:

$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow +TE' \mid \varepsilon \\
T &\rightarrow FT' \\
T' &\rightarrow *FT' \mid \varepsilon \\
F &\rightarrow (E) \mid \textbf{id}
\end{aligned}
$$

(Aho,Sethi,Ullman, pp. 176)

Exemplu de parser recursiv

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

ParseE1() {
if (lookahead==T_PLUS)
{
     MatchToken(T_PLUS);
    ParseT();
    ParseE1();
    }
}
```

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

ParseT1() {
if (lookahead==T_STAR)
{
     MatchToken(T_STAR);
     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 == T_LPAREN) {
   MatchToken(T_LPAREN); ParseE(); MatchToken(T_RPAREN);
   }
else
   MatchToken(T_ID);
```

Analiza descendent recursiva

Cum alegem intre doua productii?

Cum stim ce conditii punem la if?

Cand emitem erori?

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

```
ParseT1() {
if (lookahead==T_STAR) {
   MatchToken(T_STAR);
   ParseF();
   ParseT1();
   }
else if (lookahead == T_PLUS) { }
else if (lookahead == T_RPAREN) { }
else if (lookahead == T_EOF) { }
else throw error();
}
```

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

Cum punem conditiile?

- Folosim doua seturi de terminali – ‘First’ si ‘Follow’
  - Plus ‘Nullable’ – multime de neterminali ce pot deriva in ε.
- Setul de terminali-prefix ai neterminalului u - notat First(u)
  - Setul de terminali care apar pe prima pozitie intr-o derivare legala a lui u
  - Daca u=>* ε, atunci ε e in First(u)
- Setul de terminali care pot urma dupa u – notat Follow(u)

Cum construim FIRST

GRAMMAR:

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

SETS:

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 (pseudocod):

1. If X is a <u>terminal</u>, $FIRST(X) = \{X\}$
2. If $X \to \varepsilon$, then $\varepsilon \in FIRST(X)$
3. If $X \to Y_1\ Y_2 \cdots Y_k$
   and $Y_1 \cdots Y_{i-1} \overset{*}{\Rightarrow} \varepsilon$
   and $a \in FIRST(Y_i)$
   then $a \in FIRST(X)$
4. If $X \to Y_1 Y_2 \cdots Y_k$
   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}

Cum construim FOLLOW

GRAMMAR:

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

SETS:

FOLLOW(E) = ~~{$}~~ { **)**, **$**}

FOLLOW(E') = { **)**, **$**}

FOLLOW(T) = { **)**, **$**}

FOLLOW – pseudocod:

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

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

A si B sunt neterminali,
α si β siruri de terminali si neterminali

(Aho,Sethi,Ullman, pp. 189)

Cum construim FOLLOW

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

GRAMMAR:

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

SETS:

FOLLOW(E) = {**)**, **$**}

FOLLOW(E') = { **)**, **$**}

FOLLOW(T) = ~~{), $}~~ {+,), $}

FOLLOW rules:

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

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

(Aho,Sethi,Ullman, pp. 189)

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

Cum construim FOLLOW

GRAMMAR:

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

SETS:

FOLLOW(E) = {**)**, **$**}

FOLLOW(E') = { **)**, **$**}

FOLLOW(T) = {**+**, **)**, **$**}

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

FOLLOW rules:

1. If S is the start symbol, then $ ∈ FOLLOW(S)
2. If A → αBβ,
   and a ∈ FIRST(β)
   and a ≠ ε
   then a ∈ FOLLOW(B)
3. If A → αB
   and a ∈ FOLLOW(A)
   then a ∈ FOLLOW(B)

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

(Aho,Sethi,Ullman, pp. 189)

Cum construim FOLLOW

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

GRAMMAR:

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

SETS:

FOLLOW(E) = {**)**, **$**}

FOLLOW(E') = { **)**, **$**}

FOLLOW(T) = {**+**, **)**, **$**}

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

FOLLOW(F) = {**+**, **)**, **$**}

FOLLOW rules:

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

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

(Aho,Sethi,Ullman, pp. 189)

