Análise Sintática LL(1)

$S \rightarrow E\ \$$
$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow \text{id}$
$F \rightarrow (E)$

É possível gerar um parser LL(1) para essa gramática?

	Nullable	FIRST	FOLLOW
E	N	(id	+) $
T	N	(id	+ *) $
F	N	(id	+ *) $
S	N	(id	

Recursão à Esquerda

$S \rightarrow E\ \$$

$E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow \text{id}$

$F \rightarrow (E)$

É possível gerar um parser LL(1) para essa gramática?

	Nullable	FIRST	FOLLOW
E	N	(id	+) $
T	N	(id	+ *) $
F	N	(id	+ *) $
S	N	(id	

Problema:

- A função que implementa E precisa chamar a si mesma caso escolha E+T.
- Porém, é a primeira ação dela, antes de avançar na cadeia de entrada
- Laço infinito!
- Acontece devido à recursão à esquerda

Recursão à Esquerda

$S \rightarrow E\ \$$

$E \rightarrow E - T$

$E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow T / F$

$T \rightarrow F$

$F \rightarrow \text{id}$

$F \rightarrow \text{num}$

$F \rightarrow (E)$

Gramáticas com recursão à esquerda não podem ser LL(1).

Fatoração (recursão à direita)!

- $E \rightarrow \text{TE'}$
- $E' \rightarrow +TE'$
- $E' \rightarrow$

Recursão à Esquerda

Generalizando:

- Tendo $X \rightarrow X\,\gamma$ e $X \rightarrow \alpha$, onde α não começa com X
- Derivamos strings da forma $\alpha\gamma^*$
  - α seguido de zero ou mais γ.
- Podemos reescrever:

$$\begin{pmatrix} X \rightarrow X\ \gamma_1 \\ X \rightarrow X\ \gamma_2 \\ X \rightarrow \alpha_1 \\ X \rightarrow \alpha_2 \end{pmatrix} \Longrightarrow \begin{pmatrix} X \rightarrow \alpha_1\ X' \\ X \rightarrow \alpha_2\ X' \\ X' \rightarrow \gamma_1\ X' \\ X' \rightarrow \gamma_2\ X' \\ X' \rightarrow \end{pmatrix}$$

Recursão à Esquerda

$S \rightarrow E\ \$$

$E \rightarrow E - T$

$E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow T / F$

$T \rightarrow F$

$F \rightarrow \text{id}$

$F \rightarrow \text{num}$

$F \rightarrow (E)$

Eliminado Recursão à Esquerda

$S \rightarrow E\ \$$

$E \rightarrow E - T$

$E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

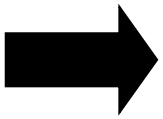
$T \rightarrow T / F$

$T \rightarrow F$

$F \rightarrow \text{id}$

$F \rightarrow \text{num}$

$F \rightarrow (E)$



$S \rightarrow E\ \$$

$E \rightarrow T\ E'$

$E' \rightarrow + T\ E'$

$E' \rightarrow - T\ E'$

$E' \rightarrow$

$T \rightarrow F\ T'$

$T' \rightarrow * F\ T'$

$T' \rightarrow / F\ T'$

$T' \rightarrow$

$F \rightarrow id$

$F \rightarrow num$

$F \rightarrow (E)$

Eliminado Recursão à Esquerda

$S \rightarrow E\ \$$
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E'$
$E' \rightarrow -\ T\ E'$
$E' \rightarrow$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T'$
$T' \rightarrow /\ F\ T'$
$T' \rightarrow$
$F \rightarrow id$
$F \rightarrow num$
$F \rightarrow (E)$

	nullable	FIRST	FOLLOW
S	no	(id num	
E	no	(id num	) $
E'	yes	+ -	) $
T	no	(id num	) + - $
T'	yes	* /	) + - $
F	no	(id num	) * / + - $

Eliminado Recursão à Esquerda

$S \rightarrow E\ \$$
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E'$
$E' \rightarrow -\ T\ E'$
$E' \rightarrow$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T'$
$T' \rightarrow /\ F\ T'$
$T' \rightarrow$
$F \rightarrow id$
$F \rightarrow num$
$F \rightarrow (E)$

	nullable	FIRST	FOLLOW
S	no	(id num	
E	no	(id num	) $
E'	yes	+ -	) $
T	no	(id num	) + - $
T'	yes	* /	) + - $
F	no	(id num	) * / + - $

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

*** Algumas colunas da tabela foram omitidas**

Recursão à Esquerda Indireta

A → Cd
B → Ce
C → A
C → B
C → f

Recursão à Esquerda Indireta

$A \rightarrow Cd$
$B \rightarrow Ce$
$C \rightarrow A$
$C \rightarrow B$
$C \rightarrow f$

$C \rightarrow A \rightarrow Cd$
$C \rightarrow B \rightarrow Ce$

$A \rightarrow Cd$
$B \rightarrow Ce$
$C \rightarrow Cd$
$C \rightarrow Ce$
$C \rightarrow f$

