Curso: Engenharia de Computação

Linguagens Formais e Compiladores

Prof. Clayton J A Silva, MSc
clayton.silva@professores.ibmec.edu.br

ibmec

Análise Sintática

Árvore sintática

- Normalmente construída como uma estrutura de dados padrão baseada em ponteiros – alocada dinamicamente à medida que a análise sintática ocorre
- Indicada por uma variável que aponta para um nó-raiz
- Cada nó é um **registro** cujos campos representam a **informação coletada pelo analisador sintático** e pelo analisador semântico
- Cada nó pode requerer o armazenamento de diferentes atributos

Árvore
Definição do registro de nó da árvore

```
// Criação de um tipo nó e um ponteiro de nó
struct nodetype {
    int info;
    struct nodetype *left;
    struct nodetype *right;
    struct nodetype *father;
};
typedef struct nodetype *NODEPTR;
```

- Os campos *left*, *right* e *father* do nó apontam para o filho esquerdo, filho direito e nó pai, respectivamente
- O campo *info* contém a informação do elemento não terminal ou terminal da análise sintática
- São informações:

- A árvores de análise mostram a **derivação** da regra de atribuição
- O **nó-raiz** contém o símbolo **não terminal** de mais alto nível do qual se obtém as produções seguintes
- Os **ramos** são definidos pelos **vértices internos** ou **nós**, rotulados pelos **símbolos não terminais**
- As **folhas** de cada ramo sempre serão rotuladas pelos **símbolos terminais**

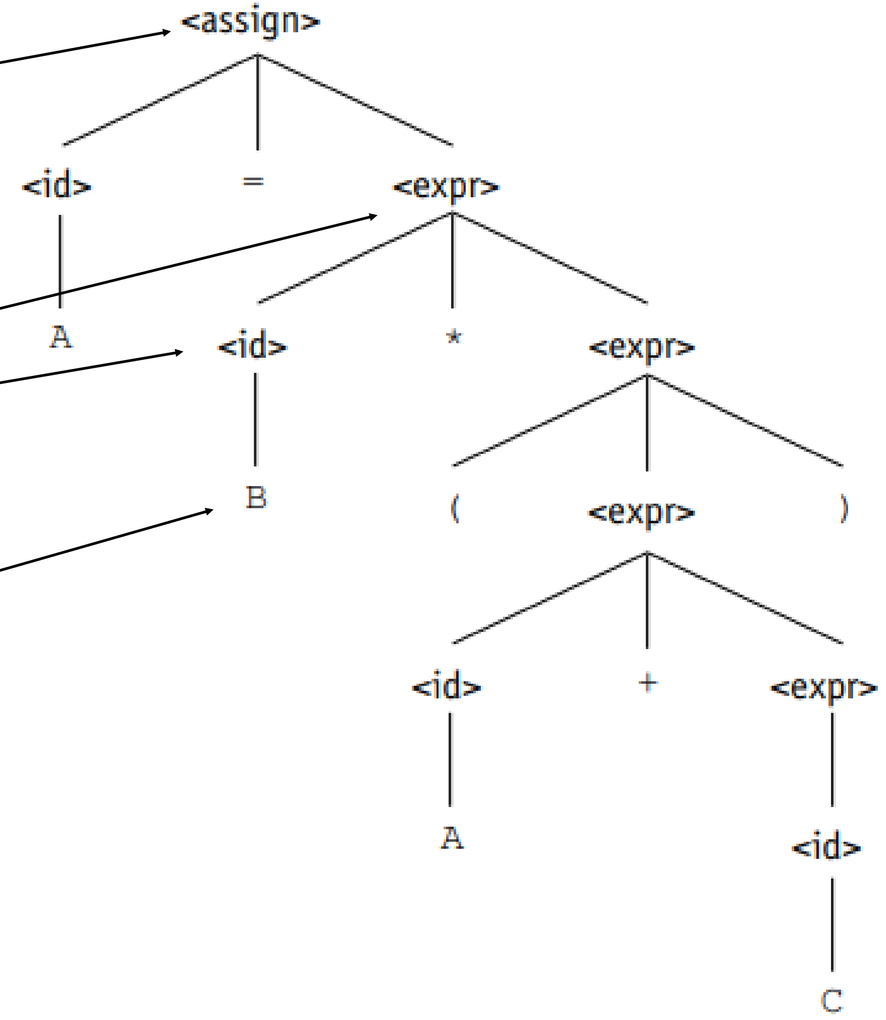


FIGURA 3.1
Uma árvore de análise sintática para a sentença simples `A = B * (A + C).`

