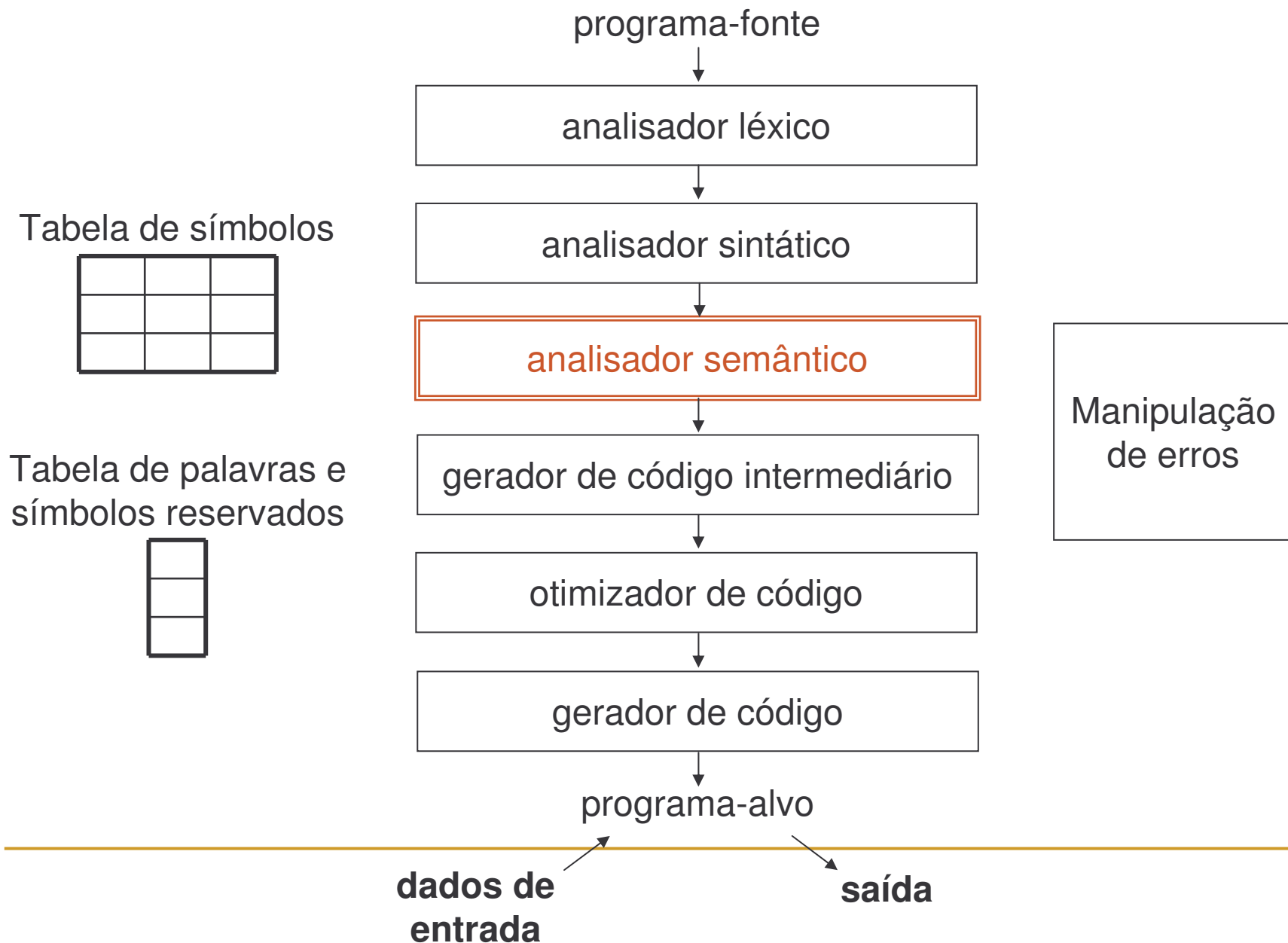


Análise Semântica

Profa. Valéria D. Feltrim
DIN – UEM

Estrutura geral de um compilador



Análise semântica

- Função: verificação do uso adequado
 - Análise contextual: declarações prévias de variáveis, procedimentos, etc.
 - Checagem de tipos
 - Outras coisas que vão além do domínio da sintaxe
 - Sensitividade ao contexto
- Tipos de análise semântica
 - Estática → em tempo de compilação: linguagens tipadas, que exigem declarações
 - C, Pascal, etc.
 - Dinâmica → em tempo de execução: linguagens em que as variáveis são determinadas pelo contexto de uso
 - LISP, PROLOG

Análise semântica

- Devido às variações de especificação semântica das linguagens de programação, a análise semântica:
 - ❑ Não é tão bem formalizada
 - ❑ Não existe um método ou modelo padrão de representação do conhecimento
 - ❑ Não existe um mapeamento claro da representação para o algoritmo correspondente
 - A análise é artesanal e dependente da linguagem de programação
-

Análise semântica

- **Semântica dirigida pela sintaxe**

- Conteúdo semântico fortemente relacionado à sintaxe do programa
- Maioria das linguagens de programação modernas