Cum construim FOLLOW

$FIRST(E') = \{+, \varepsilon\}$
$\boxed{FIRST(T') = \{*, \varepsilon\}}$
$FIRST(F) = \{(, id\}$
$FIRST(T) = \{(, id\}$
$FIRST(E) = \{(, id\}$

GRAMMAR:

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

SETS:

FOLLOW(E) = {**)**, **$**}

FOLLOW(E') = { **)**, **$**}

FOLLOW(T) = {**+**, **)**, **$**}

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

FOLLOW(F) = ~~{**+**, **)**, **$**}~~ {**+**, **∗**, **)**, **$**}

FOLLOW rules:

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

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

(Aho,Sethi,Ullman, pp. 189)

Algoritmul generic recursiv LL(1)

Pentru fiecare non-terminal A se creaza o functie de parsare.

Pentru fiecare regula A→ɑ se adauga un test
 if (lookahead in FIRST(ɑFOLLOW(A)))

Pentru fiecare nonterminal din ɑ se apeleaza functia de parsare.

Pentru fiecare terminal din ɑ , se verifica lookahead-ul (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
…
```

Recursivitatea stanga

Cand o gramatica are cel putin o productie de forma

$A \rightarrow A\alpha$

spunem ca este o gramatica **recursiva stanga**.

Analizoarele descendente nu functioneaza (fara backtracking) pe gramatici recursive stanga.

Recursivitatea poate sa nu fie imediata

$A \rightarrow B\alpha$

$B \rightarrow A\ \beta$

(Aho,Sethi,Ullman, pp. 176)

Eliminarea recursivitatii stanga

Se face prin rescrierea gramaticii

$List \rightarrow List\ Item \mid Item$

$\Rightarrow$

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

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

$\Rightarrow$

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

(Aho,Sethi,Ullman, pp. 176)

Eliminarea recursivitatii stanga

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

$\rightarrow$

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

Cazul general (recursivitate imediata):

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

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

Factorizare stanga

Sa analizam o instructiune if:

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

Pentru a o putea analiza LL, trebuie factorizata stanga:

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);
   }
}
```

Factorizare stanga

- Cazul general:

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

- Factorizat:

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

Eliminarea ambiguitatilor

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

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

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

- Apare explicita precedenta operatorilor, asociativitatea stanga sau dreapta

Eliminarea ambiguitatilor

- Productii ce pot produce ambiguitati:
  $X \rightarrow aAbAc$

- Cazul general:
  $A \rightarrow A\ B\ A \mid \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$

- Dezambiguizat:
  $A \rightarrow A'\ B\ A \mid A'$
  $A' \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$

“Dangling else”

- Ambiguu:

  Statement –> if Expr then Statement
  | if Expr then Statement else Statement
  | Other

“if Expr **then if** Expr **then** Other **else** Other”

- Factorizat ramane tot ambiguu:

  Statement –> if Expr then Statement CloseIf
  | Other
  CloseIf -> ε | else Statement

- Algoritmul de parsare poate rezolva implicit unele ambiguitati.

“Dangling else”

- Ambiguu:

  Statement –> if Expr then Statement
  | if Expr then Statement else Statement
  | Other

“**if** Expr **then if** Expr **then** Other **else** Other”

- Dezambiguizat:

  Statement -> Open | Closed
  Closed -> if Expr then Closed else Closed
  | Other
  Open -> if Expr then Statement
  | if Expr then Closed else Open

- Nu poate fi factorizat - limbajul nu este LL(1), dar este LR(1)

Automatizarea parsarii

Echivalenta cu un automat push-down

Parsarea se poate face cu un automat si o tabela.

(Limbaj == LL(1) daca nu exista conflicte in tabela!!!)

