

1.1.2 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.

- Isto indica que deve ser analisado primeiro
- Pode causar informações de precedência conflitantes em uma mesma gramática
- A solução para isso é remodelar a gramática e acomodar os operadores para que não possam conflitar futuramente

Ex:

```

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

A gramática acima gera a mesma linguagem anterior do exemplo que foi demonstrado. Mas ela indica a ordem de precedência usual de operadores de multiplicação e de adição.

Seja a sentença **A = B + C * A**

```

<atribuição> => <id> = <expr>
=> A = <expr>
=> A = <expr> + <termo>
=> A = <termo> + <termo>
=> A = <fator> + <termo>
=> A = <id> + <termo>
=> A = B + <termo>
=> A = B + <termo> * <fator>
=> A = B + <fator> * <fator>
=> A = B + <id> * <fator>
=> A = B + C * <fator>
=> A = B + C * <id>
=> A = B + C * A
    
```

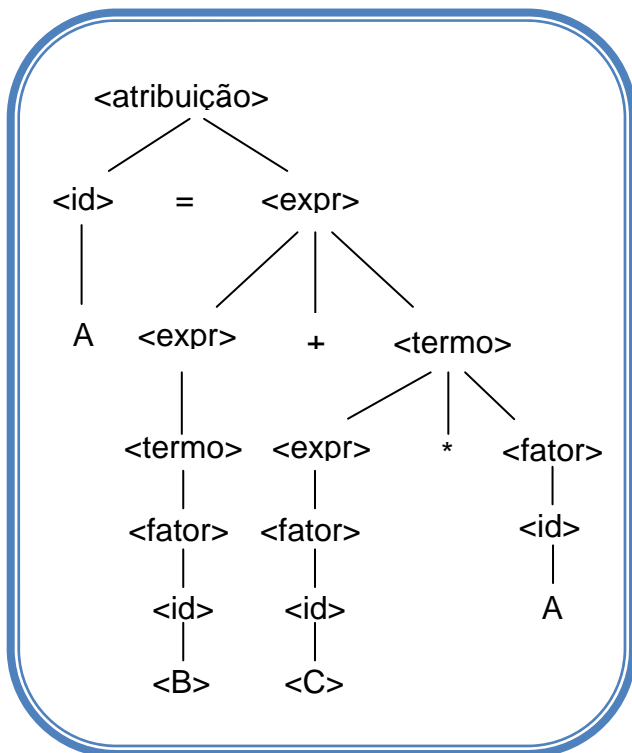
A conexão entre árvores de análise e derivações é muito estreita: cada uma pode ser facilmente construída a partir da outra. Toda a derivação com uma gramática não-ambígua possui uma árvore de análise única.

Apesar de possuir uma única árvore de análise, o exemplo anterior também pode ser representado por outro tipo de derivação (extrema direita neste caso), seja o exemplo:

```

<atribuição> => <id> = <expr>
=> <id> = <expr> + <termo>
=> <id> = <termo> + <termo> + <fator>
=> <id> = <expr> + <termo> * <id>
=> <id> = <expr> + <termo> * A
=> <id> = <expr> + <fator> * A
=> <id> = <expr> + <id> * A
=> <id> = <expr> + C * A
=> <id> = <termo> + C * A
=> <id> = <fator> + C * A
=> <id> = <id> + C * A
=> <id> = B + C * A
=> A = B + C * A
    
