Insper

Lógica da Computação

Aula 05

Raul Ikeda

2º semestre de 2018

Aula Passada

1. Gramática Regulares
2. Autômatos Finitos

Esta Aula

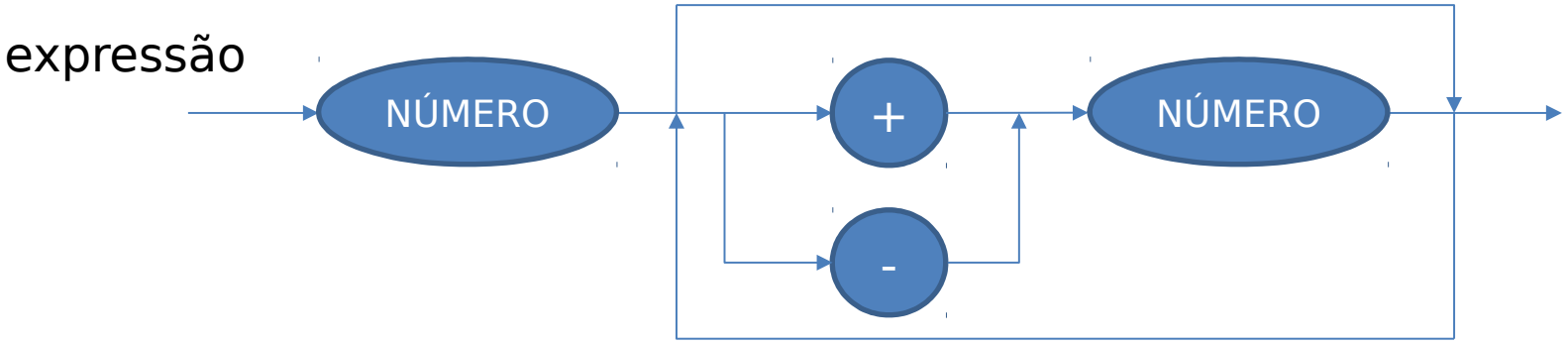
1. EBNF
2. Precedência de Operadores
3. Melhorias no Compilador:
    1. Multiplicação e Divisão
    2. Comentários

Extended BNF

- Junto com a ABNF (augmented BNF), é uma das derivações da BNF – Backus Naur Form.
- Criada por Niklaus Wirth, serve também para representar uma linguagem formal e muito usada para linguagens de programação.
- Consiste de regras equivalentes às regras de produção de uma gramática.
- Sua formulação é muito parecida com as expressões regulares.

- Vantagens sobre a BNF:
  - Tamanho mais enxuto.
  - Inclui opções e repetições.
  - Separa os símbolos com aspas, o que permite que os símbolos da própria notação possam também ser usados.
  - Possui terminal, permite mais de uma regra em uma linha.
  - Comentários

EBNF – Exemplo



EBNF:

```
expressão = número, { (“+” | “-”), número} ;
```

Obs: considerem número como qualquer número inteiro.

Exemplo EBNF Python

10. Full Grammar specification

This is the full Python grammar, as it is read by the parser generator and used to parse Python source files:

```
# Grammar for Python

# NOTE WELL: You should also follow all the steps listed at
# https://devguide.python.org/grammar/

# Start symbols for the grammar:
#       single_input is a single interactive statement;
#       file_input is a module or sequence of commands read from an input file;
#       eval_input is the input for the eval() functions.
# NB: compound_stmt in single_input is followed by extra NEWLINE!
single_input: NEWLINE | simple_stmt | compound_stmt NEWLINE
file_input: (NEWLINE | stmt)* ENDMARKER
eval_input: testlist NEWLINE* ENDMARKER

decorator: '@' dotted_name [ '(' [arglist] ')' ] NEWLINE
decorators: decorator+
decorated: decorators (classdef | funcdef | async_funcdef)

async_funcdef: ASYNC funcdef
funcdef: 'def' NAME parameters ['->' test] ':' suite

parameters: '(' [typedargslist] ')'
typedargslist: (tfpdef ['=' test] (',' tfpdef ['=' test])* [',' [
        '*' [tfpdef] (',' tfpdef ['=' test])* [',' ['**' tfpdef [',']]]
      | '**' tfpdef [',']]]
  | '*' [tfpdef] (',' tfpdef ['=' test])* [',' ['**' tfpdef [',']]]
  | '**' tfpdef [','])
tfpdef: NAME [':' test]
```

