Construção de Compiladores

Análise Semântica – parte 2

Profa. Helena Caseli
helenacaseli@dc.ufscar.br

Análise Semântica

- Tabela de símbolos
  - Estrutura principal da compilação
  - Está relacionada a todas as etapas da compilação
    - Mas é na análise semântica que melhor se ajusta
      - Captura a sensitividade ao contexto e as ações executadas no decorrer do programa
    - Fundamental na geração de código
  - Permite saber, durante a compilação de um programa:
    - o tipo
    - o valor
    - o escopo

    de seus elementos (números e identificadores)
  - Pode ser utilizada para armazenar as palavras reservadas e símbolos especiais da linguagem

Análise Semântica

- Tabela de símbolos
  - Exemplo
    - Cada token tem atributos/informações diferentes associadas

Cadeia	Token	Categoria	Tipo	Valor	...
i	ident	var	inteiro	1	...
fat	ident	proc	-	-	...
2	num	-	inteiro	2	...
...					

    - Exemplo de atributos para uma **variável**
      - Tipo (inteira, real etc.), nome, endereço na memória, escopo (programa principal, função, etc.) entre outros
    - Para **vetor**, ainda seriam necessários atributos de tamanho do vetor, o valor de seus limites, etc.

Análise Semântica

- Tabela de símbolos
  - Principais operações
    - Inserir
      - Armazena informações fornecidas pelas declarações
    - Verificar
      - Recupera informação associada a um elemento declarado no programa quando esse elemento é utilizado
    - Remover
      - Remove (ou torna inacessível) a informação a respeito de um elemento declarado quando esse não é mais necessário
  - O comportamento da tabela de símbolos depende fortemente das propriedades da linguagem sendo compilada

Análise Semântica

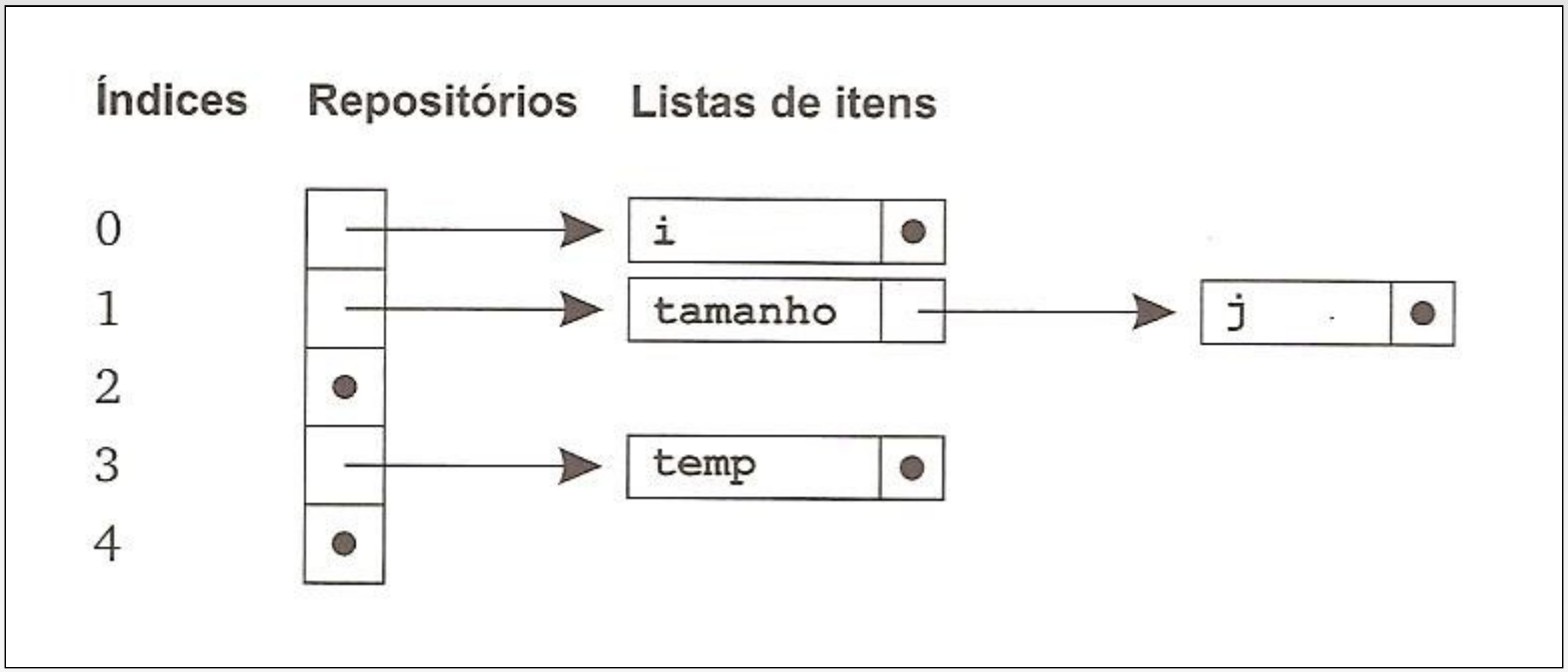
- Tabela de símbolos
  - Quando é acessada pelo compilador
    - Sempre que um elemento é mencionado no programa
      - Verificar ou incluir sua declaração
      - Verificar seu tipo, escopo ou alguma outra informação
      - Atualizar alguma informação associada ao identificador (por exemplo, valor)
      - Remover um elemento quando este não se faz mais necessário ao programa