Un exemplu de parser LL

Grammar:

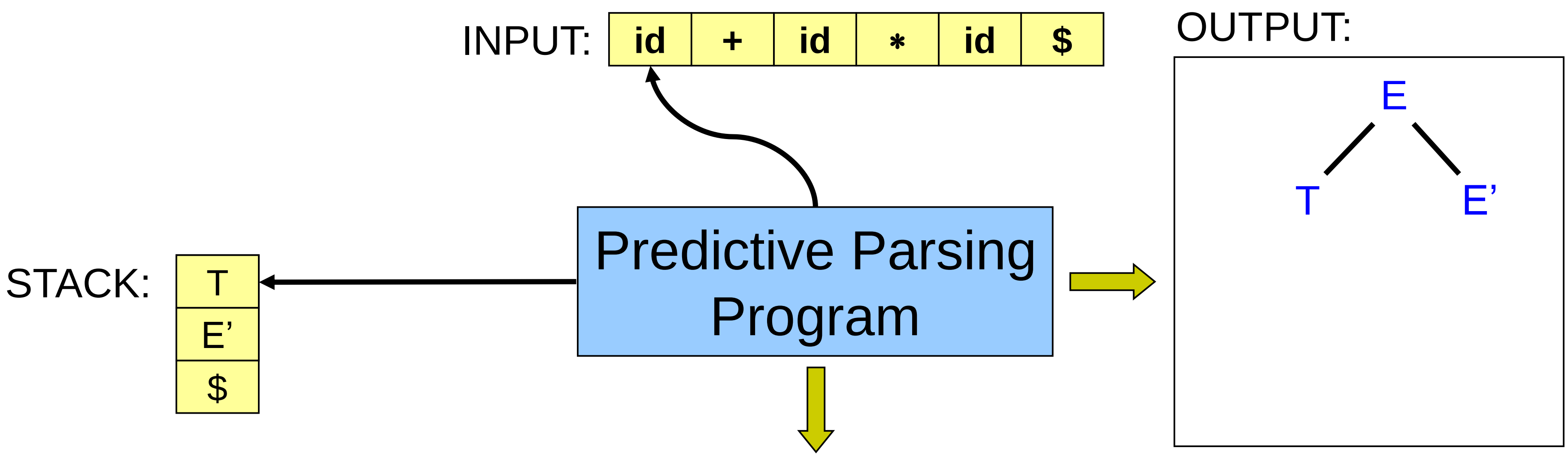
$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow +TE' \mid \varepsilon \\
T &\rightarrow FT' \\
T' &\rightarrow *FT' \mid \varepsilon \\
F &\rightarrow (\ E\ ) \mid \mathbf{id}
\end{aligned}
$$

Parsing Table:

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

(Aho,Sethi,Ullman, pp. 188)

