V. Análise Semântica

1. Verificação de Unicidade

Em linguagens imperativas, um identificador deve ser declarado uma única vez em um determinado escopo (bloco).

2. Verificação de Classes de Identificadores

Os tokens *id* devem receber um atributo *classe* na tabela de símbolos, que especifica se são identificadores de variáveis, constantes, procedimentos, funções ou programa. Alguns comandos devem verificar a classe destes objetos.

Ex. – Pascal:

S → id := E; *{ se id.classe ∈ [constante, procedimento, programa] então ERRO }*

A verificação de classe e unicidade pode ser feita adicionando-se um campo *classe* no registro da tabela de símbolos:

- O analisador léxico insere o registro com o campo vazio
- Ações semânticas inseridas nas regras de declaração de identificadores devem verificar se o valor é vazio e então atribuir o valor de classe correspondente ao tipo de declaração (variáveis, constantes, procedimentos, funções etc.).
- Ações semânticas inseridas nos locais onde os identificadores são usados verificam a classe.

3. Verificação de Tipos

Tipos podem ser divididos em básicos (atômicos) ou construídos.

Ex. – Pascal:

- Tipos básicos: inteiro, real, caractere, lógico.
- Tipos construídos: vetores, conjuntos, registros.

3.1 Expressões de Tipos (exp_tipos)

Representam o tipo de uma construção de linguagem e são definidos da seguinte forma:

a) Um tipo básico é uma exp-tipo. Além dos tipos atômicos da linguagem, são considerados o tipo_erro e o tipo_vazio.
b) Uma exp_tipo pode receber um nome para identificá-la.
c) Uma exp_tipo pode ser formada através da aplicação de um operador, chamado de construtor de tipos, a uma outra exp_tipo.

São construtores de tipo:

a) Arranjos ou vetores: *Arranjo(I,T)* é uma exp_tipo que indica um arranjo de elementos do tipo *T*, indexados por um conjunto *I.*

   Ex. - Pascal: Var Nome: array [1..40] of char;

   Nome tem exp_tipo: *arranjo (1..40, caracter)*

b) Produtos: É o produto cartesiano entre duas exp_tipos.

Ex.: *inteiro x real*

c) Registros: É um produto onde os campos têm nomes específicos.

```
Ex.: Type TAluno = record
                       Nome: array[1..40] of char;
                       Numero: integer;
                       Nota :real;
                End;
```

TAluno é representado pela exp_tipo: *registro((nome x arranjo(1..40,caractere)) x (número x inteiro) x (nota x real))*

d) Apontadores: Denota o endereço de um objeto na memória.

Ex.: Var P:^Taluno;

A exp_tipo de P é representada por: *apontador (Taluno)*

e) Funções: É o mapeamento de elementos do domínio em elementos da imagem. Em linguagens de programação, o mapeamento se dá entre o domínio de tipos *D* e o intervalo de tipos *R*. É representado pela exp_tipo: $D \rightarrow R$.

Ex.: *mod* possui a exp_tipo*: inteiro x inteiro → inteiro*.
Ex.: function f(a,b:char): ^integer; é representada por:
caractere x caractere → apontador(inteiro)

3.2 Especificação de um Verificador de Tipos

O verificador de tipos é um esquema de tradução que sintetiza o tipo de uma expressão a partir do tipo de suas sub-expressões.

Uma Linguagem Simples

Dada a gramática:

P → D ; E
D → D ; D | id : T
T → char | integer | array[num]of T | ^T
E → const | id | E mod E | E[E] | E^

- A linguagem possui 2 tipos básicos. A construção *array[num]of T* é representada pela exp_tipo *arranjo(1..num, T)* e ^*T* por *apontador(T).*

- O verificador de tipos tem a função de incluir o tipo dos identificadores na tabela de símbolos. A primeira regra semântica do E.T. faz esta tarefa:

D → id : T *{ id.posição*→tipo:=*T.tipo }*

- O atributo *T.tipo* é sintetizado nas regras:

T → char *{ T.tipo := caracter }*
T → integer *{ T.tipo := inteiro }*
T → ^T_1 *{ T.tipo := apontador (T_1.tipo) }*
T → array[num]of T_1 *{ T.tipo = arranjo(1..num.lex, T_1.tipo) }*

- As regras *P→D;E* e *D→D;D* garantem que todas as variáveis serão declaradas antes de *E* gerar o conjunto de expressões do programa.

Verificação de Tipos de Expressões

- A regra:

E → cosnt *{ E.tipo := const.tipo }*

sintetiza o atributo *E.tipo* a partir do tipo da constante encontrada pelo analisador léxico.

- A regra:

E→ id *{ E.tipo := id.posição→ tipo }*

sintetiza o atributo *E.tipo* através do tipo do identificador, armazenado na tabela de símbolos.

- O tipo da função *mod* é verificado a partir do tipo dos seus operandos:

E → E_1 mod E_2 *{ se E_1.tipo = inteiro E E_2.tipo = inteiro então E.tipo:=inteiro senão ERRO}*

- Em uma referência a arranjos, o índice deve ser inteiro para que o tipo de expressão seja definido.

E → E_1 [E_2] *{ se E_2.tipo = inteiro E E_1.tipo = arranjo(i,t) então E.tipo:=t senão ERRO}*

- Referências a objetos apontados são da forma:

E → E_1^ *{ se E_1.tipo = apontador(t) então E.tipo:=t senão ERRO}*

Verificação de Tipos dos Comandos

S → id := E *{ se NÃO compatível(id.tipo,E.tipo) então ERRO}*

S→ if E then S_1 *{ se E.tipo ≠ lógico então ERRO}*

S → while E do S_1 *{ se E.tipo ≠ lógico então ERRO}*

Verificação de funções

Para referência a funções:

E → id(P) *{ se id.tipo = P.tipo→t então E.tipo:= t senão ERRO}*

P → E Z *{ se Z.tipo ≠ tipo_vazio então P.tipo:= E.tipo X Z.tipo senão P.tipo:= E.tipo}*

Z → ,E Z_1 *{ se Z_1.tipo ≠ tipo_vazio então Z.tipo:= E.tipo X Z_1.tipo senão Z.tipo:= E.tipo}*

Z → λ *{ Z.tipo := tipo_vazio}*

Ex. - Pascal:

```
Var A : char;                                  /* A.tipo := caracter */
Function F(x:real;y:integer):char;             /*F.tipo:= real xinteiro→caracter*/

.....

Begin
   A := F ( 1.0, 2);
.....
```