Análise Semântica

- Tabela de símbolos
  - Como é frequentemente acessada, o acesso tem de ser eficiente
    - Implementação
      - Estática
      - Dinâmica: melhor opção
    - Estrutura de dados
      - Listas, matrizes
      - Árvores de busca (por exemplo, B e AVL)
    - Acesso
      - Sequencial, busca binária, etc.
      - *Hashing*: opção mais eficiente
        - O elemento do programa é a chave e a função *hash* indica sua posição na tabela de símbolos
        - Necessidade de tratamento de colisões

Análise Semântica

- Tabela de símbolos
  - Exemplo de *hashing* com resolução de colisões para a inclusão dos identificadores i, j, tamanho e temp



  - Tabela hashing com encadeamento separado

Análise Semântica

- Tabela de símbolos – Questões de Projeto
  - Tamanho da tabela
    - Tipicamente, de algumas centenas a mil “linhas”
    - Dependente da forma de implementação
      - Na implementação dinâmica, não é necessário se preocupar tanto com isso
  - Uma única tabela X várias tabelas
    - Diferentes declarações têm diferentes informações e atributos
      - Por exemplo, variáveis não têm número de argumentos, enquanto procedimentos têm
    - É interessante usar apenas uma quando a linguagem proíbe o mesmo identificador inclusive entre tipos diferentes de declarações

Análise Semântica

- Tabela de símbolos – Questões de Projeto
  - Escopo
    - Representação
      - Várias tabelas ou uma única tabela com a identificação do escopo (como um atributo ou por meio de listas ligadas, por exemplo) para cada identificador
    - Tratamento
      - Inserção de identificadores de mesmo nome, mas em níveis diferentes
      - Remoção de identificadores cujos escopos deixaram de existir
    - Regras gerais
      - Declaração antes do uso
      - Aninhamento mais próximo
    - → Quando, onde e por quanto tempo os identificadores vão ”existir”

Análise Semântica

- Tabela de símbolos
  - Escopo
    - Exemplo

Variáveis globais e locais com mesmo nome

```
program Ex;
var i,j: integer;

function f(tamanho: integer): integer;
var i,temp: char;

   procedure g;
   var j: real;
   begin
      ...
   end;

   procedure h;
   var j: ^char;
   begin
      ...
   end;

begin (* f *)
   ...
end;

begin (* programa principal *)
   ...
end.
```

Análise Semântica

- Tabela de símbolos
  - Escopo
    - Exemplo

Subrotinas aninhadas

```
program Ex;
var i,j: integer;

function f(tamanho: integer): integer;
var i,temp: char;

   procedure g;
   var j: real;
   begin
      ...
   end;

   procedure h;
   var j: ^char;
   begin
      ...
   end;

begin (* f *)
   ...
end;

begin (* programa principal *)
   ...
end.
```

Análise Semântica

- Tabela de símbolos
  - Escopo
    - Exemplo

Subrotinas aninhadas

Como implementar a tabela de símbolos para lidar com esses casos?

```
program Ex;
var i,j: integer;

function f(tamanho: integer): integer;
var i,temp: char;

   procedure g;
   var j: real;
   begin
      ...
   end;

   procedure h;
   var j: ^char;
   begin
      ...
   end;

begin (* f *)
   ...
end;

begin (* programa principal *)
   ...
end.
```