Fonte: https://docs.python.org/3/reference/grammar.html

Extended BNF

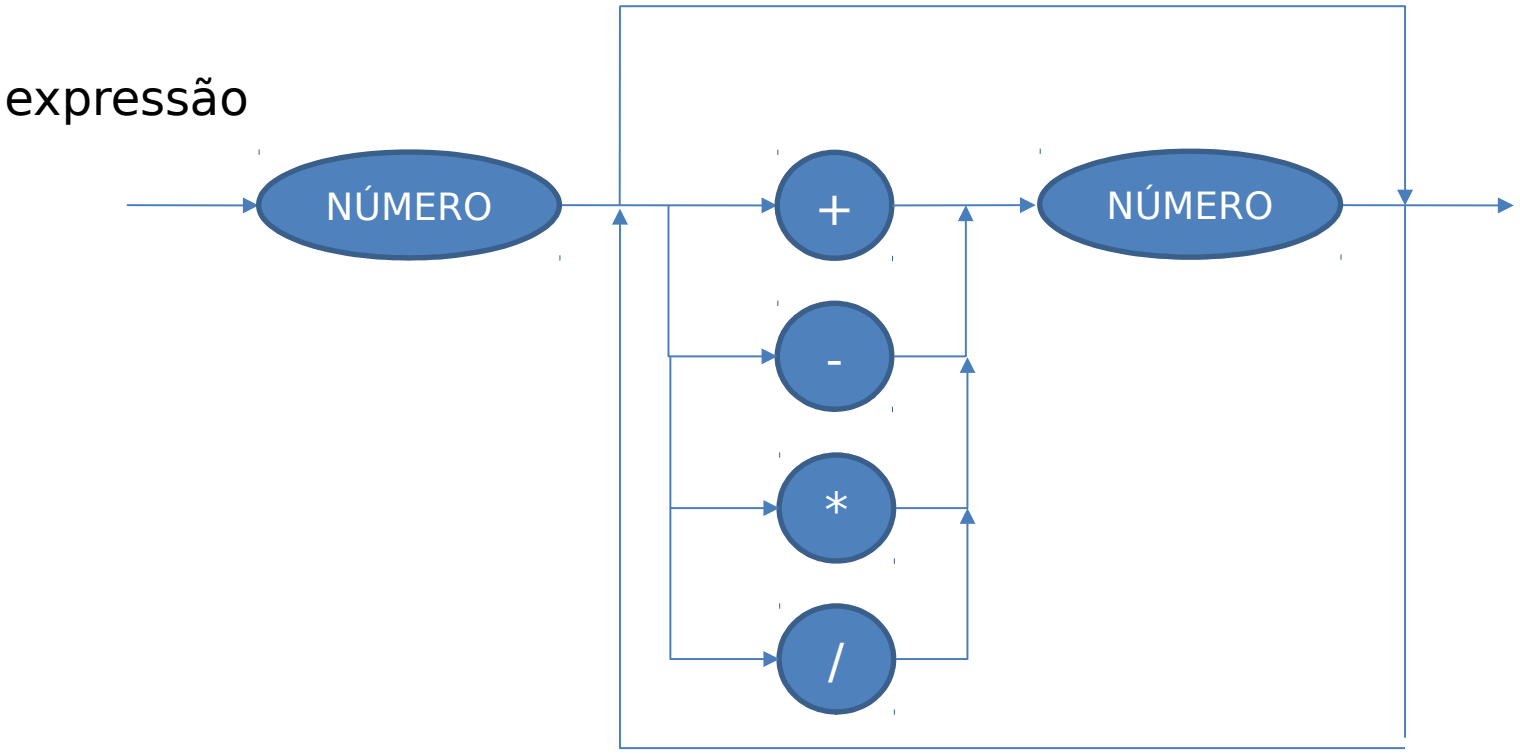
- Utiliza os seguintes símbolos:

Notação	Significado
=	Definição
,	Concatenação
;	Término
|	Condicional
“ ... “	símbolo terminal
[...]	Opcional
{ ... }	Repetição
(...)	Agrupamento
(* ... *)	Comentário

- Toda EBNF pode ser convertida em BNF e vice-versa.

