

Gramáticas - 4

Notação **EBNF** - BNF estendida

Notação **usada com o YACC** (gerador de parsers Bottom-up)

Notações

- BNF: já vista
- EBNF: existem pelo menos 3 estilos principais de BNF estendidas
 - Derivada de expressões regulares
 - Baseada na definição de Wirth
 - **Augmented Backus-Naur Form (ABNF)**, uma extensão do BNF documentada no [RFC 2234](http://www.faqs.org/rfcs/rfc2234.html)
<http://www.faqs.org/rfcs/rfc2234.html>
- Para ser usada com o gerador de compiladores YACC
- YACC mais informal

EBNF derivada de ER

- ? Significa um item opcional (as vezes aparece como *opt*)
- + e * significam repita 1 ou mais vezes, ou 0 ou mais vezes

```
number = digit+ .  
identifier = letter (letter | digit)* .  
functioncall = functionname "(" parameterlist? ")" .
```

EBNF baseada em Wirth

- [] significa um item opcional
{ } significa repita 0 ou mais vezes

```
number = digit {digit} .  
identifier = letter {letter | digit} .  
functioncall = functionname "(" [parameterlist] ")" .
```

ABNF

- [] significa um item opcional
- * significa repetição, com números extra para indicar limites
- (e.g. * significa 0 ou mais, 1* significa 1 ou mais, 2*5 significa entre 2 e 5 vezes).

```
number = 1*digit .  
identifier = letter *(letter | digit) .  
functioncall = functionname "(" [parameterlist] ")" .
```

EBNF baseada em Wirth (Módulo-2)

- <http://cui.unige.ch/db-research/Enseignement/analyseinfo/Modula2/expression.html>
- $\text{expression} ::= \text{simple_expression} [("=" | "\#" | "<>" | "<" | "<=" | ">" | ">=" | "IN") \text{simple_expression}]$
- $\text{simple_expression} ::= ["+" | "-"] \text{term} \{ ("+" | "-" | "OR") \text{term} \}$
- $\text{term} ::= \text{factor} \{ ("*" | "/" | "DIV" | "MOD" | "AND" | "&") \text{factor} \}$
- $\text{factor} ::= \text{number} | \text{string} | \text{set} | \text{designator} ["(" [\text{expression} \{ "," \text{expression} \} "]")] | "(" \text{expression} ")" | "NOT" \text{factor}$

Regras de Precedência, Prioridade e Associatividade

- Ajudam na decisão da interpretação correta de expressões nas LP
- Def: Um operador é **associado à esquerda** se os operandos são agrupados da esq para dir e **associado à direita** se são agrupados da dir para esq.
- Está relacionada com a **posição da recursão nas regras** que permitem um operador ser aplicado mais de uma vez. Por convenção $a+a+a = (a+a) + a$

Pascal

- 1) Expressões parentizadas são resolvidas primeiro
- 2) Not associados a direita
- 3) * / div mod and associados a esquerda
- 4) + - or " "
- 5) < > <> >= <= = ocorrem uma vez

$EXP \rightarrow ES \langle \rangle ES \mid ES$ (uma vez)

$ES \rightarrow ES + T \mid T$ (esq)

$T \rightarrow T * F \mid F$ (esq)

$F \rightarrow \text{not } F \mid U$ (dir)

$U \rightarrow \langle \text{inteiro} \rangle \mid \langle \text{real} \rangle \mid (EXP)$ { as duas últimas regras geralmente aparecem juntas na gramática do Pascal }

- Os níveis de **prioridade** indicam a quais operadores são permitidos agrupar seus operandos primeiro (resolver primeiro).