Recursão à Esquerda Indireta

$A \rightarrow Cd$
$B \rightarrow Ce$
$C \rightarrow A$
$C \rightarrow B$
$C \rightarrow f$

$C \rightarrow A \rightarrow Cd$
$C \rightarrow B \rightarrow Ce$

$A \rightarrow Cd$
$B \rightarrow Ce$
$C \rightarrow Cd$
$C \rightarrow Ce$
$C \rightarrow f$

$A \rightarrow Cd$
$B \rightarrow Ce$
$C \rightarrow fC'$
$C' \rightarrow dC'$
$C' \rightarrow eC'$
$C' \rightarrow$

Eliminando Recursão à Esquerda Indireta

```
impose an order on the nonterminals, A1, A2, ..., An

for i ← 1 to n do;
    for j ← 1 to i - 1 do;
        if ∃ a production Ai → Aj γ
            then replace Ai → Aj γ with one or more
                 productions that expand Aj
    end;
    rewrite the productions to eliminate
         any direct left recursion on Ai
end;
```

Fatoração à Esquerda

- Um outro problema para *predictive parsing* ocorre em situações do tipo:

$$S \rightarrow \text{if } E \text{ then } S \text{ else } S$$
$$S \rightarrow \text{if } E \text{ then } S$$

- Regras do mesmo não terminal começam com os mesmo símbolos

Fatoração à Esquerda

$$S \rightarrow \text{if } E \text{ then } S \text{ else } S$$
$$S \rightarrow \text{if } E \text{ then } S$$

- Criar um novo não-terminal para os finais permitidos:

$$S \rightarrow \text{if } E \text{ then } S\ X$$
$$X \rightarrow$$
$$X \rightarrow \text{else } S$$

Análise Descendente (Predictive Parsing)

```
void S() { E(); eat(EOF); }

void E() {
    switch (tok) {
        case ?: E(); eat(PLUS); T(); break;
        case ?: E(); eat(MINUS); T(); break;
        case ?: T(); break;
        default: error(); }
}

void T() {
    switch (tok) {
        case ?: T(); eat(TIMES); F(); break;
        case ?: T(); eat(DIV); F(); break;
        case ?: F(); break;
        default: error(); }
}
```

Funciona ???

$S \rightarrow E\ \$$
$E \rightarrow E + T$
$E \rightarrow E - T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow T / F$
$T \rightarrow F$
$F \rightarrow$ id
$F \rightarrow$ num
$F \rightarrow (E)$

Análise Descendente (Predictive Parsing)

$S \rightarrow E\,\$$
$E \rightarrow T\,E'$
$E' \rightarrow +\,T\,E'$
$E' \rightarrow -\,T\,E'$
$E' \rightarrow$
$T \rightarrow F\,T'$
$T' \rightarrow *\,F\,T'$
$T' \rightarrow /\,F\,T'$
$T' \rightarrow$
$F \rightarrow id$
$F \rightarrow num$
$F \rightarrow (E)$

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

*** Algumas colunas da tabela foram omitidas**

Análise Descendente (Predictive Parsing)

$S \rightarrow E\,\$$
$E \rightarrow T\,E'$
$E' \rightarrow +\,T\,E'$
$E' \rightarrow -\,T\,E'$
$E' \rightarrow$
$T \rightarrow F\,T'$
$T' \rightarrow *\,F\,T'$
$T' \rightarrow /\,F\,T'$
$T' \rightarrow$
$F \rightarrow id$
$F \rightarrow num$
$F \rightarrow (E)$

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

* Algumas colunas da tabela foram omitidas

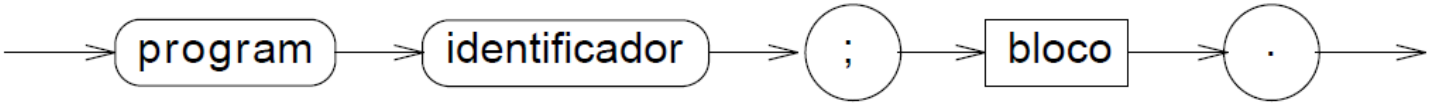
```
void T() {
    switch (tok) {
     case ID:
     case NUM:
     case LPAREN: F(); Tprime(); break;
     default: print("expected id, num, or left-paren");
}}

void Tprime() {
 switch (tok) {
     case PLUS: break;
     case TIMES: eat(TIMES); F(); Tprime(); break;
     case RPAREN: break;
     case EOF: break;
     default: print("expected +, *, right-paren, or end-of-file");
}}
```