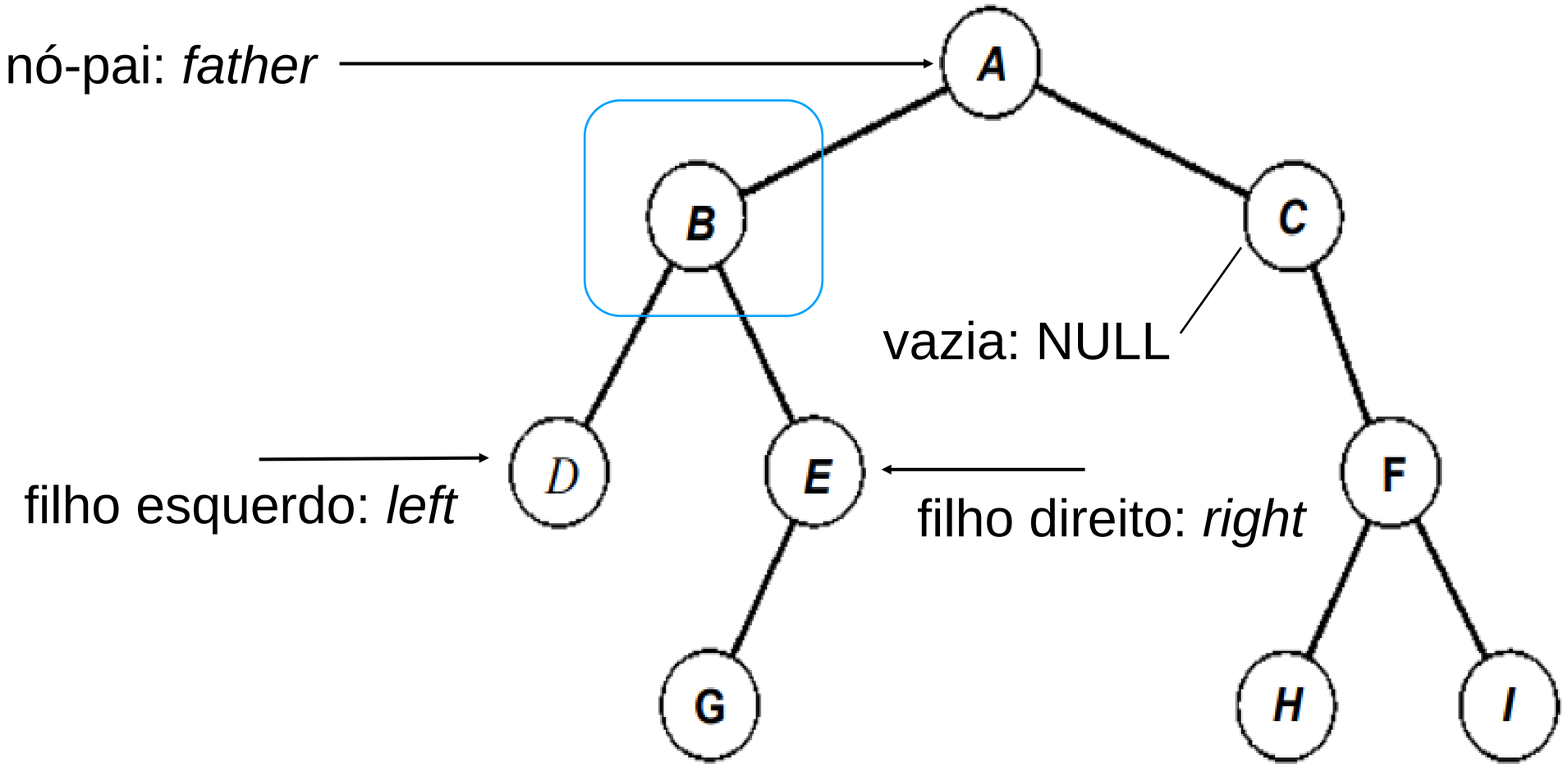
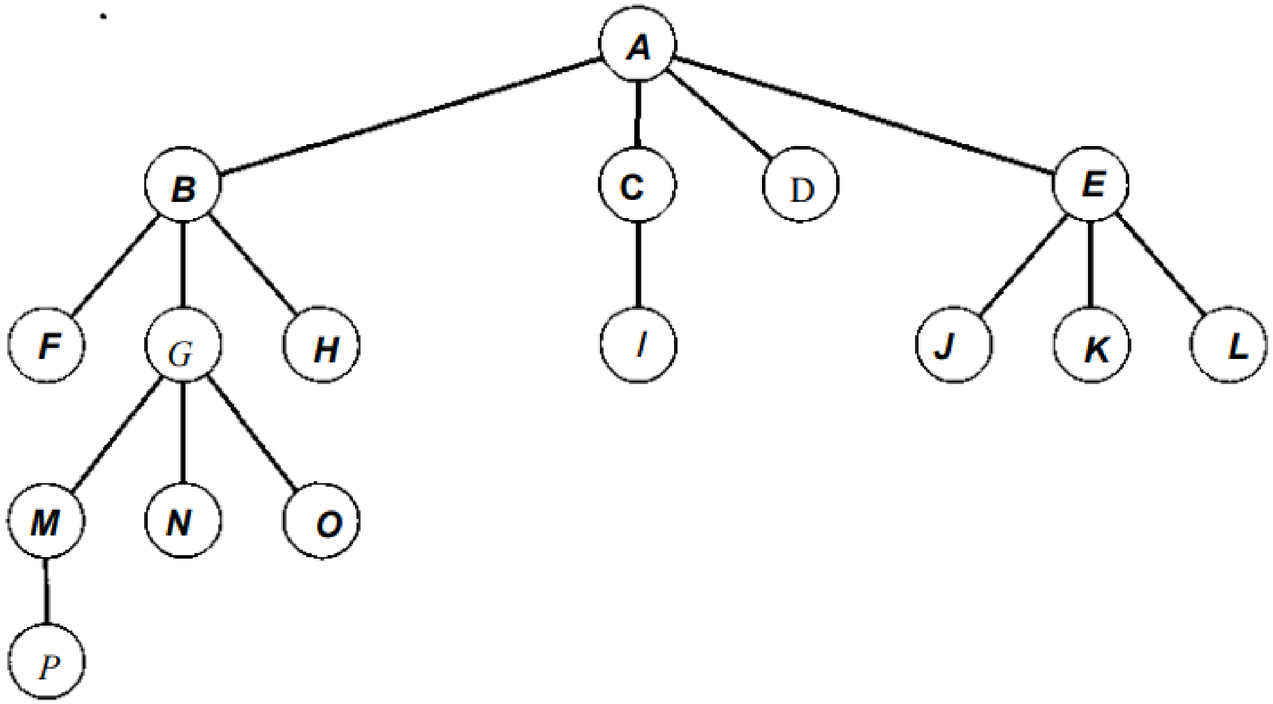