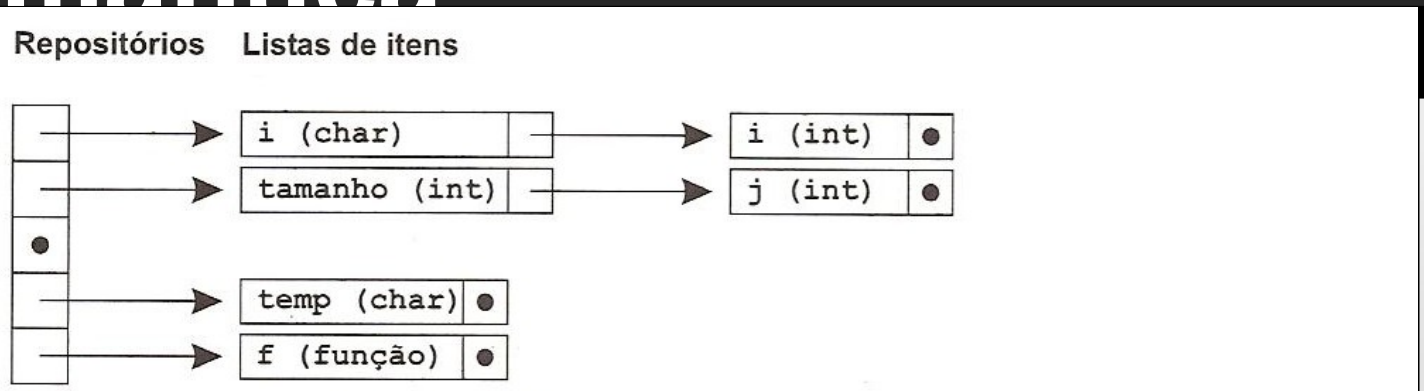
Análise Semântica

- Tabela de símb
  - Escopo

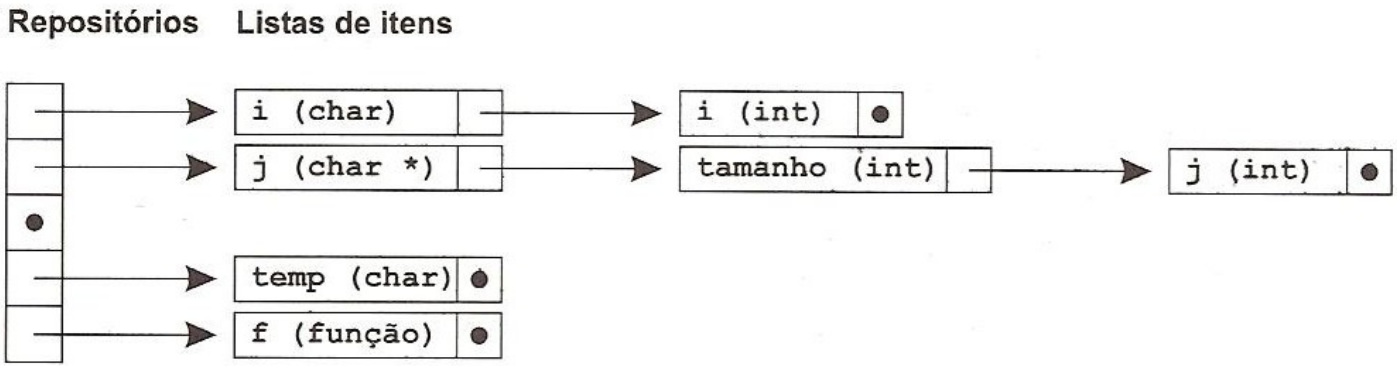
Opção 1

Age como uma pilha

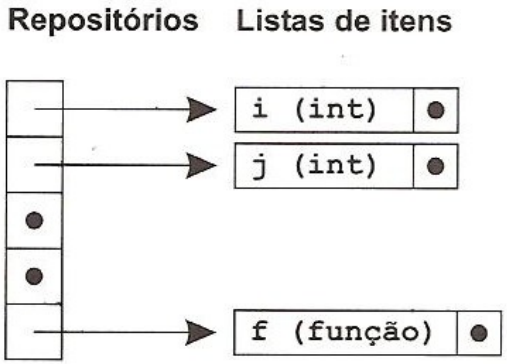
- Insere as declarações mais recentes, ocultando as antigas
- Remove as mais recentes, voltando ao escopo anterior
- Acesso às mais recentes