```

A única árvore de análise possível para as duas expressões acima é demonstrada a seguir:

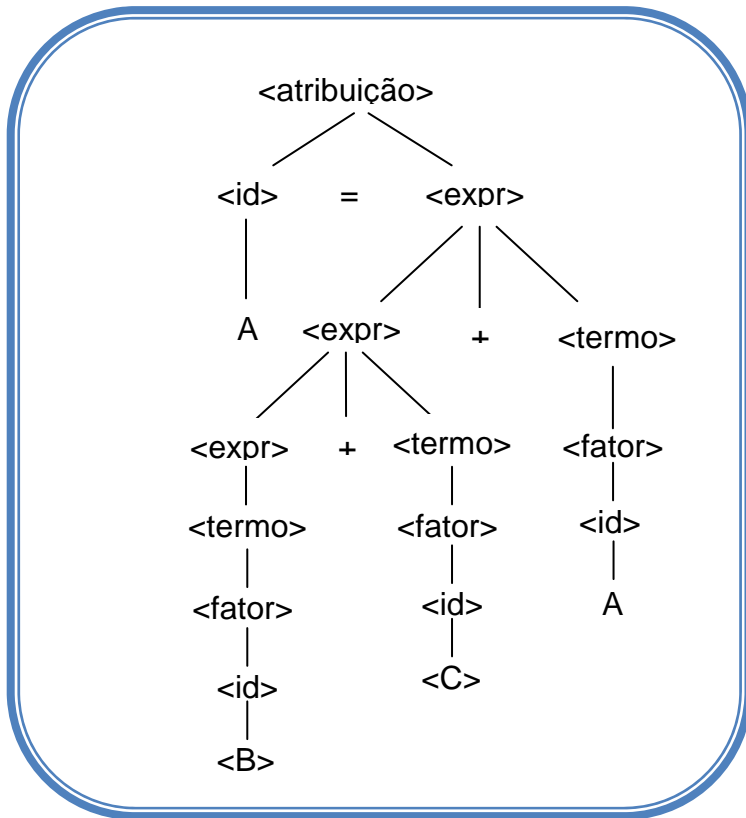


1.2 Associatividade de Operadores

A ordem utilizada para operadores iguais em lugares diferentes na expressão é representada com o mesmo efeito na árvore de análise? Nem sempre, pois em alguns casos será necessário que os operadores compliquem a solução de dadas expressões. A expressão $A = B + C + A$, por exemplo necessita que seja determinado através de sua árvore de análise qual dos operadores será resolvido primeiro.

Neste caso, pode-se somar $C + A$ e depois com B ou $B + C$ e depois com A . Desta forma, ao criarmos a árvore de análise poderemos ter problemas com qual das operações será resolvida primeiro.

O nome desta tarefa é o que resulta em associatividade de operadores, e ela pode ser associativa à esquerda, que é o mais típico, ou associativa à direita. O exemplo a seguir ilustra melhor esta propriedade:



2 BNF Estendida ou EBNF

Existem extensões da gramática de Backus-aur devido às suas inconveniências de leitura em alguns casos. Todas as versões de extensões desenvolvidas são consideradas Extended BNF ou simplesmente EBNF.

Estas extensões surgiram apenas para melhorar a legibilidade e capacidade de escrita da BNF e permitem que as operações possam ser construídas de maneiras mais dinâmicas.

São 3 as extensões conhecidas e aplicadas a BNF:

Regra1: A primeira delas permite o uso opcional de um comando através da inserção de colchetes, que não exige que o comando seja necessariamente utilizado na expressão em questão. Por exemplo, a expressão:

<seleção> → if (<expressão> <instrução> [else <instrução>];

demonstra que sem o uso dos colchetes seria necessário o uso de duas regras.

Regra 2: A segunda regra é o uso de chaves no lado direito da expressão, indicando que a parte contida nas chaves poderá ser repetida infinitas vezes ou omitida completamente. Seja o exemplo a seguir:

<lista_ident> → <identificadores> {, <identificador>}

Regra 3: a terceira regra lida com expressões de múltipla escolha, são colocadas as opções dentro de parênteses separadas pelo operador | (pipe) – que significa o operador **OU**. Seja o exemplo a seguir:

`<inst_for> → for <var> := <expr> (to | downto) <expr> do <inst>`

Neste caso, seriam necessárias duas regras na BNF para descrever a estrutura acima.

Exercícios

1. Baseado nas gramáticas abaixo comprove que é possível gerar as seguintes sequências de derivações obedecendo as regras propostas.

- a. Para a sentença: $A = B * A * A$

```

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

- i. Gramática:

- b. Para as sentenças:

- i. $A = C * B + C$

- ii. $C = D * A - A + B$

```

<atribuição> → <id> = <expr>
<id>          → A | B | C | D
<expr>       → <expr> / <expr>
              | <expr> * <expr>
              | <expr> + <expr>
              | <expr> - <expr>
              | <id>
    
```

- iii. Gramática :

2. Gere a árvore de análise para as sentenças geradas de acordo com as regras a seguir:

- a. Sentença: $A = B + C + A$

```

<atribuição> → <id> = <expr>
<id>          → A | B | C | D
<expr>       → <expr> / <expr>
              | <expr> * <expr>
              | <expr> + <expr>
              | <expr> - <expr>
              | <id>
    
```

- i. Gramática:

- b. Com a gramática anterior, a sentença: $B = A + A + A + D$

- c. Com a gramática anterior, a sentença: $D = A / B * C$