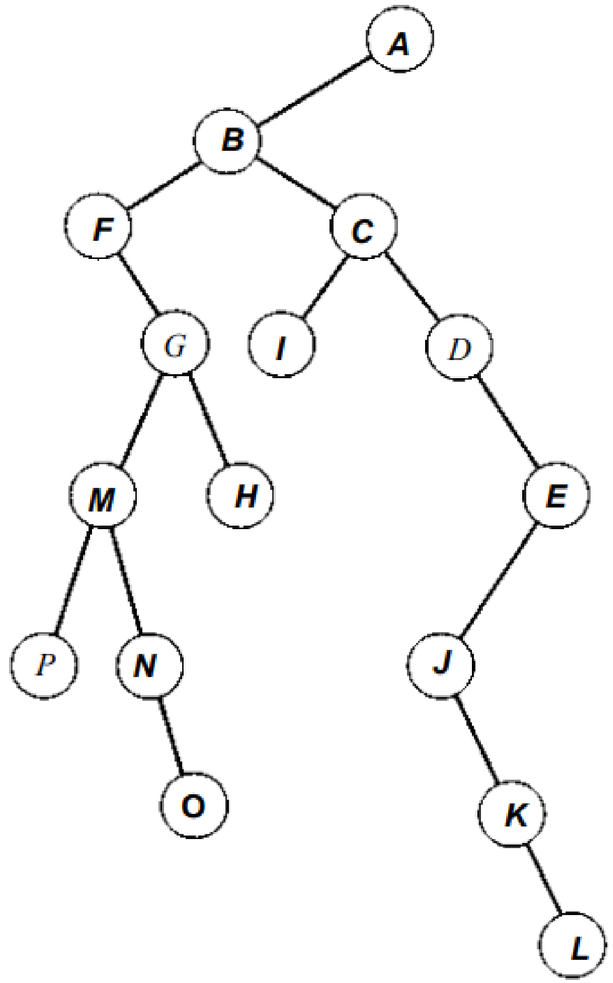