(a) Após o processamento das declarações do corpo de f



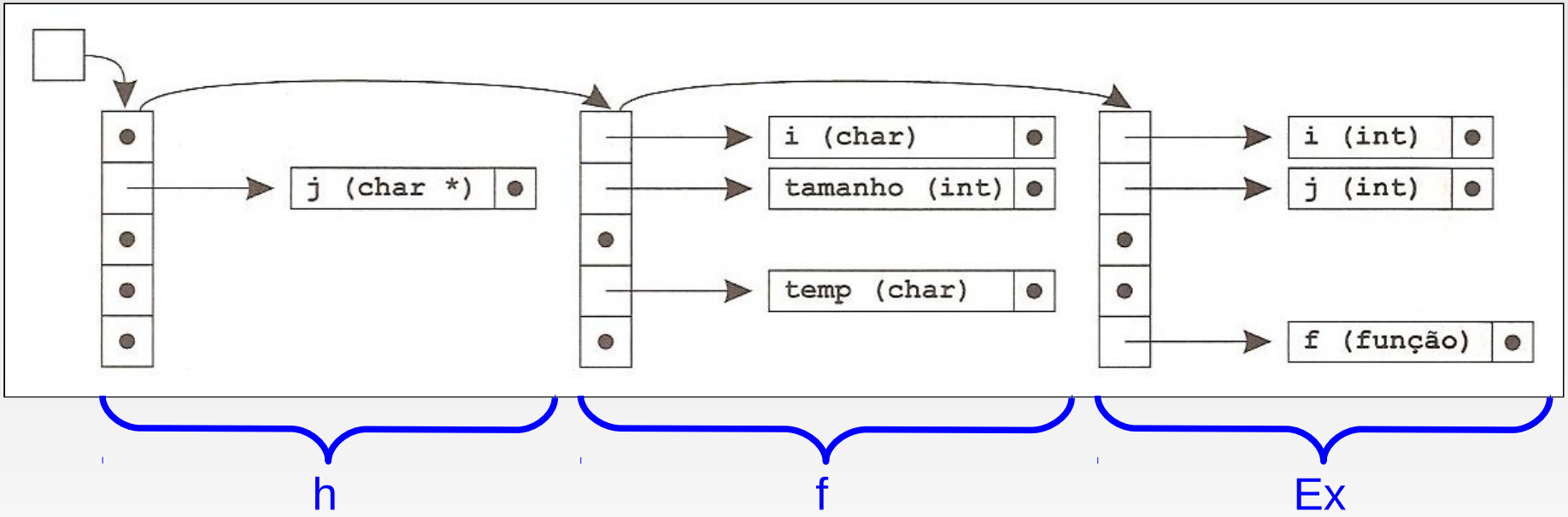
(b) Após o processamento da declaração da segunda declaração composta aninhada dentro do corpo de f



(c) Após abandonar o corpo de f (e apagar suas declarações)

Análise Semântica

- Tabela de símbolos
  - Escopo
    - Opção 2: tabelas separadas para cada escopo
      - Mudar o escopo requer apenas a mudança do ponteiro



- Outra opção ou adição: nível de aninhamento

2 1 0

Análise Semântica

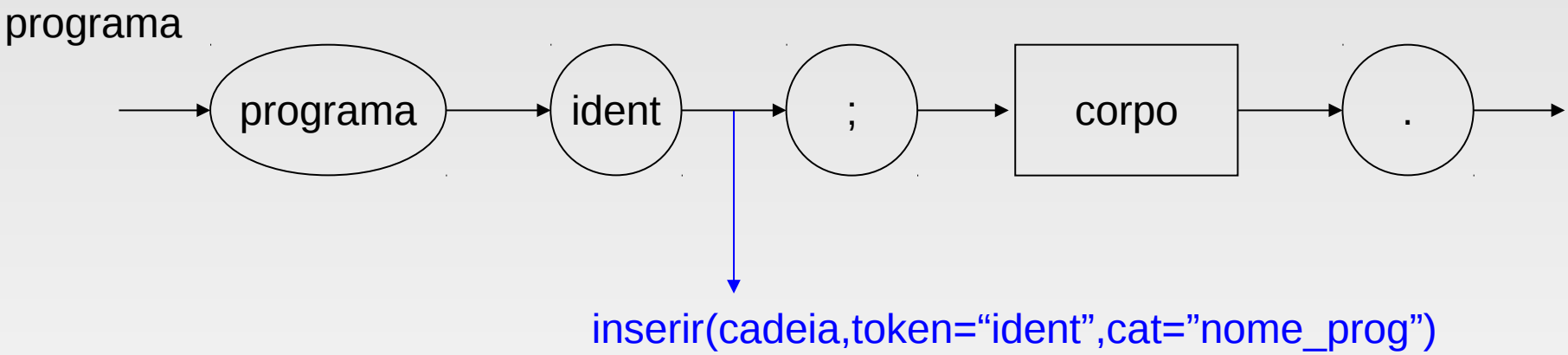
- Tabela de símbolos
  - Principais operações
    - Inserção de elementos na tabela
      - Associa regras semânticas às regras gramaticais
      - Verifica se o elemento já não consta na tabela
    - Busca de informação na tabela
      - Realizada antes da inserção
      - Busca informações para análise semântica
    - Remoção de elementos da tabela
      - Torna inacessíveis dados que não são mais necessários (por exemplo, após o escopo ter terminado)
      - Linguagens que permitem estruturação em blocos

