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**

- 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 (Módula-2)

- http://cui.unige.ch/db-research/Enseignement/analyseinfo/Modula2/expression.html

- expression ::= simple_expression [("=" | "#" | "<>" | "<" | "<=" | ">" | ">=" | "IN") simple_expression]

- simple_expression ::= ["+" | "-"] term { ("+" | "-" | "OR") term }

- term ::= factor { ("*" | "/" | "DIV" | "MOD" | "AND" | "&") factor }

- factor ::= number | string | set | designator ["(" [expression { "," expression }] ")"] | "(" expression ")" | "NOT" factor

Notação EBNF – BNF estendida

- Observem que, na gramática de Módula-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 -> T + E | T => T (+E | λ)

De uma forma geral, se:

A -> $\beta\gamma_1$ | $\beta\gamma_2$| ... | $\beta\gamma_n$ com β <> λ

Podemos fatorá-la em:

A -> β (γ_1 | γ_2 | ... | γ_n)

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

- E -> E + T | T

E => E + T => E + T + T =>* T + T + T + ... + T

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

Portanto, E -> T {+T}

3) Combinação da Fatoração e Substituição da Recursão

- E -> E + T | E - T | T

E -> E (+ T | -T) | T
E -> T {+T | -T}
E -> T {(+|-) T}

EBNF (Wirth)
[α] = α | λ
{α} = α^*

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 | E - T | + T | - T | T

T ::= T * F | T / F | F

F ::= a | b | (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) – Analisador 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 ( )	Dir-esq (not)
**	Dir-esq (**)
Sinais	Dir-esq (sinais)
* DIV / MOD AND	Esq-dir (mulop)
+ - OR	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/vclrfcsharpspec_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

Nenhuma meta-informação separa terminal de não terminal

Gramática de C# adaptada para BNF

<expression>::= <conditional-expression>
|<assignment>

<conditional-expression>::= <conditional-or-expression>
| <conditional-or-expression> ? <expression> : <expression>

<assignment>::= <unary-expression> <assignment-operator> <expression>

<assignment-operator>::= = | += | -= | *= | /= | %= | &= | |= | ^= | <<= | >>=

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

- Usaremos um parser top-down no projeto de compiladores (JAVACC)

- 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.