Figura 5.1.1 Uma árvore binaria.





- Produzindo uma árvore binária a partir de uma árvore não-binária
- Derivação à esquerda

Árvore
Alocação dinâmica do endereço de nó

```
NODEPTR getnode(){
    NODEPTR p;
    p = (NODEPTR) malloc(sizeof(struct nodetype));
    return(p);
}
```

- **Função *getnode*()**
  - Será alocado um endereço disponível ao ponteiro p
  - *sizeof* é aplicado a um tipo de estrutura e retorna o número de bytes necessário para a estrutura inteira do nó.
  - A função retorna um ponteiro adequado para o nó de interesse

Árvore
Criação do nó único

- **Função *maketree*(x)**
  - Define um ponteiro de nó
  - Atribui um endereço
  - Cria um nó ÚNICO da árvore, com o campo de informação *x*
  - Retorna um ponteiro para o nó
  - Observe que, no exemplo, o campo de informação é do tipo char

```
NODEPTR maketree(char x){
      NODEPTR p;
      p = getnode();
      p->info = x;
      p->left = NULL;
      p->right = NULL;
      p->father = NULL;
      return(p);
}
```

Árvore
Criação do nó com conteúdo filho à esq

```
void setleft(NODEPTR p, char x){
    if (p == NULL)
        printf("insercao vazia\n");
    else if (p->left != NULL)
            printf("insercao incorreta\n");
        else
            p->left = maketree(x);
}
```

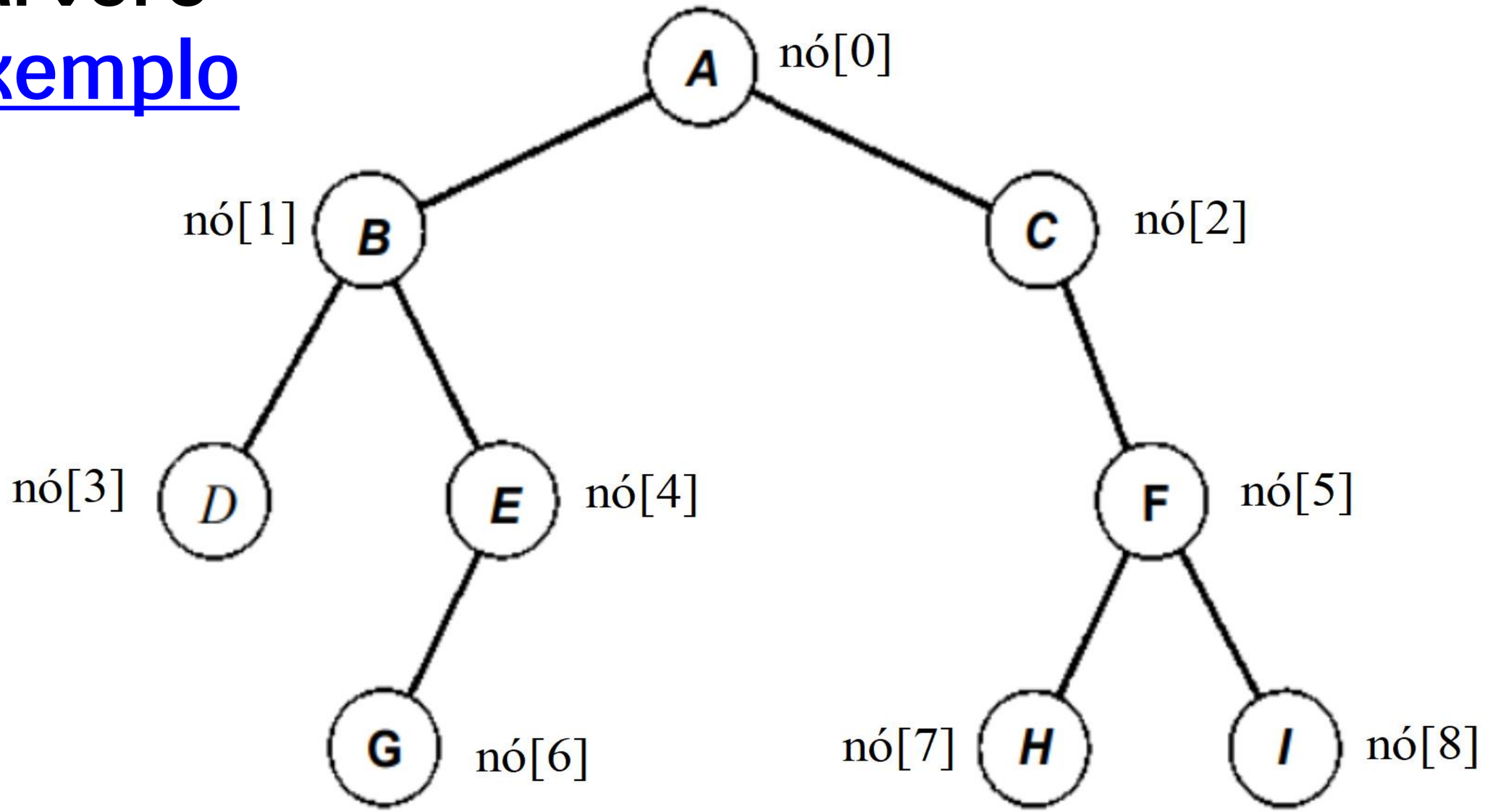
- **Função *setleft*(NODEPTR p, char x)**
  - Para um ponteiro de nó, atribui um endereço uma informação ao nó à esquerda