    - → As sub-rotinas de inserção, busca e remoção podem ser inseridas diretamente na gramática de atributos

Análise Semântica

<programa> ::= programa ident ; <corpo> .

- **Inserção** de elementos na tabela
  - Principalmente nas declarações



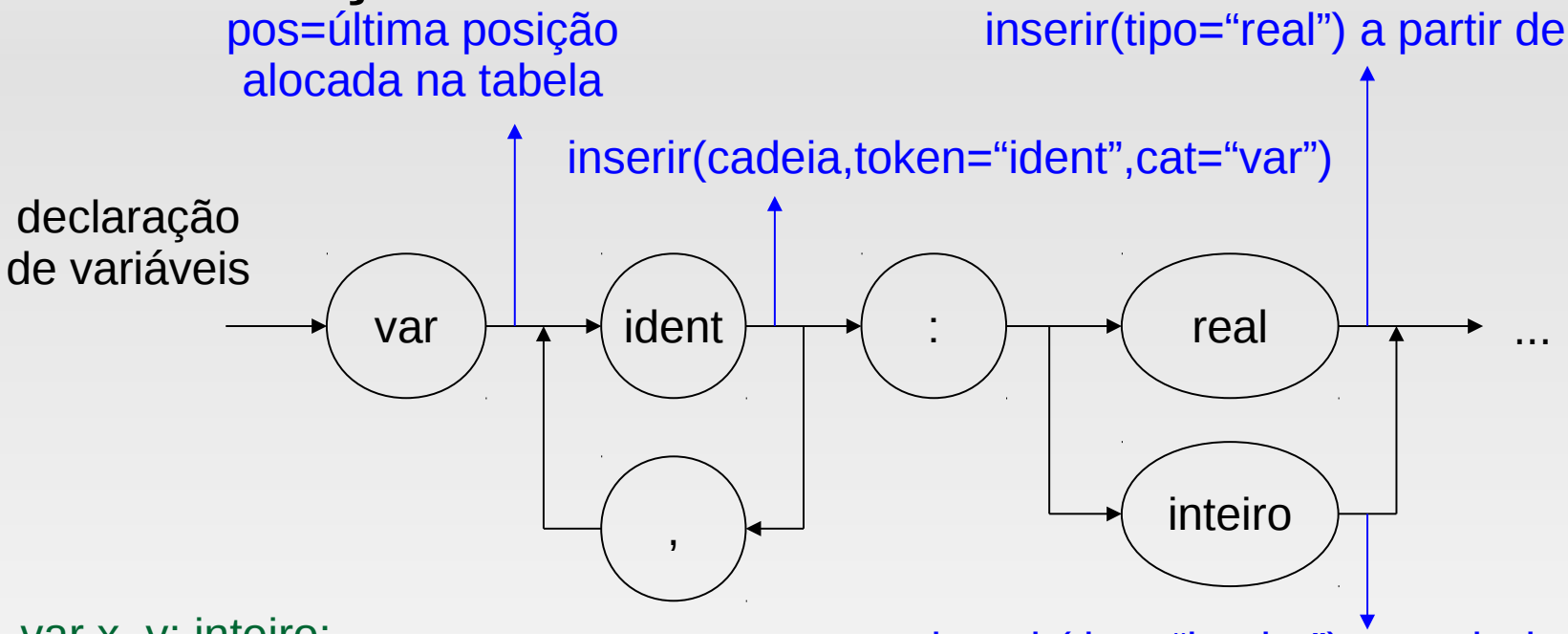
programa meu_prog ...

Cadeia	Token	Categoria	Tipo	Valor	...
meu_prog	ident	nome_prog	-	-	...