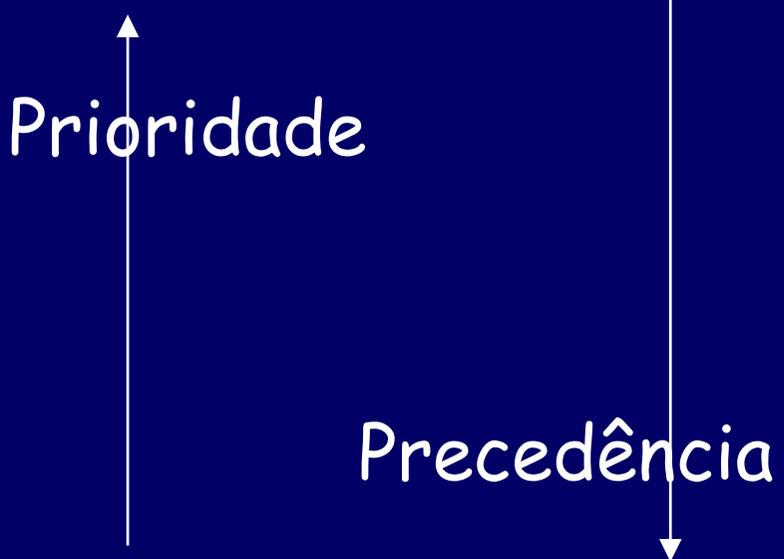
Not

* / div mod and

+ - or

< > <> >= <= =

Prioridade



Precedência

- Quanto maior a **precedência** maior o escopo (abrangência na árvore de derivação).

Notação EBNF - BNF estendida

- Observem que, na gramática de Módulo-2 como os **não terminais** não aparecem entre colchetes angulares os **terminais** precisam ser discriminados.
 - Nesta gramática eles aparecem entre aspas. Podem também aparecer negritados.
- Notação propícia para implementar um tipo de analisador sintático chamado de **DESCENDENTE RECURSIVO com procedimentos recursivos**. Veremos ela no nosso curso.
- Nos analisadores descendentes (**Top-Down**) a gramática **não** pode ter recursão à esquerda.
- Mas como eliminá-la sem alterar a associação de operadores como **+ - * /** que são associados à esquerda??

Fazemos uso de 3 operações:

1) Fatoração

$$E \rightarrow T + E \mid T \Rightarrow T (+E \mid \lambda)$$

De uma forma geral, se:

$$A \rightarrow \beta\gamma_1 \mid \beta\gamma_2 \mid \dots \mid \beta\gamma_n \text{ com } \beta \leftrightarrow \lambda$$

Podemos fatorá-la em:

$$A \rightarrow \beta (\gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_n)$$

2) Substituição da notação recursiva pela iterativa quando houver recursão à esquerda.

• $E \rightarrow E + T \mid T$

$E \Rightarrow E + T \Rightarrow E + T + T \Rightarrow^* T + T + T + \dots + T$

Notação de Wirth usa $\{\alpha\}$ para denotar a repetição de α zero ou mais vezes.

Portanto, $E \rightarrow T \{+T\}$

3) **Combinação da Fatoração e Substituição da Recursão**

• $E \rightarrow E + T \mid E - T \mid T$

$E \rightarrow E (+ T \mid - T) \mid T$

$E \rightarrow T \{+ T \mid - T\}$

$E \rightarrow T \{(+ \mid -) T\}$

EBNF (Wirth)

$[\alpha] = \alpha \mid \lambda$

$\{\alpha\} = \alpha^*$

Na gramática em EBNF **não** podemos ver as regras de associatividade (não há recursão à esquerda),

MAS a associatividade de operadores é implementada diretamente no compilador segundo a tabela fornecida pela linguagem.

Exercício: passar para EBNF

$E ::= E + T \mid E - T \mid + T \mid - T \mid T$

$T ::= T * F \mid T / F \mid F$

$F ::= a \mid b \mid (E)$

Notação usada com o YACC (PASCAL)

- <http://www.moorecad.com/standardpascal/pascal.y>
- **expression** : simple_expression | simple_expression relop simple_expression ;
- **simple_expression** : term | **simple_expression** addop term ;
- **term** : factor | **term** mulop factor ;
- **factor** : sign **factor** | exponentiation ;
- **exponentiation** : primary | primary **STARSTAR** exponentiation ;
- **primary** : variable_access | unsigned_constant | function_designator | set_constructor | **LPAREN** expression **RPAREN** | **NOT** primary ;