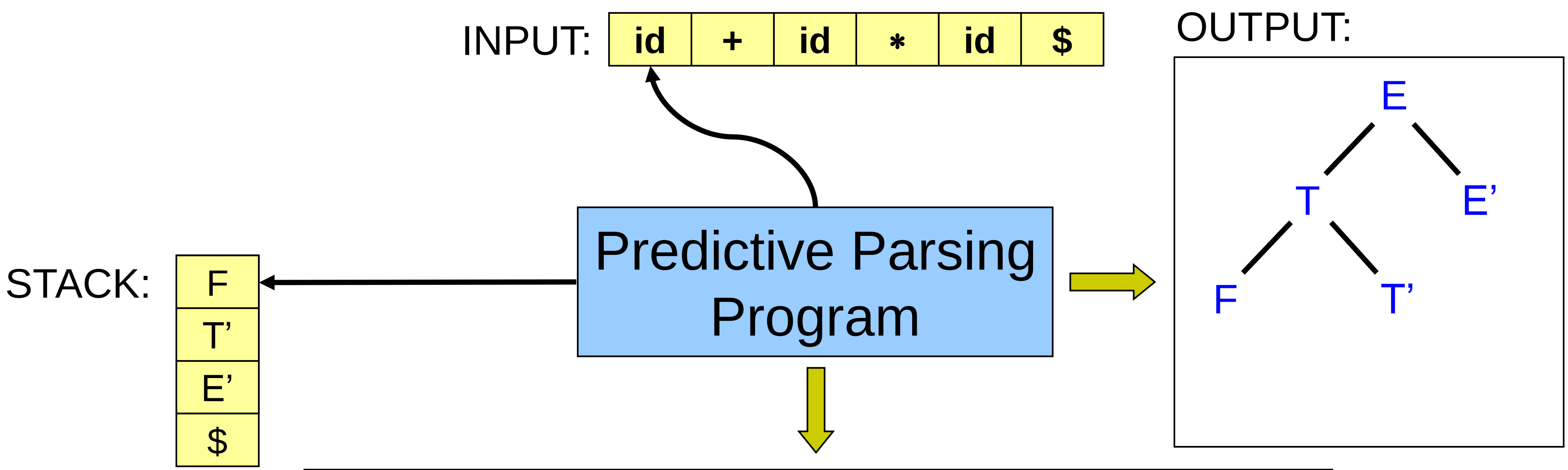
Un exemplu de parser LL



PARSING TABLE:

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	$F \rightarrow$ **id**			$F \rightarrow (E)$		

Un exemplu de parser LL

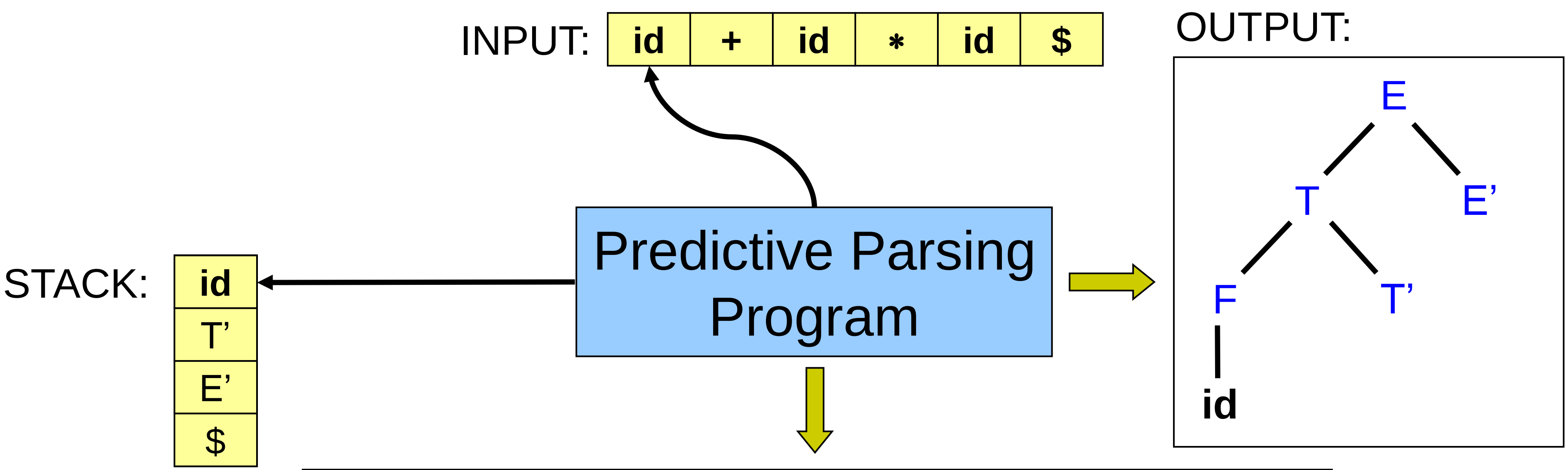


PARSING TABLE:

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. 186)