PASCAL

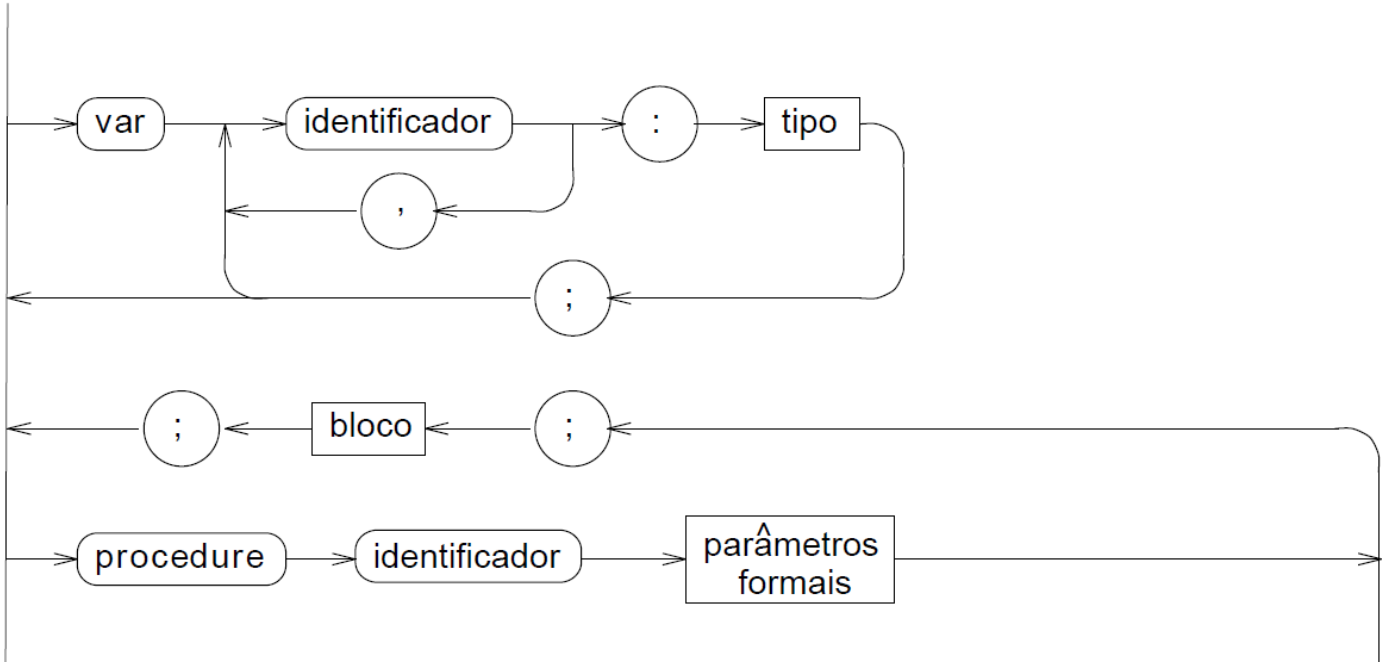
```
program ex;
    var m: integer;
function F(n:integer; var k:integer):integer;
var p,q:integer;
begin
    if n<2 then
    begin
        F:=n;
        k:=0
    end
    else
    begin
        F:=F(n-1,p)+F(n-2,q);
        k:=p+q+1
    end;
    write(n,k)
end
begin
    write(F(3,m),m);
end.
```

PASCAL - Cartas Sintáticas

programa:

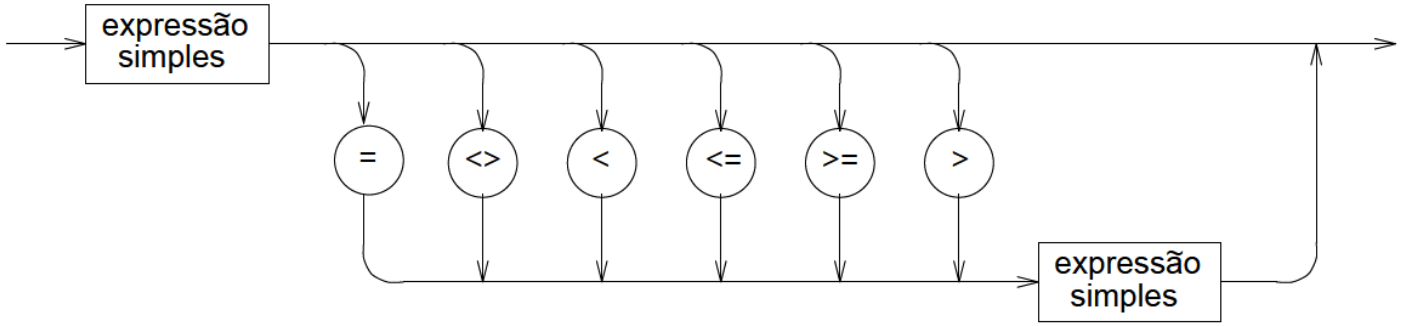


bloco:



PASCAL - Cartas Sintáticas

expressão:




```
expressao → expressao_simples
expressao → expressao =  expressao_simples
expressao → expressao <> expressao_simples
expressao → expressao <  expressao_simples
expressao → expressao <= expressao_simples
expressao → expressao >= expressao_simples
expressao → expressao >  expressao_simples
```