Notação usada com o YACC (Yet Another Compiler Compiler) - **Analizador Bottom-up**

- Os símbolos do VT aparecem logo no começo da notação em maiúsculas (tokens):

```
%{  
/*
```

```
* grammar.y
```

```
Pascal grammar in Yacc format, based originally on BNF given
```

```
* in "Standard Pascal -- User Reference Manual", by Doug Cooper.
```

```
* This in turn is the BNF given by the ANSI and ISO Pascal standards,  
and so, is PUBLIC DOMAIN. The grammar is for ISO Level 0 Pascal.
```

```
*/
```

```
%}
```

```
%token AND ARRAY ASSIGNMENT CASE CHARACTER_STRING COLON COMMA CONST DIGSEQ  
%token DIV DO DOT DOTDOT DOWNTO ELSE END EQUAL EXTERNAL FOR FORWARD FUNCTION  
%token GE GOTO GT IDENTIFIER IF IN LABEL LBRAC LE LPAREN LT MINUS MOD NIL NOT  
%token NOTEQUAL OF OR OTHERWISE PACKED PBEGIN PFILE PLUS PROCEDURE PROGRAM RBRAC  
%token REALNUMBER RECORD REPEAT RPAREN SEMICOLON SET SLASH STAR STARSTAR THEN  
%token TO TYPE UNTIL UPARROW VAR WHILE WITH
```

- As regras da gramática terminam com um ;
- A recursão a esquerda é permitida, assim como a direita

Dêem as regras de:

- Prioridade
- Associatividade

de cada operador do Pascal com a ajuda da gramática usada com o YACC.

Como o parser usado no YACC é bottom-up podemos ter recursão a esquerda e a direita. Assim, fica muito fácil dizer a associatividade de um operador ao olhar para a gramática.

Prioridade e Associatividades dos operadores segundo esta gramática do PASCAL

Associatividade



Not ()

**

Sinais

* DIV / MOD AND

+ - OR

> >= < <= <> =

Dir-esq (not)

Dir-esq (**)

Dir-esq (sinais)

Esq-dir (mulop)

Esq-dir (adop)

Não há com relop

Gramática de C# - notação YACC-ish

- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csspec/html/vclrfcsharpsec_c.asp

conditional-expression:

conditional-or-expression

conditional-or-expression ? expression : expression

assignment:

unary-expression assignment-operator expression

assignment-operator: **one of**

= += -= *= /= %= &= |= ^= <<= >>=

expression:

conditional-expression

assignment

Observem como as regras alternativas aparecem em linhas diferentes

19

Nenhuma meta-informação separa terminal de não terminal

Gramática de C# adaptada para BNF

$\langle \text{expression} \rangle ::= \langle \underline{\text{conditional-expression}} \rangle$
 $\quad \quad \quad | \langle \underline{\text{assignment}} \rangle$

$\langle \text{conditional-expression} \rangle ::= \langle \underline{\text{conditional-or-expression}} \rangle$
 $\quad \quad \quad | \langle \underline{\text{conditional-or-expression}} \rangle ? \langle \underline{\text{expression}} \rangle : \langle \underline{\text{expression}} \rangle$

$\langle \text{assignment} \rangle ::= \langle \underline{\text{unary-expression}} \rangle \langle \underline{\text{assignment-operator}} \rangle \langle \underline{\text{expression}} \rangle$

$\langle \text{assignment-operator} \rangle ::= = \mid += \mid -= \mid *= \mid /= \mid \% = \mid \& = \mid |= \mid ^= \mid \ll = \mid \gg =$

Mais formal

Resumo sobre notações

- Em parsers top down devemos usar uma notação de gramática que não permita a recursão a esquerda, que é proibida → EBNF
- Em parsers bottom-up podemos usar uma notação BNF e usar recursão a esquerda e direita. Usamos também a notação apropriada ao YACC quando pretendemos usar tal gerador de analisadores sintáticos.