- Muitas vezes, a semântica de uma linguagem de programação não é claramente especificada

- O projetista do compilador tem que analisar e extrair a semântica
-

Análise semântica

- Em geral, a **gramática de atributos** de uma GLC especifica:
 - ❑ Comportamento semântico das operações
 - ❑ Checagem de tipos
 - ❑ Manipulação de erros
 - ❑ Tradução do programa
 - A gramática de atributos diz quais ações serão realizadas e quando
 - ❑ Pode ser uma ação de verificação semântica ou de tradução
-

Tabela de símbolos

- **Tabela de símbolos**: estrutura essencial para a análise semântica
- Permite saber durante a compilação de um programa o **tipo** e **endereço** de seus elementos, **escopo** destes, número e tipo dos **parâmetros** de um procedimento, etc.
 - Cada categoria de token tem atributos/informações diferentes associadas

Cadeia	Token	Categoria	Tipo	Endereço	...
i	id	var	integer	1	...
fat	id	proc	-	-	...
...					

Tabela de símbolos

- Exemplo de atributos para identificador de variável
 - Nome da variável, tipo (inteira, real, etc.), escopo (global, local, etc.), endereço na memória, etc.
 - Para vetores, ainda seriam necessários atributos como o tamanho do vetor, valor e tipo de seus limites, etc.
 - Exemplo de atributos para identificador de procedimento
 - Nome do procedimento, lista de argumentos (número, ordem e tipo), escopo, etc.
-

Tabela de símbolos

- Principais operações na tabela de símbolos
 - **Inserir**: armazena na TS informações fornecidas pelas declarações no programa
 - **Busca**: recupera da tabela informações de um elemento declarado no programa quando esse elemento é utilizado
 - **Remover**: remove (ou torna inacessível) da tabela informações sobre um elemento declarado que não se mostra mais necessário no programa (p.e., fora de escopo)
-

Tabela de símbolos

- A tabela é acessada pelo compilador sempre que um elemento é mencionado no programa
 - Verificar ou incluir sua declaração
 - Verificar seu tipo, seu escopo ou alguma outra informação
 - Remover um elemento quando este não se faz mais necessário ao programa
-

Tabela de símbolos

- Questões de projeto

- **Estrutura da tabela de símbolos**: determinada pela eficiência das operações de inserir, verificar e remover
 - Várias possibilidades
 - Implementação
 - Estática ou Dinâmica (Melhor opção)
 - Estrutura
 - Listas lineares
 - Árvores de busca (por exemplo, B e AVL)
 - *Hashing*
 - Opção mais eficiente
 - O identificador é a chave e a função *hash* indica sua posição na tabela de símbolos
 - Necessidade de tratamento de colisões
-

Tabela de símbolos

- Questões de projeto

- **Tamanho da tabela**: tipicamente, de algumas centenas a mil campos
 - Dependente da forma de implementação
 - Na implementação dinâmica, não é necessário se preocupar com isso
 - Uma **única tabela** para todas as declarações ou **várias tabelas**, sendo uma para cada tipo de declaração (constantes, variáveis, procedimentos e funções)
 - Diferentes declarações têm diferentes informações/atributos
 - por exemplo, variáveis não têm número de argumentos, enquanto procedimentos têm
-

Tabela de símbolos

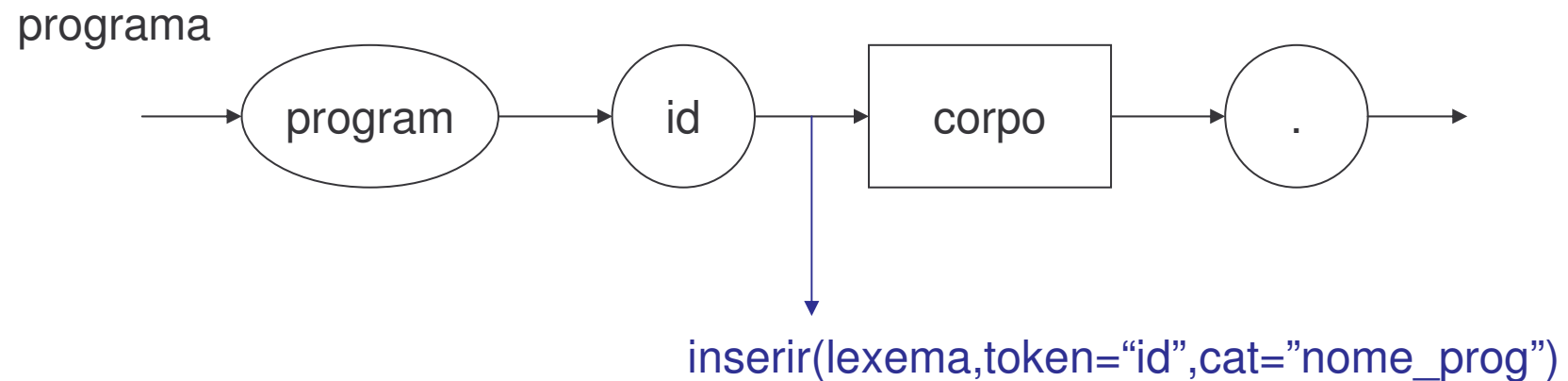
- ❑ **Representação de escopo** de identificadores do programa
 - **Várias tabelas** ou uma **única tabela** com a identificação do escopo para cada identificador
 - Tratamento de escopo
 - ❑ Inserção de identificadores de mesmo nome, mas em níveis diferentes
 - ❑ Remoção/bloqueio de identificadores cujos escopos deixaram de existir
-