Multiplicação e Divisão

- Para incluir a multiplicação e divisão no compilador, uma primeira tentativa seria:

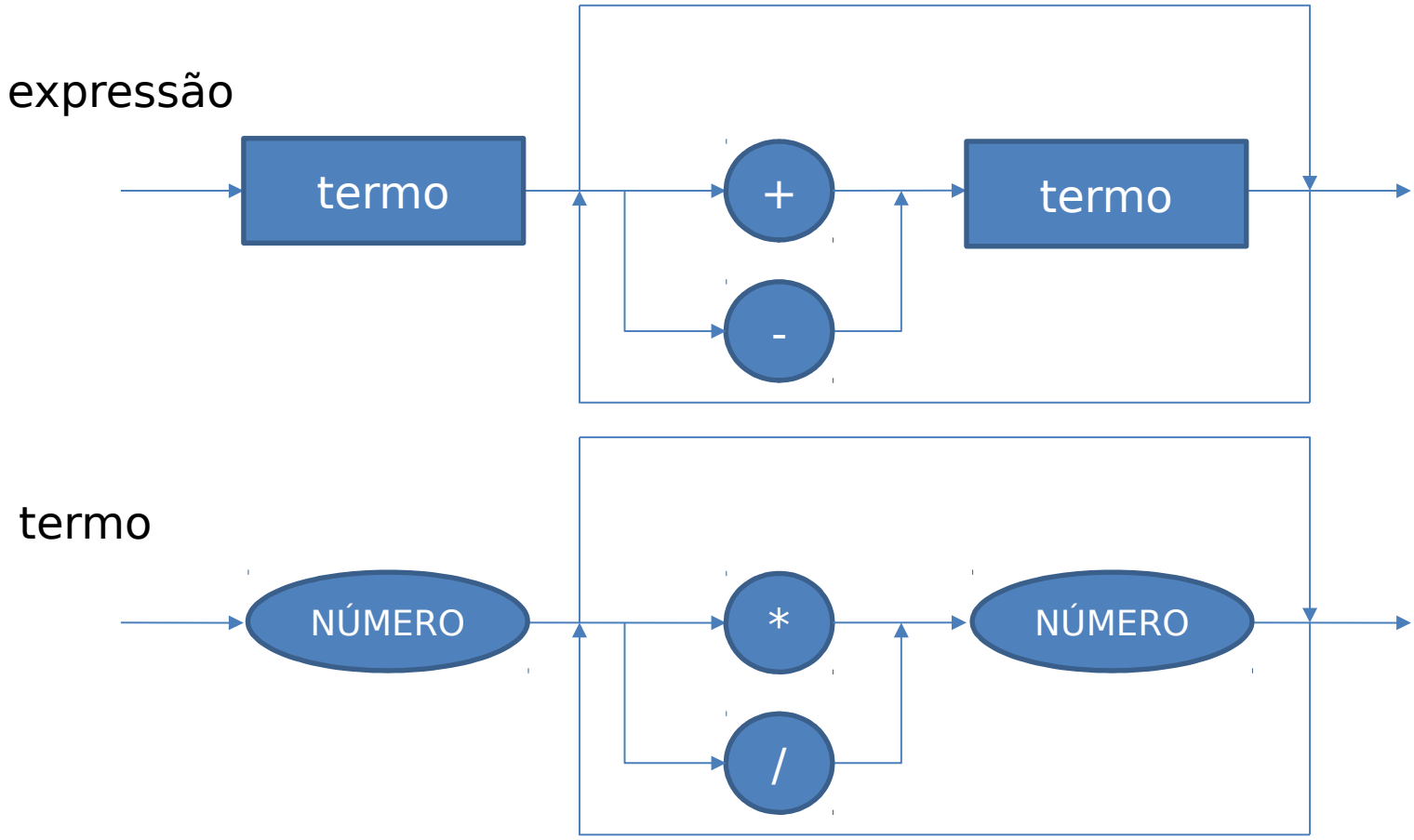


- Contudo esta abordagem não funciona, como visto no Roteiro 1.

Precedência de Operadores

- Qual operador tem maior precedência?
- E depois?
- Só existem operadores de números?

Sugestão: 2 blocos



Discussão: Isso funciona? SOMA e SUB ainda funcionam?

Comentários

- Formato:

  /* comentário */

- Para pensar um pouco:
  - Como e aonde isso será tratado?
  - O que acontece se aparecer: /*/*?
  - Precisa colocar na EBNF?

Atividade: Roteiro 2

- Roteiro Impresso ou PDF no Blackboard.

Próxima Aula

- Pumping Lemma
- Autômatos Finitos Não Determinísticos

- Referências:
    - Marcus et al Pag. 156 e 253