Insper

Lógica da Computação

Aula 09

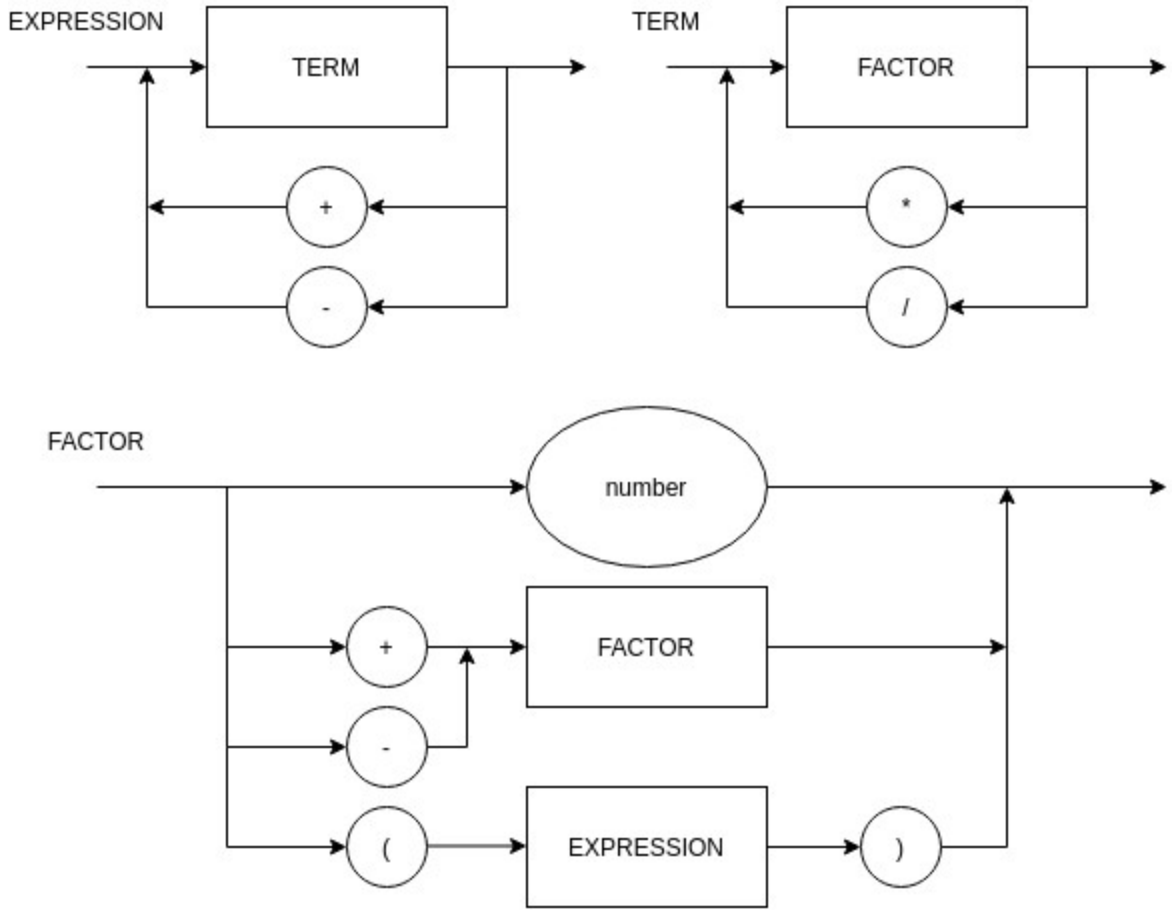
Raul Ikeda

1º semestre de 2020

Esta Aula

- Abstract Syntax Tree - **AST**

Situação Atual



EBNF e Gramática

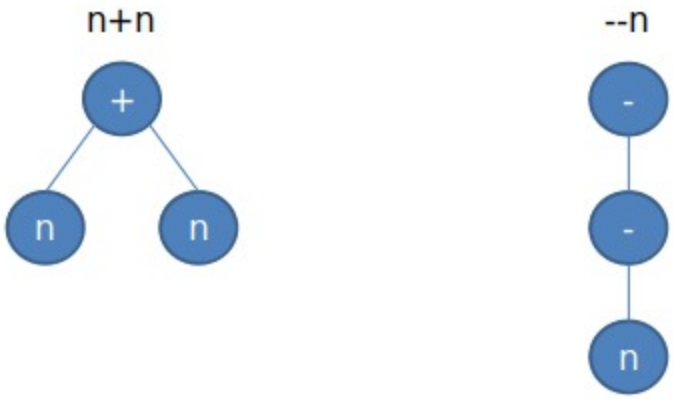
```
EXPRESSION = TERM, { ("+" | "-"), TERM } ;
TERM = FACTOR, { ("*" | "/"), FACTOR } ;
FACTOR = ("+" | "-") FACTOR | "(" EXPRESSION ")" | number ;
```

$$G = (\{E, T, F, +, -, *, /, (,), n\}, \{+, -, *, /, (,), n\}, P, E)$$

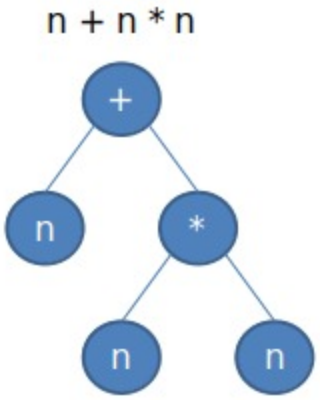
$$P = \begin{cases} E \to T \\ E \to E + T \\ E \to E - T \\ T \to F \\ T \to T * F \\ T \to T / F \\ F \to -F \\ F \to +F \\ F \to (E) \\ F \to n \end{cases}$$

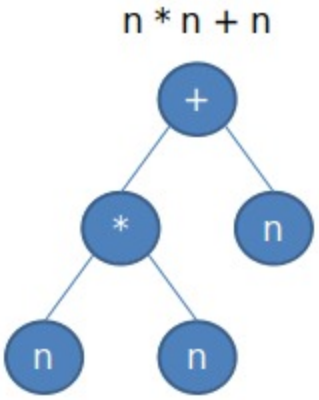
AST - Abstract Syntax Tree