Tabela de símbolos

- Exemplos de ações realizadas com acesso à TS
 - Inserção de elementos na tabela
 - Verificar se o elemento já não consta na tabela
 - Busca de informação na tabela
 - Realizada antes da inserção
 - Busca de informações para análise semântica
 - Remoção de elementos da tabela
 - Tornar inacessíveis dados que não são mais necessários
 - Por ex., após o escopo ter terminado
 - Linguagens que permitem estruturação em blocos
-

Tabela de símbolos

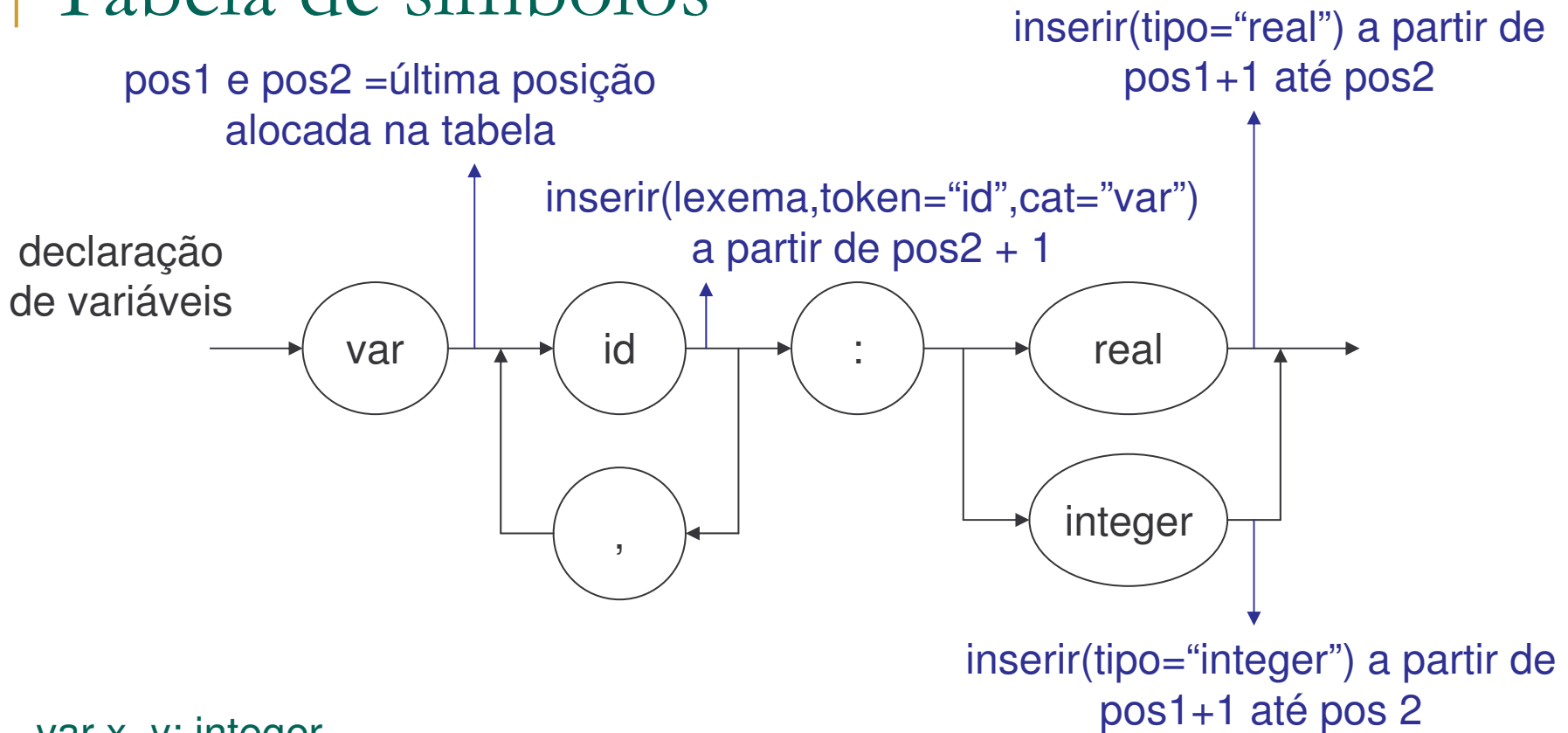
- Inserção de elementos na tabela
 - Declaração



program meu_prog ...

Cadeia	Token	Categoria	Tipo	Endereço	...
meu_prog	id	nome_prog	-	-	...