Árvore sintática

- A rotina setright(p,x) para criar um filho direito de node(p) com o conteúdo x é semelhante
- Nem sempre é necessário usar os campos *father*, *left* e *right*.
- Se uma árvore for sempre percorrida de cima para baixo (da raiz para as folhas), a operação *father* nunca será usada; nesse caso, será desnecessário um campo *father*.

Árvore

Exemplo



Análise Semântica

Considerações gerais

- A análise semântica ocorre antes da compilação, razão pela qual é chamada também de **análise semântica estática**
- Requer a construção de uma **tabela de símbolos** para acompanhar o significado dos nomes estabelecidos nas declarações e efetuar inferência e verificação de tipos
- Um método comum de implementar é identificar **atributos** ou propriedades de **entidades da linguagem**
- As entidades precisam ser computadas por **equações de atributos** ou **regras semânticas**
- O conjunto de atributos e equações é denominado **gramática de atributos**
- **Semântica dirigida pela sintaxe**: o conteúdo semântico deve ser fortemente relacionado com a sintaxe

ATRIBUTOS

- Qualquer propriedade de uma construção de uma linguagem de programação.
- Exemplos:
  - Tipo de dados de uma variável
  - Valor de uma expressão
  - Localização de uma variável na memória
  - Quantidade de dígitos significativos em um número
  - ...

ATRIBUTOS

- Tipicamente as gramáticas de atributos são escritas na forma de tabelas

Regra gramatical	**Regras semânticas**
Regra 1	Equações de atributos associadas
.	
.	
.	
Regra n	Equações de atributos associadas

Exemplos: regras semânticas

- Seja a gramática

  número → *número dígito* | *dígito*

  dígito → *[0-9]*

- A regra sintática *número* → *dígito* relaciona-se

  com a regra semântica *número.valor = dígito.valor*

- A regra sintática *número* → *número dígito* relaciona-se

  com a regra semântica *número.valor = número.valor * 10 + dígito.valor*

Exemplos: regras semânticas

Regra gramatical	Regras semânticas
$número_1 \rightarrow$ $número_2$ *dígito*	$número_1.val =$ $número_2.val * 10 + dígito.val$
número → *dígito*	*número.val* = *dígito.val*
dígito → 0	*dígito.val* = 0
dígito → 1	*dígito.val* = 1
dígito → 2	*dígito.val* = 2
dígito → 3	*dígito.val* = 3
dígito → 4	*dígito.val* = 4
dígito → 5	*dígito.val* = 5
dígito → 6	*dígito.val* = 6
dígito → 7	*dígito.val* = 7
dígito → 8	*dígito.val* = 8
dígito → 9	*dígito.val* = 9

Exemplos: regras semânticas

- Seja a gramática

  exp → *exp* * *termo* | *exp* – *termo* | *termo*

  termo → *termo* * *fator* | *fator*

  fator → (*exp*) | *número*

Regra gramatical	Regras semânticas
$exp_1 \rightarrow exp_2$ + *termo*	$exp_1.val = exp_2.val$ + *termo.val*
$exp_1 \rightarrow exp_2$ – *termo*	$exp_1.val = exp_2.val$ – *termo.val*
exp → *termo*	*exp.val* = *termo.val*
$termo_1 \rightarrow termo_2$ * *fator*	$termo_1.val = termo_2.val$ * *fator.val*
termo → *fator*	*termo.val* = *fator.val*
fator → (*exp*)	*fator.val* = *exp.val*
fator → **número**	*fator.val* = **número**.*val*

IBMEC.BR

/IBMEC

IBMEC

@IBMEC_OFICIAL

@IBMEC

Ibmec