Análise Semântica

<decl_var> ::= var ident <mais_ident> : <tipo>
<mais_ident> ::= , ident <mais_ident> | ε
<tipo> ::= inteiro | real

- **Inserção** de elementos na tabela



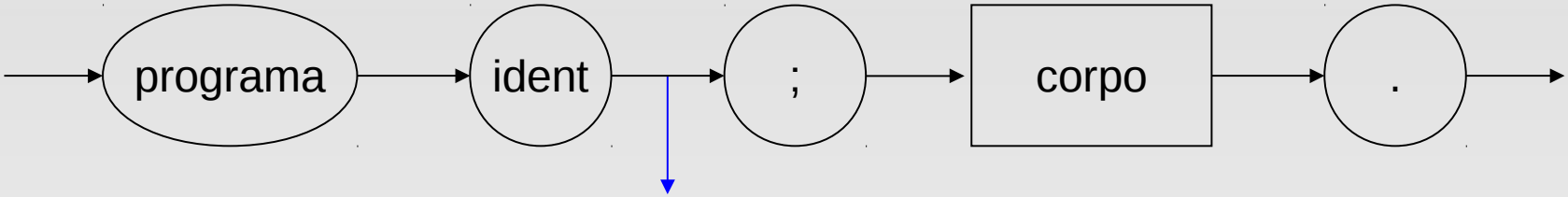
var x, y: inteiro;

Cadeia	Token	Categoria	Tipo	Valor	...
meu_prog	ident	nome_prog	-	-	...
x	ident	var	inteiro		...
y	ident	var	inteiro		...

Análise Semântica

- Exemplo de procedimento

programa




```
procedimento programa (Seg)
Inicio
        se (simbolo=programa) então obtem_simbolo(cadeia,simbolo)
        senão ERRO(Seg+{ident});
        se (simbolo=ident) então
                inserir(cadeia,"ident","nome_prog")
                obtem_simbolo(cadeia,simbolo)
        senão ERRO(Seg+{;});
        se (simbolo=simb_pvir) então obtem_simbolo(cadeia,simbolo)
        senão ERRO(Seg+P(corpo));
        corpo(Seg+{.});
        se (simbolo=simb_ponto) então obtem_simbolo(cadeia,simbolo)
        senão ERRO(Seg);
fim
```

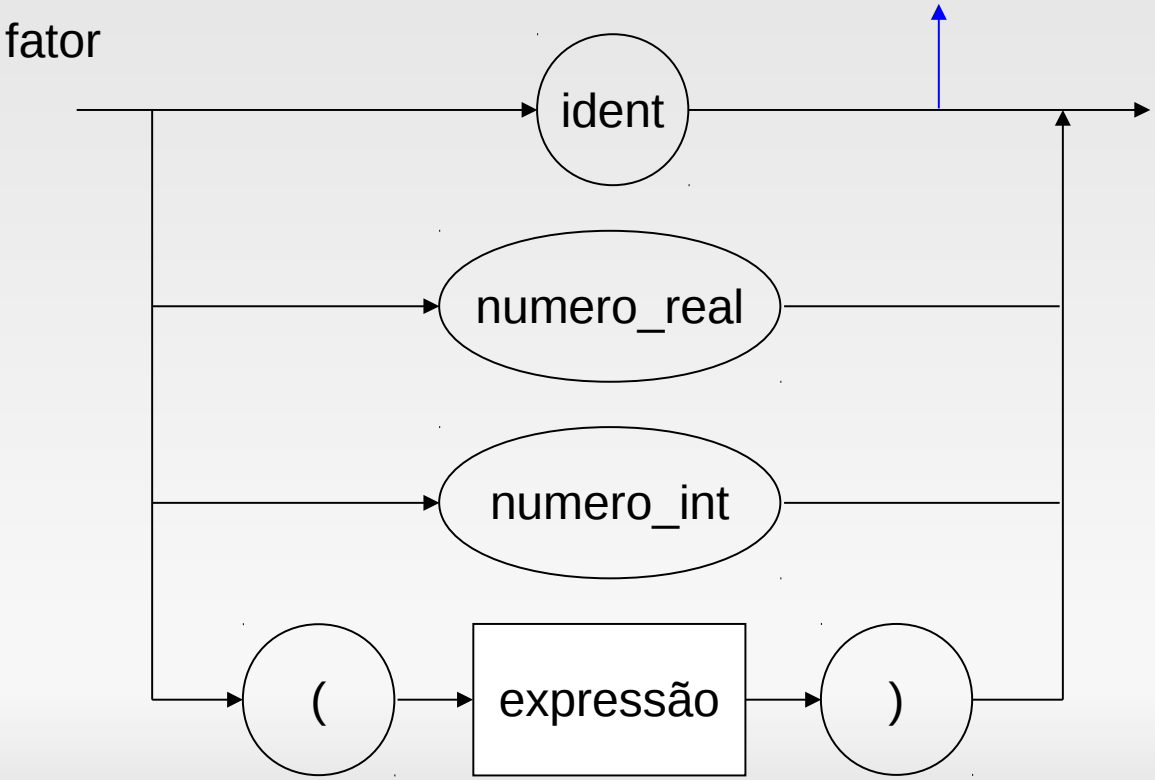
Análise Semântica

<fator> ::= ident | numero_real | numero_int
| (<expressão>)