Tabela de símbolos



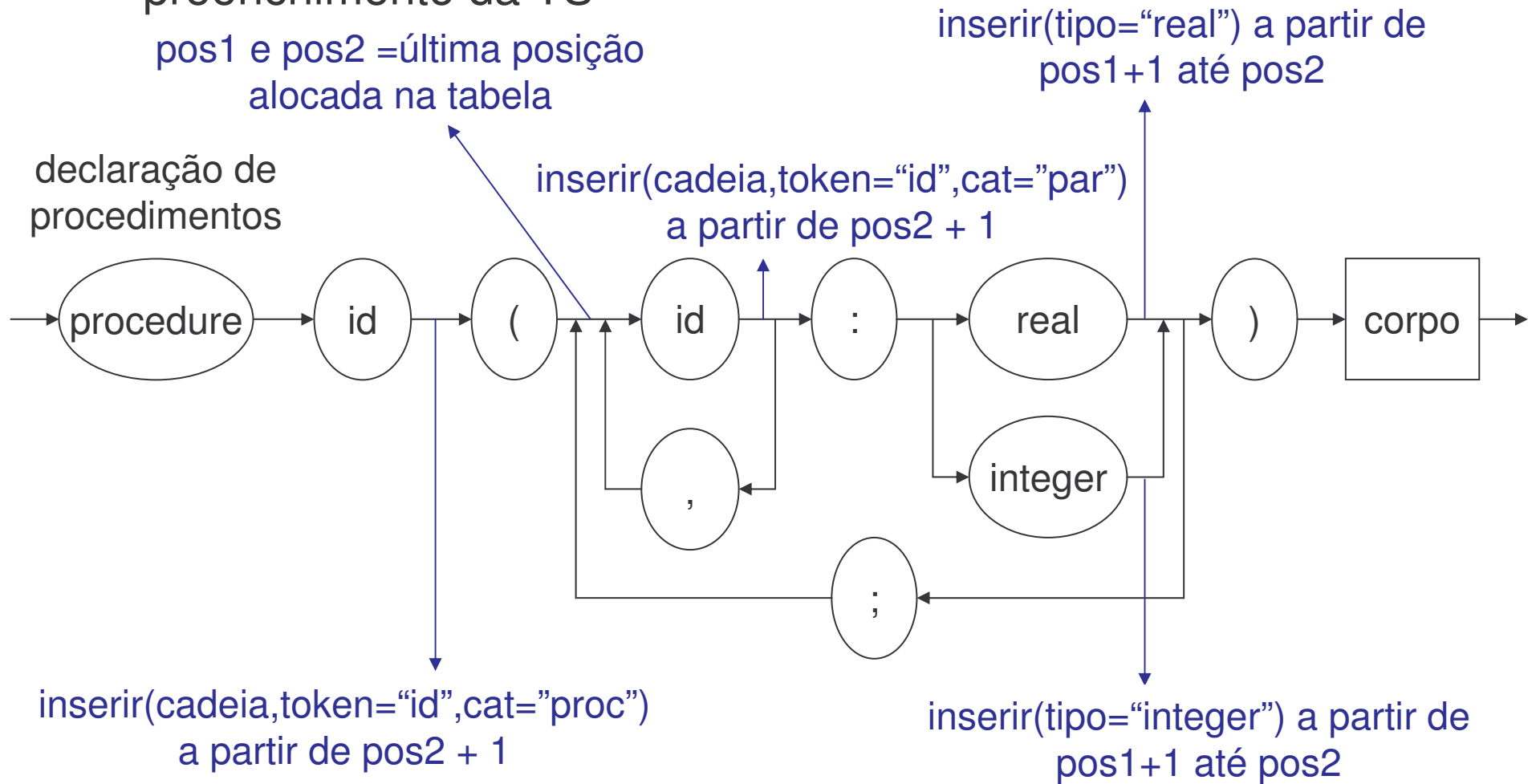
var x, y: integer

Cadeia	Token	Categoria	Tipo	Endereço	...
meu_prog	id	nome_prog	-	-	...
x	id	var	integer		...
y	id	var	integer		...

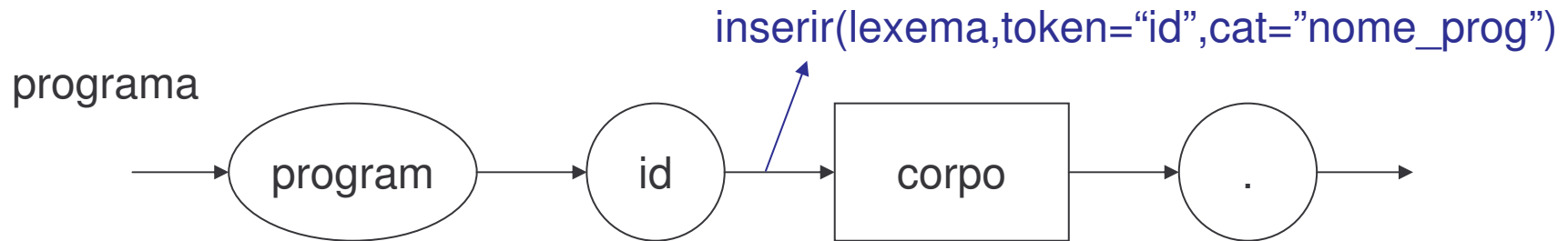
Tabela de símbolos

- Exercício: inclua as funções adequadas para o preenchimento da TS

pos1 e pos2 = última posição
alocada na tabela



Exemplo de procedimento



procedimento programa(Seg)

