Análise Semântica e Tratamento de Erros Dependentes de Contexto

O componente Semântico de uma LP

Tarefas da Análise Semântica

Implementação da Tabelas de Símbolos

Ações Semânticas em Compiladores Dirigidos por Sintaxe e Erros da Análise Semântica

TS implementada estaticamente como uma “pilha”

```
Type categoria = (constante, tipo, variavel, procedimento,funcao, parametro);
     classet = (valor, referencia, procedimento, funcao);
     dim = record
              inf, sup: integer
           end;
     item = record
              ident: string[tam_max];
              nivel: integer;
              case categ: categoria of
                constante: (case tipoc: integer of
                                  1: (valori: integer);
                                  2: (valorc: char);
                                  3: (valorr: real);
                                  4: (valors: string);
                                  5: (valorb: boolean););
                tipo: (nbytes: integer; dimensao: dim; tipo_elem: integer);
                procedimento: (npar1: integer; end1; integer);
                funcao: (npar2: integer; end2: integer; tipo_f: integer);
                parametro: (classe: classet; end3: integer; tipo_p: integer);
                variavel: (end4: integer; tipo_v: integer)
           end;
     TS: record
           pilha: array [1..max] of item;
           topo: integer
         end;
```

Simplificação: arrays de inteiros de 1 dimensão, índices inteiros

Os 11 campos da TS para FRANKIE com algumas extensões

Grupos que tratam de TIPOS (char, float/double, string, vetor, struct, union, enum) estenderão ela com os descritores

TS:	ident, nível, categoria, tipo, ender, nbytes, valor, classetransf, npar, inf, sup

Ações Semânticas em Compiladores Dirigidos por Sintaxe

Programa e Bloco

<programa> ::=
program <identificador> **;**
<bloco>**.**

<bloco> ::=
[<parte de definições de constantes>]
[<parte de definições de tipos>]
[<parte de declarações de variáveis>]
[<parte de declarações de sub-rotinas>]
<comando composto>

Declarações

<parte de definições de constantes> :: =

const <definição de constante>

{; <definição de constante>} ;

<definição de constante> ::= <identificador> = <constante>

Rs(0) Rs(0´)

<constante> ::= [+|-] (<identificador> | <numero>)

Rs(0´´) Rs(0´´´)

<parte das definições de tipo> ::=

typedef <type-specifier> <declarator list1> ;

<type-specifier>::= (**boolean** | **int** | **char** | **float** | **double** | <struct-or-union> | <enum> | <identificador>)

Rs(9)

Rs(9)

<declarator list1 > ::= <identificador> [[<numero>]]

Rs(1) Rs(10) Rs(12) Rs(11) Rs(2)

{ , <identificador> [[<numero>]] }

Rs(1) Rs(10) Rs(12) Rs(11) Rs(2)

<parte de declarações de variáveis> ::=
<declaration> {; <declaration>};

<declaration> ::= <declaration-specifiers>
<declarator list2>

Rs(4)

<declaration-specifiers> ::= <type-specifier>

<declarator list2> ::= <identificador> {,
<identificador> }

Rs(3)

Rs(3)

<parte de declarações de subrotinas> ::=

{<declaração de procedimento> ; | <declaração de função> **;**}

<declaração de procedimento> ::=

procedure <identificador> 

[(<parâmetros formais>)] **;** <bloco> 

<type-specifier> <identificador>

Rs(7)

Rs(6)

Rs(13)

([<parâmetros formais>]) ;

<bloco_função>

Rs(8)

<bloco_função> ::=

{ {<parte de declarações de variáveis>}

[<comando> { **;** <comando>}]

}

<parâmetros formais> ::=

<seção de parâmetros formais>

{ ; <seção de parâmetros formais>}

Rs(20)

<seção de parâmetros formais> ::=

[var] <lista de identificadores 3> : <identificador>

Rs(14) ou Rs(15) Rs(19)

<lista de identificadores 3> ::=

<identificador> {, <identificador> }

Rs(18) Rs(18)

<comando composto ::=

begin <comando> { **;** <comando>} **end**

<comado> ::=

<atribuição>

| <chamada de procedimento>

Se simbolo = “read” then

<lista de identificadores 3> ::=

<identificador> {, <identificador> }

Rs(21) Rs(21)

senão se símbolo = “write” then

<lista de expressões 1> ::=

<expressão> {, <expressão>} Não faz nada

senão <chamada de procedimento def usuário>

| <comando composto>

| <comando condicional 1>

| <comando repetitivo 1>

| <repita>

| <for>

| <case>

| <bloco_função> { não precisam colocar se não quiserem }

| return <expresão>

<atribuição>::=

<variável> **:=** <expressão>

Rs(25´) Rs(22´)

<chamada de procedimento> ::=

<identificador> [**(** <lista de expressões> **)**]

Rs(21) Rs(25´) ou Rs(25) Rs(24)

<comando condicional 1> ::=

if <expressão> **then** <comando>

[**else** <comando>]

