

Árvore de Análise

Também conhecida como **Parse Tree**

Descrevem naturalmente a estrutura hierárquica das linguagens que a definem

Seja o exemplo a seguir:

```

<atribuição> → <id> = <expr>
<id>          → A | B | C
<expr>       → <id> + <expr>
               | <id> * <expr>
               | (<expr>)
               | <id>
    
```

Por exemplo, a instrução

Será derivada a seguir:

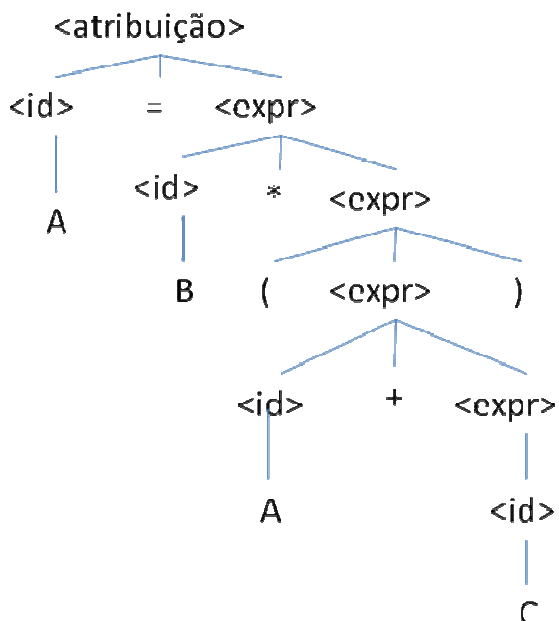
```

A = B * (A + C)
    
```

```

<atribuição> => <id> = <expr>
=> A = <expr>
=> A = <id> * <expr>
=> A = B * <expr>
=> A = B * (<expr>)
=> A = B * (A + <expr>)
=> A = B * (A + <id>)
=> A = B * (A + C)
    
```

A árvore de análise do exemplo anterior demonstra a construção da estrutura feita na derivação anterior



Note que a construção da árvore continua sendo feita se utilizando a derivação à extrema esquerda.

A derivação prossegue até que a forma sentencial não contenha nenhum não-terminal

- Existem outras derivações utilizadas como a extrema direita e ordens combinando extrema esquerda com extrema direita.
- A ordem de derivação não influencia a linguagem final gerada pela gramática
- Após escolher exaustivamente todas as combinações de opções a linguagem inteira será gerada

Observações:

- Todo o vértice interno de uma árvore de análise é dito símbolo não terminal
- Toda folha (vértices sem filhos) é rotulada como símbolo terminal
- Toda a sub-árvore de uma árvore de análise descreve uma instância de abstração na instrução

Propriedades da Árvore de Análise

Para as propriedades das árvores de análise estudaremos 3 propriedades que as compõem:

- ▶ Ambiguidade
- ▶ Precedência de operadores
- ▶ Associatividade de operadores

Ambiguidade

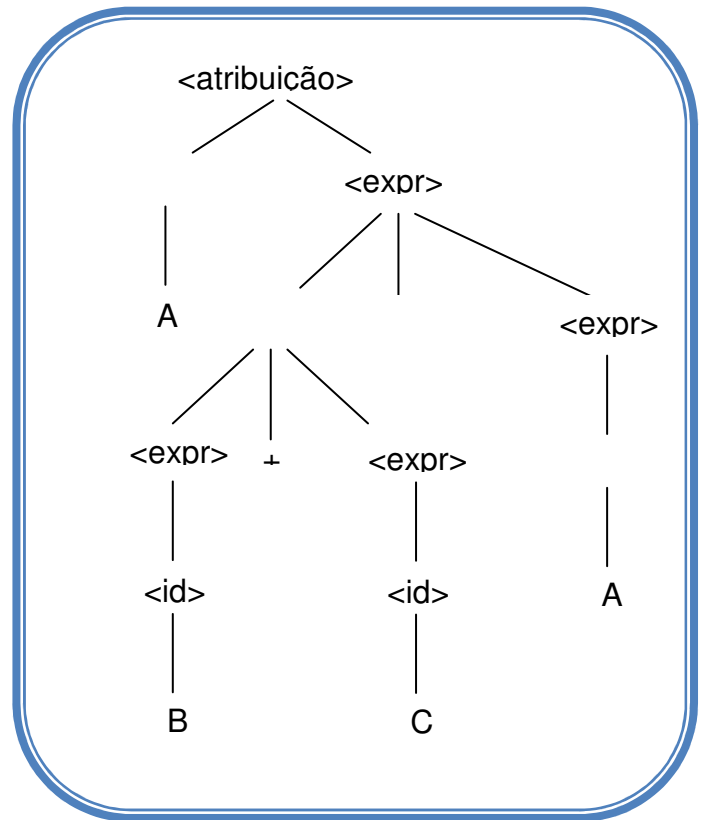
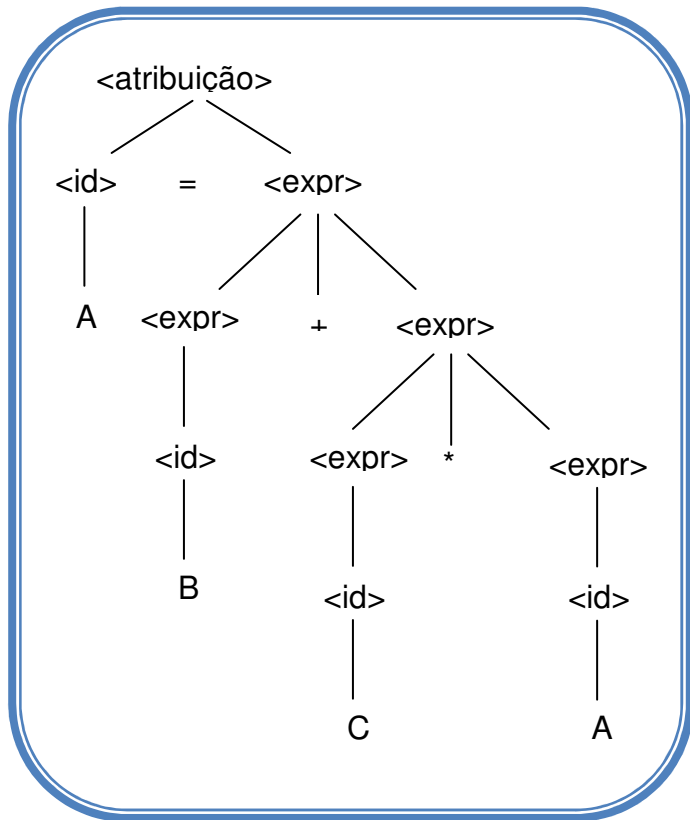
Uma árvore é considerada ambígua quando existirem duas ou mais árvores de análise distintas.

Ex:

```

<atribuição> → <id> = <expr>
<id>          → A | B | C
<expr>       → <expr> + <expr>
              | <expr> * <expr>
              | ( <expr> )
              | <id>
    
```

Como exemplo de gramática ambígua, podemos exemplificar a sentença $A = B + C * A$. Nesta sentença, poderá-se gerar duas árvores de análise distintas, como segue abaixo:



Detalhes:

Esta ambigüidade ocorre porque a gramática não especifica a estrutura a ponto de conseguirmos prever casos como estes demonstrados acima. Visto que a árvore de análise poderá crescer tanto para a esquerda como para a direita.

O compilador enfrenta este problema na geração do código, pois baseiam a semântica desta estrutura em suas formas sintáticas. Quer dizer que o compilador escolhe o que vai fazer examinando a sua árvore de análise, se uma estrutura tiver mais de uma árvore de análise, o significado daquela não poderá ser determinado de forma única.

Precedência de Operadores

Uma gramática pode descrever certa estrutura sintática de tal forma que parte do significado da estrutura possa ser deduzido de sua árvore de análise. Ocorre quando temos operadores em posições distintas de níveis da árvore, mais abaixo, por exemplo, e que deverá ser analisado primeiro. Isto poderá indicar que este tem mais precedência sobre um operador produzido mais acima na árvore.

No exemplo da árvore de análise anterior, a árvore da esquerda o operador de multiplicação é gerado mais baixo na árvore, o que poderia indicar que ele tem precedência sobre o operador de adição na expressão.

Já na segunda árvore, podemos notar que é o oposto da primeira, demonstrando claramente informações de precedência conflitantes.