Início

se (token=t_program) então prox_token(lexema,token)

senão ERRO(Seg+{id});

se (token=id) então

 inserir(lexema,"id","nome_prog")

 prox_token(lexema,token)

senão ERRO(Seg+P(corpo));

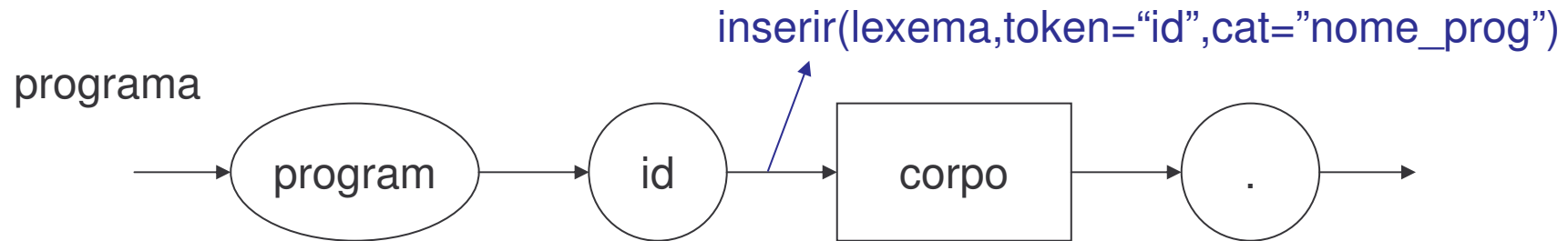
corpo(Seg+{.});

se (token=t_ponto) então prox_token(lexema,token)

senão ERRO(Seg);

fim

Exemplo de procedimento



procedimento programa(Seg)

Início

se (token=t_program) então prox_tok

senão ERRO(Seg+{id});

se (token=id) então

inserir(lexema,"id","nome_pr

prox_token(lexema,token)

senão ERRO(Seg+P(corpo));

corpo(Seg+{.});

se (token=t_ponto) então prox_token(lexema,token)

senão ERRO(Seg);

fim

E quando a análise
sintática é **ascendente**?
Em qual momento
serão realizadas as
ações?

Tratamento de escopo

- A maioria das linguagens de programação implementa escopo estático e, para resolver conflito de nomes, aplica a regra do contexto envolvente mais próximo (escopo ancestral mais próximo)
 - Escopo estático = escopo léxico
 - São criados por subprogramas ou blocos
 - Para vincular uma referência a uma variável, deve se encontrar a declaração dessa variável no escopo atual ou nos seus escopos ancestrais
 - Variáveis com o mesmo nome, mas em escopos diferentes, escondem as variáveis definidas no pai estático
-

Tratamento de escopo

- Como diferenciar variáveis globais de locais na TS
 - Tratamento de variáveis de mesmo nome, mas de escopos diferentes

```
program meu_prog
var x, y: integer
  procedure meu_proc(x: integer)
    var y: real
    begin
      read(y);
      x:=x+y
    end;
begin
  read(y);
  x:=x*y
end.
```

← *y: integer está oculto neste ponto*

Tabela de símbolos

- Possibilidades para tratamento de escopos

- Inclusão de um campo a mais na tabela de símbolos indicando o nível da variável no programa
 - Controle do nível durante a compilação do programa
 - Quando se chama um procedimento (ou função), faz-se $nível := nível + 1$
 - Quando se sai de um procedimento (ou função), faz-se $nível := nível - 1$
 - Busca do fim para início da TS a fim de encontrar a declaração mais recente
 - Associação das variáveis locais a um procedimento (ou função) à entrada relativa ao procedimento (ou função) por meio, por exemplo, de uma lista encadeada
 - Atenção: para a checagem de tipos, deve-se saber quantos são e quais são os parâmetros de um procedimento (ou função) na tabela de símbolos
 - Tabelas diferentes para diferentes escopos
-

Tratamento de escopo

- “Árvore de símbolos” = uma tabela para cada escopo

```
int pot (int base, int exp){
    if(!exp)
        return 1;
    return base * pot(base, exp-1);
}

int main (int argc, void **argv){
    int a, b;
    a = 2;

    b = pot(5, a);

    return b;
}
```

-----ÁRVORE DE SÍMBOLOS-----

```
Escopo = global
    Símbolo: main
        Tipo: INT
        Flags: FUNÇÃO
    Símbolo: pot
        Tipo: INT
        Flags: FUNÇÃO
```

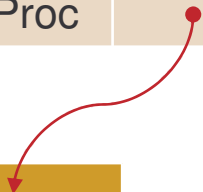
```
Escopo = pot
    Símbolo: exp
        Tipo: INT
        Flags: PARÂMETRO
    Símbolo: base
        Tipo: INT
        Flags: PARÂMETRO
```

```
Escopo = main
    Símbolo: a
        Tipo: INT
        Flags: VARIÁVEL
    Símbolo: b
        Tipo: INT
        Flags: VARIÁVEL
    Símbolo: argc
        Tipo: INT
        Flags: PARÂMETRO
    Símbolo: argv
        Tipo: VOID
        Flags: PARÂMETRO PONTEIRO (Prof = 2)
```