Un exemplu de parser LL



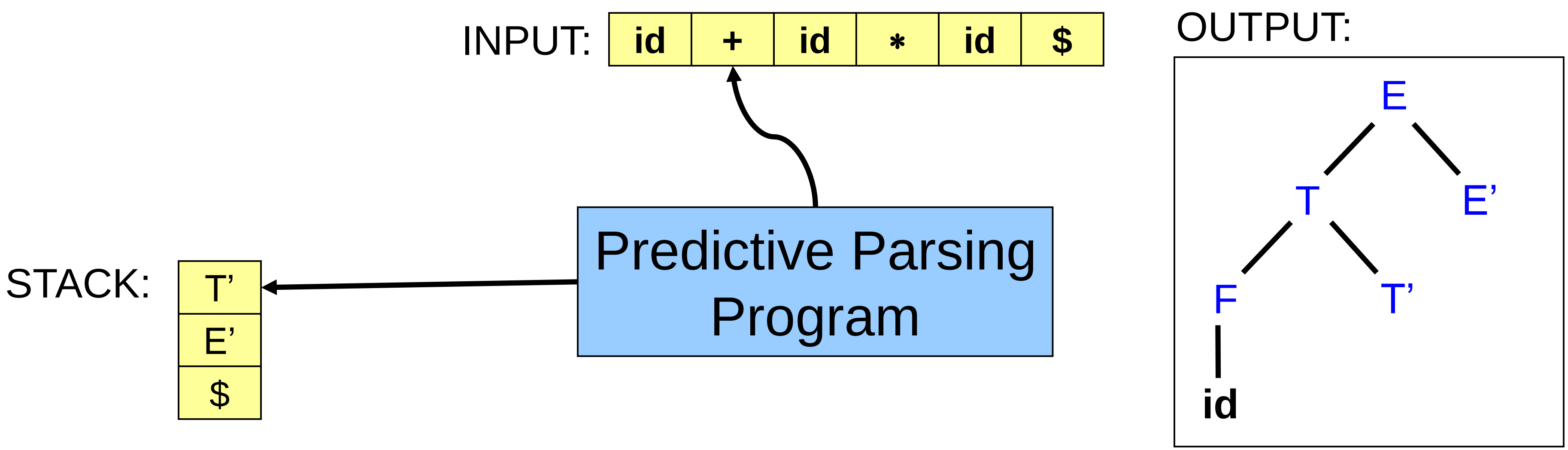
PARSING TABLE:

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to$ **id**			$F \to (E)$		

(Aho,Sethi, Ullman, pp. 188)

Un exemplu de parser LL

Actiunea cand Top(Stack) = input ≠ $: 'Pop' din stiva, avanseaza pe banda de intrare.

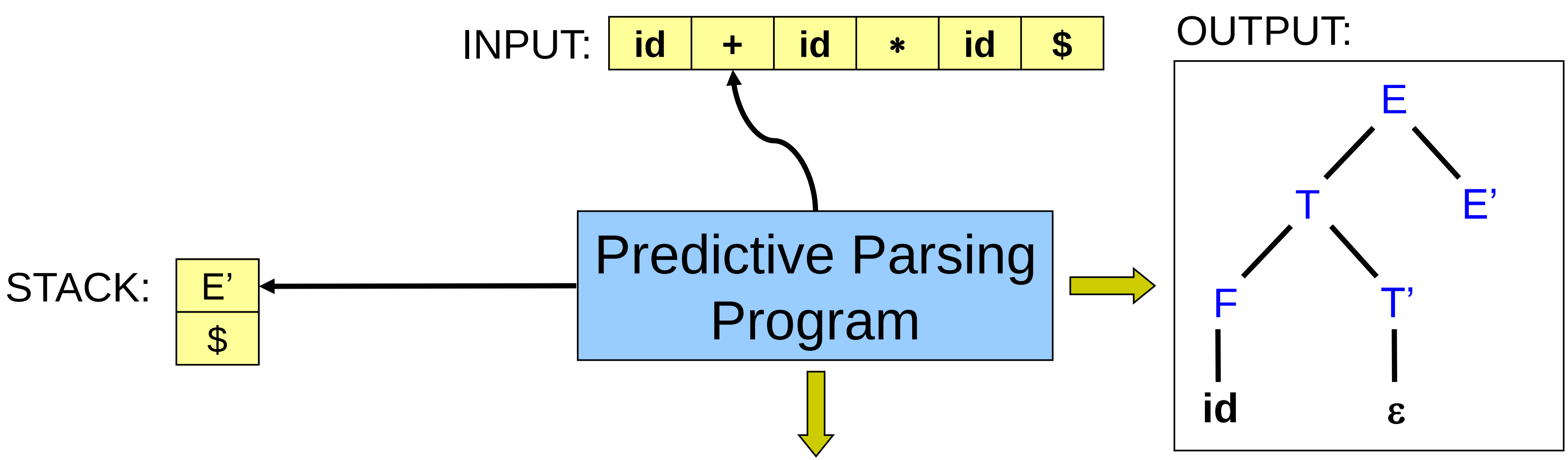


PARSING TABLE:

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)

Un exemplu de parser LL



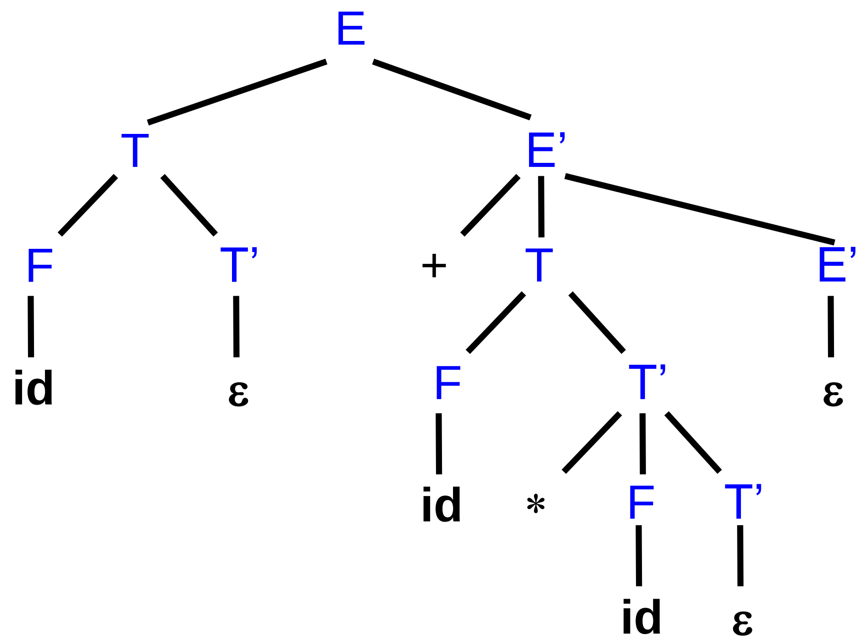
PARSING TABLE:

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	$F \rightarrow$ **id**			$F \rightarrow (E)$		

(Aho,Sethi, Ullman, pp. 188)

Un exemplu de parser LL

Si tot asa, se construieste urmatorul arbore de derivare:



$E' \rightarrow +TE'$

$T \rightarrow FT'$

$F \rightarrow \textbf{id}$

$T' \rightarrow * \, FT'$

$F \rightarrow \textbf{id}$

$T' \rightarrow \varepsilon$

$E' \rightarrow \varepsilon$

Cand Top(Stack) = input = $

Parserul se opreste si accepta intrarea.

(Aho,Sethi, Ullman, pp. 188)

Cum construim tabela?

- Folosim doua seturi de terminali – ‘First’ si ‘Follow’
  - Plus ‘Nullable’ – multime de neterminali ce pot deriva in ε.
- Setul de terminali-prefix ai neterminalului u - notat First(u)
  - Setul de terminali care apar pe prima pozitie intr-o derivare legala a lui u
  - Daca u=>* ε, atunci ε e in First(u)
- Setul de terminali care pot urma dupa u – notat Follow(u)

GRAMMAR:

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid$ **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(α), add $A \rightarrow \alpha$ to M[A, a]

PARSING TABLE:

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	$F \rightarrow$ **id**			$F \rightarrow (E)$		

Reguli pentru construit tabela de parsare

(Aho,Sethi,Ullman, pp. 190)

GRAMMAR:

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

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(α), add $A \rightarrow \alpha$ to M[A, a]

PARSING TABLE:

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)		

Reguli pentru construit tabela de parsare

(Aho,Sethi,Ullman, pp. 190)