- **Busca** de informação
  - Sempre que um elemento do programa é utilizado
  - Verifica-se se foi declarado, seu tipo, etc.

busca(cadeia,token="ident",cat="var",tipo,escopo)

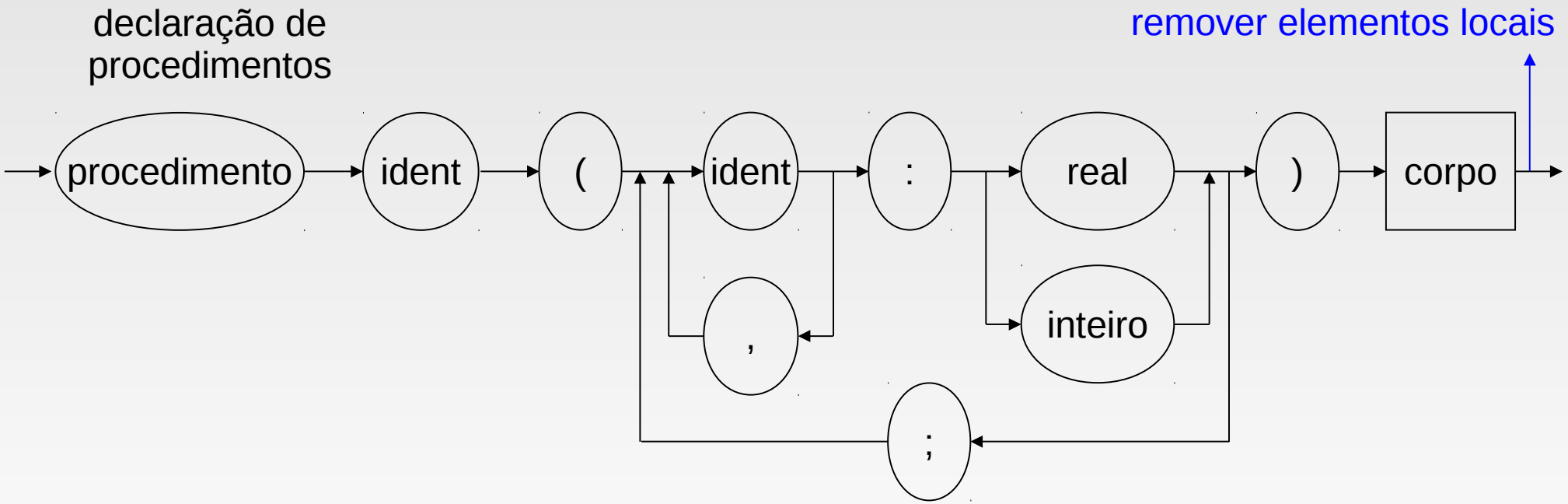
valores de retorno



Análise Semântica

<decl_proc> ::= procedimento ident (<par>)
<corpo>
<par> ::= ident <mais_ident> : <tipo> <mais_par>
<mais_par> ::= ; <par> <mais_par> | ε

- **Remoção** de elemento
  - Variáveis locais dos procedimentos
    - Atenção: parâmetros precisam ser mantidos



Análise Semântica

- Verificação do uso adequado dos elementos do programa
  - Declaração de identificadores
    - Erro: identificador não declarado ou declarado duas vezes
    - Verificado durante a construção da tabela de símbolos
  - Compatibilidade de tipos em comandos
    - Checagem de tipos é dependente do contexto
      - Atribuição: normalmente, tem-se erro quando inteiro:=real
      - Comandos de repetição: while booleano do, if booleano then
      - Expressões e tipos esperados pelos operadores
        - Erro: inteiro+booleano

Análise Semântica

- Verificação do uso adequado dos elementos do programa
  - Concordância entre parâmetros formais e atuais, em termos de número, ordem e tipo
    - Declaração: procedimento p(var x: inteiro; var y: real)
      - procedimento p(x:inteiro; y:inteiro)
      - procedimento p(x:real; y:inteiro)
      - procedimento p(x:inteiro)
  - Tratamento de escopo
    - Variável local utilizada no programa principal