Tratamento de escopo

```
program meu_prog
var x, y: integer
  procedure meu_proc(x: integer)
  var y: real
  begin
    read(y);
    read(a);
    x:=a+y
  end;
begin
  read(y);
  x:=x*y
end.
```

global		
meu_prog	nomeProg	
x	var	integer
y	var	integer
a	var	integer
meu_proc	nomeProc	



meu_proc		
x	param	integer
y	var	real

Tabela de símbolos

- Busca de informação
 - Sempre que um identificador do programa é utilizado
 - comando e fator
 - Verifica-se se foi declarado, seu tipo, etc.
-

Tabela de símbolos

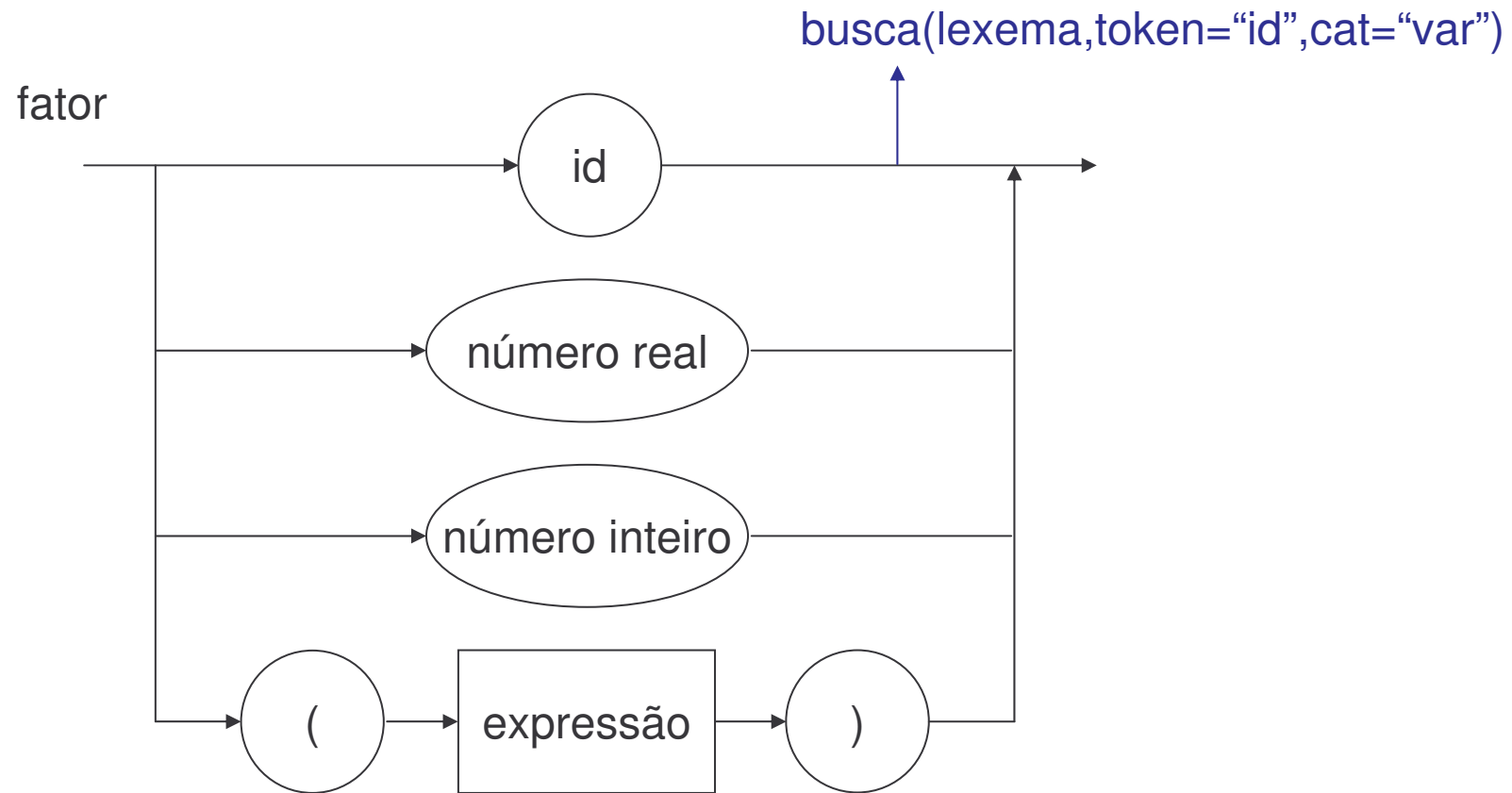
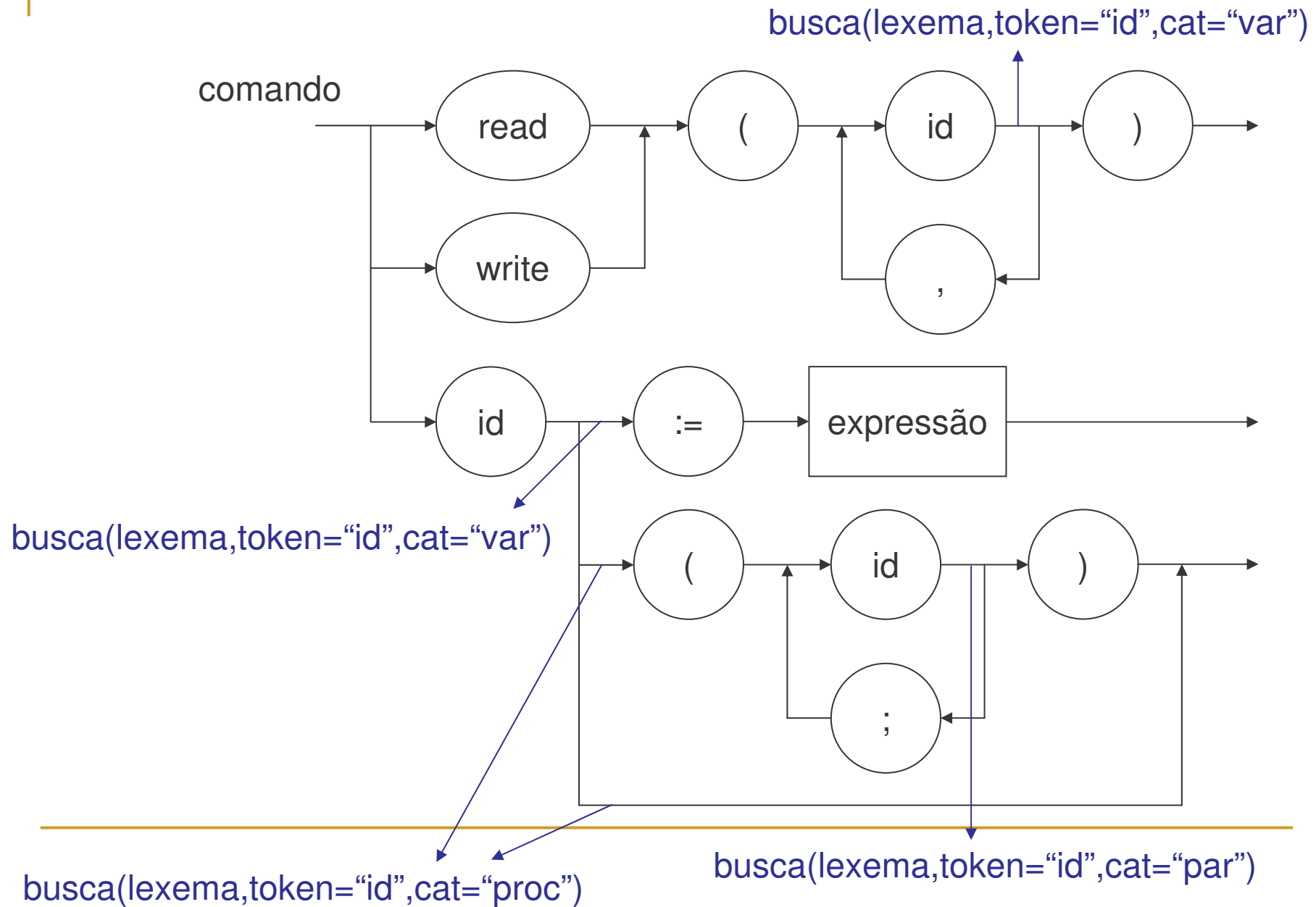