<comando repetitivo 1> ::=

while <expressão> **do** <comando>

Expressões

<expressão> ::=

<expressão simples> [<relação>
    <expressão simples>]

<relação> ::=

    = | **<>** | **<** | **<=** | **>=** | **>**

<expressão simples> ::=

    [**+** | **-**] <termo> {(**+** | **-** | **or**) <termo>}

<termo> ::=

    <fator> {(***** | **div** | **and** | ***/***) <fator> }

<fator> ::=

<variavel>

| <número_int>

Rs(26)

| <número_real>

Rs(26)

| <chamada de função>

| **(** <expressão> **)**

| **not** <fator>

<variável> ::=

<identificador>

Rs(21)

| <identificador> **[** <expressão> **]**

Rs(21)

<lista de expressões> ::=

<expressão> {**,** <expressão> }

Rs(23) Rs(23)

<chamada de função> ::=

<identificador> [(<lista de expressões>)]

Rs(21) Rs(22´) ou Rs(22) Rs(24)

Observações

- A rotina semântica 8 é usada em procedimentos e funções no final de <bloco> e é muito importante!
- Nela todos os identificadores declarados no nível corrente são apagados da TS,
  - MENOS as informações sobre o tipo e passagem de cada parâmetro da função ou procedimento. Devemos encadear estas informações (só estas 2, na ordem de declaração dos parâmetros) na entrada do procedimento (ou função) para ser usada posteriormente para checar o uso dos parâmetros

- Observem que criamos várias listas de identificadores com nomes diferentes (mesmo que sejam iguais na forma), pois precisamos aplicar rotinas semânticas diferentes.

Erros da Análise Semântica

- Os erros gerados pelas rotinas semânticas vistas em classe são simples, pois a linguagem é um Pascal Simplificado com tipos inteiros e booleano, construtor array e com comandos de atribuição, while-do, if-then-else, chamada de procedimento e E/S.

- Portanto, os erros levantados pelas rotinas semânticas são:

- identificador (de todas as categorias) já declarado
- tipo não definido
- índice superior menor que índice inferior
- identificador não declarado
- incompatibilidade no número de parâmetros
- erros relativos à categoria:
  - “função não definida”,
  - “função, variável, parâmetro, ou constante não definidos”;
  - “procedimento não definido”;
  - “função, procedimento, variável ou parâmetro não definidos”

Tratamento de Algumas Extensões

- Case
- Break
- Módulos
- STRUCT
- Enum

Comando CASE

26. <comando condicional 2> ::= **case** <expressão> **of** <elemento do case> { ; <elemento do case> } **end**

27. <elemento do case>::= <constante> { , <constante> } : <comando>

5. <constante> ::= [+|-] (<identificador> | <numero_inteiro>)

- Comando Case do Pascal: as constantes devem ser únicas.
- Checar na análise da declaração se houve duplicação.

```
Case i of
1,2: ...
2,3: ...
^ erro
```

Comando Break

- Comandos que permitem desviar o fluxo de controle de comandos de repetição ou de seleção, por exemplo, o break da linguagem C:

  - desviam o controle para o fim de tais comandos mais aninhados.
  - Não deve haver break sem um desses comandos de seleção (case) ou repetição senão um erro será gerado.

Módulos

- Em Módula-2, os procedimentos e funções tem nome no cabeçalho e no end.

- Eles devem ser idênticos (“casados”).
-
- O compilador deve checar se o mesmo nome foi usado nos dois lugares senão um erro será gerado.

Tipo Record

11. <record> ::= record <lista de campos> end
12. <lista de campos> ::= <lista de identificadores> : <tipo>
    { ; <lista de identificadores> : <tipo> }

15. <lista de identificadores> ::= <identificador> {, <identificador>}

- As variáveis do tipo Record (ou Struct) devem ser usadas com os seus campos.
- Na declaração do tipo os campos devem ser únicos.
- Checar na análise da declaração se houve duplicação, indicando erro

Tipo Enumerado

10. <enumerado> ::= (<lista de identificadores>)

15. <lista de identificadores> ::= <identificador> {, <identificador>}

- O tipo enumerado deve ter constantes distintas.
- Checar na análise da declaração se houve duplicação, indicando erro

Exercício

- Dado o programa em FRANKIE abaixo, pede-se:
  - Mostre na Tabela de Símbolos (PS) os identificadores e atributos que estão disponíveis a cada início de comando composto.
  - Use os nomes dados abaixo para os 11 atributos que podem aparecer na TS para FRANKIE.

TS: ident, nível, categoria, tipo, ender, nbytes, valor, classetransf, npar, inf, sup

```
Program x;
Int i;
Procedure p1 (k :int );
    Int i;
    Procedure p2;
   begin
          i := i + 2
    end;
Begin
i:= k;
p2;
Write(i)
End; {p1}
```

```
Procedure p3;
    procedure p4;
    int i;
    begin
          i := 4
    end;
Begin
    i := i+3; write(i); p4;
    write(i)
End; {p3}
Begin {pp}
i := 10; p1(i); write(i);
p3; write(i)
End.
```