Análise Semântica

- Verificação de tipos
  - Expressão de tipo
    - Tipo básico
      - Booleano, caractere, real, etc.
    - Formada por meio da aplicação de um construtor de tipos a outras expressões de tipo
      - Construtor de tipos: arrays, registros, ponteiros, funções, etc.
  - Sistema de tipos
    - Coleção de regras para as expressões de tipos

  - → 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

Análise Semântica

- Verificação de tipos
  - Equivalência de expressões de tipo

    function tipoIgual (t1, t2: TipoExp): booleano;
    - Retorna verdadeiro se t1 e t2 representam o mesmo tipo segundo as regras de equivalência de tipos da linguagem
  - 2 tipos principais
    - **Equivalência de nomes –** os tipos são compatíveis se
      - Têm o mesmo nome do tipo, definido pelo usuário ou primitivo
      - Ou aparecem na mesma declaração
    - **Equivalência estrutural –** os tipos são compatíveis se
      - Possuem a mesma estrutura (p. ex. representada por árvores sintáticas)
      - → Única disponível na ausência de nomes para tipos
    - → A maioria das linguagens implementa as duas estratégias de compatibilidade de tipos

Análise Semântica

- Verificação de tipos
  - Exemplo
    - Para as declarações abaixo

```
type t = array[1..20] of integer;
var a, b: array[1..20] of integer;
c: array[1..20] of integer;
d: t;
e, f: record
    a: integer;
    b: t
end
```

    - Pode-se observar que
      - (**a** e **b**), (**e** e **f**) e (**d**, **e.b** e **f.b**) têm equivalência de nomes
      - **a**, **b**, **c**, **d**, **e.b** e **f.b** têm tipos compatíveis estruturalmente

Análise Semântica

- Verificação de tipos
  - Exemplo
    - Para as declarações abaixo

```
type t = array[1..20] of integer;
var a, b: array[1..20] of integer;
c: array[1..20] of integer;
d: t;
e, f: record
    a: integer;
    b: t
end
```

    - Pode-se observar que
      - (**a** e **b**), (**e** e **f**) e (**d**, **e.b** e **f.b**) têm equivalência de nomes
      - **a**, **b**, **c**, **d**, **e.b** e **f.b** têm tipos compatíveis estruturalmente

Análise Semântica

<atribuicao> ::= ident := <expressao> ;

- Exemplo de verificação de tipos
  - Em uma regra sintática de atribuição de tipos iguais

```
procedimento atribuição(Seg)
Inicio
        se (simbolo=ident)
                então se busca(cadeia,simbolo,cat="var")=FALSE
                        então ERRO("variável não declarada")
                        senão tipo1:=recupera_tipo(cadeia,simbolo,cat="var");
                      obtem_simbolo(cadeia,simbolo)
                senão ERRO(Seg+{simb_atrib});
        se (simbolo=simb_atrib)
                então obtem_simbolo(cadeia,simbolo)
                senão ERRO(Seg+{id});
        expressao(tipo2);
        se tipoIgual( tipo1, tipo2) = false então ERRO("tipos incompatíveis na atribuição");
        se (simbolo=simb_ponto-virgula)
                então obtem_simbolo(cadeia,simbolo)
                senão ERRO(Seg+P(comandos));
fim
```

Análise Semântica

- Verificação de tipos
  - Pontos importantes
    - Polimorfismo – construções válidas para mais de um tipo
      - Uma função que troca o valor de duas variáveis de tipos iguais independentemente de quais tipos são
      - Uma função que conta os elementos de uma lista sem levar em consideração os tipos dos elementos da mesma
    - Sobrecarga – diversas declarações separadas que se aplicam a um mesmo nome
      - Mesmo operador, significados distintos dependendo do contexto
      - p. ex. + soma e + concatenação
    - Amarração estática X dinâmica
      - Estática: declaração explícita do tipo, boa para compilação
      - Dinâmica: tipo inferido na execução, boa para interpretação

Análise Semântica

- Considerações finais
  - 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 (como BNF)
    - Não há uniformidade na quantidade e nos tipos de análise estática semântica entre linguagens
    - Não existe um mapeamento claro da representação para o algoritmo correspondente
  - Análise é artesanal, dependente da linguagem de programação

Análise Semântica

- Considerações finais
  - Ferramentas para geração automática de analisadores semânticos
    - Algumas foram construídas, mas nenhuma atingiu as condições de uso e disponibilidade de Lex e Yacc
    - Algumas ferramentas interessantes baseadas em gramáticas de atributos são LINGUIST e GAG