Tabela de símbolos



Tratamento semântico

- Verificação do uso adequado dos elementos do programa
 - Declaração de identificadores
 - Erro: identificador não declarado ou declarado duas vezes
 - Compatibilidade de tipos em comandos
 - Checagem de tipos
 - Concordância entre parâmetros formais e atuais, em termos de número, ordem e tipo
-

Tratamento semântico

- Declaração de identificadores
 - Verificado durante a construção da tabela de símbolos
- Compatibilidade de tipos
 - Atribuição: inteiro:=inteiro, real:=inteiro, string:=cadeia de caracteres
 - Normalmente, tem-se erro quando inteiro:=real
 - Conversão implícita (coerção) ou explícita dos tipos
 - Comandos de repetição: while booleano do..., if booleano then...
 - Expressões e tipos esperados pelos operadores: inteiro+inteiro, real*real, inteiro+real, inteiro/inteiro, booleano and booleano
 - Erro: inteiro+booleano
 - Arrays: vetor[integer]

Tratamento semântico

- Concordância entre parâmetros formais e atuais, em termos de número, ordem e tipo
 - Por exemplo, se declarado:
procedure p(var x: integer; var y: real)
 - Erros
 - procedure p(x:integer, y:integer) {erro de tipo}
 - procedure p(y:real, x:integer) {erro de ordem}
 - procedure p(x:integer) {erro de número}
 - Tratamento de escopo
 - Erro: variável local a um procedimento utilizada no programa principal
-