- Assim como árvore de derivação é uma forma de representar uma cadeia em função de uma gramática. A AST representa a ordem com que o diagrama sintático é percorrido.
- Ideia: **alguns** símbolos terminais da nossa gramática vão virar **nós** de uma **árvore**. Dependendo do símbolo, este poderá possuir 0 ou mais filhos.
- Exemplos:



AST – A ordem importa

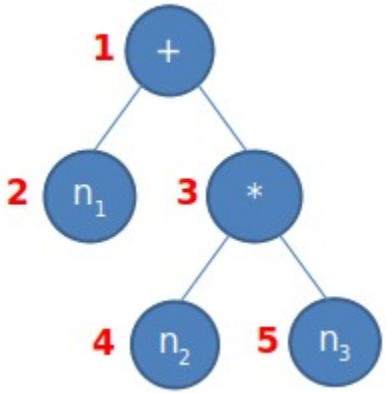




Para montar corretamente, é preciso realizar o **tracing** do diagrama sintático.

AST – Para que serve?

$$n_1 + n_2 * n_3$$



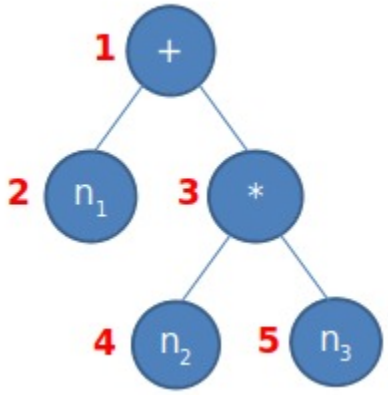
Percorrendo a árvore em profundidade da esquerda para a direita:

+ n_1 (* n_2 n_3) **Notação Polonesa ou Prefix**

Pergunta do milhão: aonde isso será útil mesmo?