GRAMMAR:

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

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 $a \in \text{FIRST}(\alpha)$, add $A \rightarrow \alpha$ to $M[A, a]$

PARSING TABLE:

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)		

Reguli pentru construit tabela de parsare

(Aho,Sethi,Ullman, pp. 190)

GRAMMAR:

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

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 $a \in \text{FIRST}(\alpha)$, add $A \rightarrow \alpha$ to $M[A, a]$

PARSING TABLE:

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

Reguli pentru construit tabela de parsare

(Aho,Sethi,Ullman, pp. 190)

GRAMMAR:

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

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(α), add $A \rightarrow \alpha$ to M[A, a]

PARSING TABLE:

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

Reguli pentru construit tabela de parsare

(Aho,Sethi,Ullman, pp. 190)

GRAMMAR:

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

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

PARSING TABLE:

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)		

Reguli pentru construit tabela de parsare

(Aho,Sethi,Ullman, pp. 190)

GRAMMAR:

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid$ **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(α), add $A \rightarrow \alpha$ to M[A, a]
2. If $A \rightarrow \alpha$:
   if $\varepsilon \in$ FIRST(α), add $A \rightarrow \alpha$ to M[A, b]
   for each terminal $b \in$ FOLLOW(A),

PARSING TABLE:

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	$F \rightarrow$ **id**			$F \rightarrow (E)$		

Reguli pentru construit tabela de parsare

(Aho,Sethi,Ullman, pp. 190)

GRAMMAR:

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

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 $M[A, \$]$

PARSING TABLE:

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)		

Reguli pentru construit tabela de parsare

(Aho,Sethi,Ullman, pp. 190)

Cand putem folosi parsere LL(1)

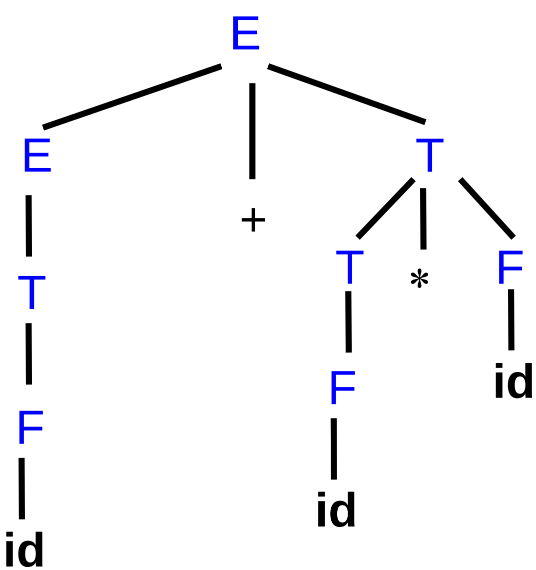
- Gramaticile LL(1) sunt gramatici neambigue, nerecursive stânga şi factorizate.
- Se poate arăta că o gramatică G este LL(1) dacă şi numai dacă pentru oricare două producţii de forma $A \to \alpha$, $A \to \beta$, cu $\alpha \neq \beta$ sunt satisfăcute următoarele condiţii:
  - $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$
  - dacă $\beta \Rightarrow^* \varepsilon$ atunci $FIRST(\alpha) \cap FOLLOW(A) = \emptyset$ iar dacă $\alpha \Rightarrow^* \varepsilon$ atunci $FIRST(\beta) \cap FOLLOW(A) = \emptyset$.