Tratamento semântico

- Tipos
 - Básicos (ou primitivos): booleano, char, inteiro, real
 - Estruturados: vetor, registro, ponteiro
 - Regras de compatibilidade de tipos
 - As regras de compatibilidade de tipos são geralmente da forma:
se duas expressões são **equivalentes**,
então retorne um certo tipo, senão erro
 - Precisa-se ter uma definição exata de quando duas expressões são equivalentes
-

Verificação de Tipos

- Atividade de assegurar que os operandos de um operador possuem tipos compatíveis
 - Entenda-se a atribuição como um operador binário
 - Tipo compatível: é um tipo válido para ser usado por um dado operador ou um tipo que pode ser convertido implicitamente em um tipo válido (**coerção**)
 - Coerção: conversão automática de tipo
 - Definida no projeto da linguagem e embutida na implementação
 - Um erro de tipo é a aplicação de um operador a um operando de tipo inadequado
-

Equivalência de Tipos

- Tipos primitivos: baseada no conceito de **inclusão**
 - Tipos definidos pelo usuário: baseada no conceito de **equivalência**
 - **Equivalência estrutural**: duas variáveis têm tipos compatíveis se os seus tipos tiverem estruturas idênticas
 - **Equivalência nominal**: duas variáveis têm tipos compatíveis somente se estiverem na mesma declaração ou em declarações que usam o mesmo nome de tipo
 - Exemplo:
 - C usa equivalência estrutural se os tipos forem primitivos e nominal se os tipos forem estruturados
-

Equivalência de Tipos

```
typedef float km;
typedef float mile;

km mile2km (mile m) {
    return (1.6093 * m);
}

main() {
    mile s = 200;
    km q = mile2km(s);
    s = mile2km(q);    //OK - eq estrutural
}
```

Equivalência de Tipos

```
typedef struct {  
    int a;  
} tipo1;
```

```
typedef struct {  
    int a;  
} tipo2;
```

```
int soma(tipo1 x, tipo2 y){  
    return (x.a + y.a);  
}
```

```
main() {  
    tipo1 x;  
    tipo2 y;  
  
    x.a = y.a = 1;  
    int z = soma(x, y); //OK  
    z = soma(x, x); //ERRO! -  
                    //eq nominal  
}
```

Tratamento semântico

- **Sistema de tipos:** coleção de regras que atuam sobre os tipos básicos da linguagem ou os estruturados, definidos ou não pelo usuário
 - Um **verificador de tipos** implementa um sistema de tipos, utilizando informações sobre a sintaxe da linguagem, a noção de tipos e as regras de compatibilidade de tipos
-

Tratamento semântico

- Verificador de tipos
 - Especificado na **gramática de atributos** e implementado como tal
 - Especificação do sistema de tipos
 - Compilação em mais de uma passagem, possivelmente
 - Comandado pela **análise sintática**
 - Compilação de uma única passagem
-

Tratamento semântico

- Exemplo: verificação de tipos na gramática de atributos

$\langle \text{exp} \rangle_1 ::= \langle \text{exp} \rangle_2 \text{ div id}$

se busca(id)=falso

então ERRO("variável não declarada")

senão se $\text{exp}_2.\text{tipo} \neq \text{inteiro}$ ou $\text{busca}(\text{id.tipo}) \neq \text{inteiro}$

então ERRO("tipos inválidos para a operação")

senão

$\text{exp}_1.\text{tipo} = \text{inteiro}$

$\text{exp}_1.\text{val} = \text{exp}_2.\text{val} / \text{id.val}$

Tratamento semântico

- Exemplo: verificação de tipos em uma regra sintática de atribuição de tipos iguais

procedimento atribuição(Seg)

Início

se (token = t_id)

então prox_token(cadeia,token)

se busca(cadeia,token,cat="var")=FALSE

então ERRO("variável não declarada")

senão tipo1:=recupera_tipo(cadeia,token,cat="var");

senão ERRO(Seg+{t_atrib});

se (token = t_atrib)

então prox_token(cadeia,token)

senão ERRO(Seg+{id});

tipo2 := expressao(Seg+{;});

se tipo1 <> tipo2 então ERRO("tipos incompatíveis na atribuição");

se (token=t_ponto-virgula)

então prox_token(cadeia,token)

senão ERRO(Seg+P(comandos));

fim

Tratamento semântico

- Exemplo: verificação de tipos em uma regra sintática de atribuição de tipos iguais

```
procedimento atribuição(Seg)
Início
se (token = t_id)
    então prox_token(cadeia,token)
        se busca(cadeia,token,cat="var")
            então ERRO("variável não declarada")
            senão tipo1:=recupera_tipo(token)
        senão ERRO(Seg+{t_atrib});
se (token = t_atrib)
    então prox_token(cadeia,token)
    senão ERRO(Seg+{id});
tipo2 := expressao(Seg+{;});
se tipo1 <> tipo2 então ERRO("tipos incompatíveis");
se (token=t_ponto-virgula)
    então prox_token(cadeia,token)
    senão ERRO(Seg+P(comandos));
fim
```

E quando a análise sintática é **ascendente**?
No caso deste exemplo, a verificação de tipos compatíveis será feita quando ocorrer uma redução pela regra sintática da atribuição