AST – Como funciona?

$$n_1 + n_2 * n_3$$

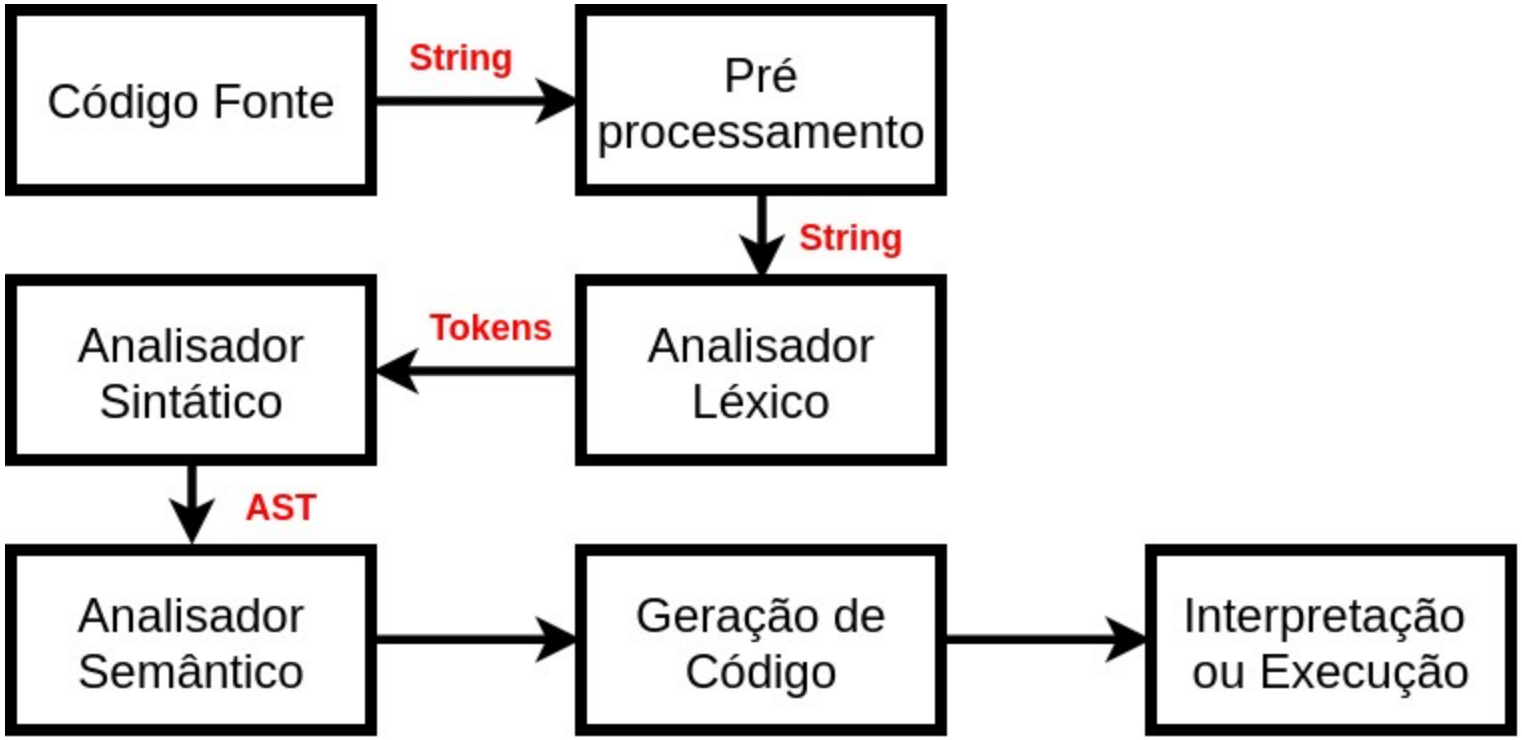


Cada nó pede para que cada filho interprete a sua parte recursivamente.

Exemplo ao lado:

#1: Resolve aí #2
#2: Retorno n_1 para #1
#1: Resolve aí #3
#3: Resolve aí #4
#4: Retorno n_2 para #3
#3: Resolve aí #5
#4: Retorno n_3 para #3
#3: Retorno n_2 * n_3 para #1
#1: Retorno n_1 + n_2 * n_3 para meu **pai**

Estado Atual



Atividade: Roteiro 4

- Roteiro Impresso ou PDF no Blackboard.

Próxima Aula

- Analisadores LL(k) e LR(k)
- Pumping Lemma para CFL
- Hierarquia de Chomsky

Referências:

- Hopcroft et al. Cap. 7
- Sipser Cap 2.3