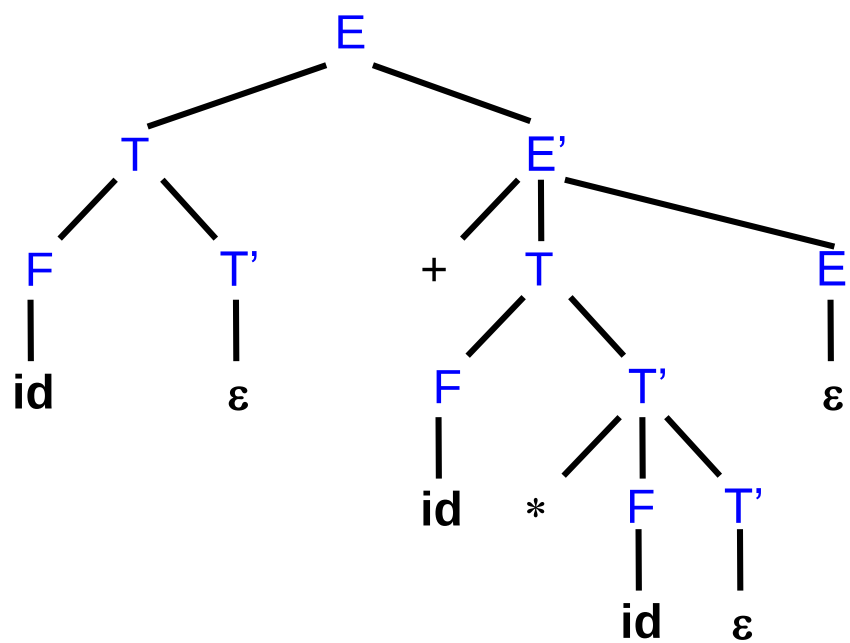
Avantaje/dezavantaje LL(1)

- Usor de scris parsere ‘de mana’
- Rapid, usor de inteles

- Trebuie transformata gramatica in general

➔ Arborele de derivare difera fata de arborele semantic





Algoritmi LL(...)

- Permit gramatici mai simple
- LL(k)
  - Foloseste pana la ‘k’ atomi lexicali in lookahead
- LL(*)
  - decizia luata folosind un DFA
  - nu este LL(∞)
  - Implementat de ANTLR

Generatoare de parsere LL

- ANTLR, JavaCC, Spirit (C++/Boost)

- Sintaxa extinsa pentru reguli (EBNF)

```
expr : term ( PLUS term )*
→ expr : term expr1;
  expr1 : PLUS term expr1 | ;
```

- Factorizare si eliminarea automata a recursivitatii

  (nu e posibil tot timpul)

```
type : type STAR | type array+ | type_name
  → type_name | (STAR | array+)*
```
- ambiguu

Reguli EBNF

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

Reguli EBNF
Parser descendent recursiv

Something?	**if** lookahead ∈ FIRST(Something)
	   code for Something ...
	else if lookahead ∉ FOLLOW(Something?)
	   ERROR();

Something*	**while** lookahead ∈ FIRST(Something)
	   code for Something ...
	if lookahead ∉ FOLLOW(Something*) **then**
	   ERROR();

Something+	**do**
	   if lookahead ∉ FIRST(Something) **then**
	      ERROR();
	   code for Something ...
	while lookahead ∉ FOLLOW(Something+);

ANTLR

- Genereaza parsere descendent recursive
- Genereaza cod Java din descrierea gramaticii
- Transformari de arbori, listeners, visitors
- Implementeaza un algoritm mai complex ca LL(1), adaptive LL(*)

```
grammar Exp;
add : mul ( '+' mul | '-')* ;
mul : atom ( '*' atom | '/' atom )* ;
atom : Number | '(' add ')' ;
Number : ('0'..'9')+ ('.' ('0'..'9')+)? ;
WS :   (' ' | '\t' | '\r'| '\n') {$channel=HIDDEN;} ;
```

Spirit

- Gramatica in C++. Nu se genereaza alt cod sursa.
- Foloseste templates si redefinirea operatorilor.

```
template <typename Iterator> struct parser
  : qi::grammar<Iterator, map<string, string>()> {
    parser() : parser::base_type(query) {

      query = pair >> *((qi::lit(';') | '&') >> pair);
      pair  = key >> -('=' >> value);
      key   = qi::char_("a-zA-Z_") >> *qi::char_("a-zA-Z_0-9");
      value = +qi::char_("a-zA-Z_0-9"); }

qi::rule<Iterator, map<string, string>()> query;
qi::rule<Iterator, pair<string, string>()> pair;
qi::rule<Iterator, string()> key, value; };
```