5.2 Ações Semânticas para a Construção de Tabelas de Símbolos

Esta seção apresenta esquemas de tradução que reconhecem declarações de variáveis e geram tabelas de símbolos. Inicialmente, é apresentado um esquema que gera tabelas de símbolos para programas monolíticos, isto é, formados por um único bloco. Posteriormente, esse esquema é estendido para permitir a geração de tabelas para programas bloco-estruturados com procedimentos aninhados.

EXEMPLO 5.5 Esquema de tradução para gerar tabela de símbolos para bloco unitário

O esquema de tradução a seguir constrói a tabela de símbolos para uma lista de declarações.

P	→	M D	
M	→	ε	{ desloc = 0 }
D	→	D ; D	
D	→	id : T	{ adSimb (id.nome, T.tipo, desloc); desloc = desloc + T.tam }
T	→	int	{ T.tipo = int; T.tam = 4 }
T	→	real	{ T.tipo = real; T.tam = 8 }
T	→	array [num] of T1	{ T.tipo = matriz (num.val, T1.tipo); T.tam = num.val * T1.tam }
T	→	^ T1	{ T.tipo = ponteiro (T1.tipo); T.tam = 4 }

O não–teminal M é empilhado logo no início do processo para atribuir o valor zero à variável *desloc*. Essa variável contém o próximo endereço disponível na área de dados do procedimento. Observe que *desloc* não é atributo de símbolo da gramática. A rotina *adSimb* adiciona um novo identificador à Tabela de Símbolos, com seu tipo e endereço. O atributo *tam* contém o tamanho em bytes para os diferentes tipos de dados aceitos nas declarações.

Observe que a redução M←ε poderia ser realizada em qualquer momento do processo de construção da tabela, pois sempre se tem a palavra vazia no topo da pilha. Contudo, ao programar o analisador, deve-se ter em mente que essa redução só deve ser efetuada no início da análise. Esse artifício de introduzir na gramática do esquema de tradução um não terminal artificial (caso de M) para forçar uma ação especial é uma técnica comumente usada.

A Figura 5.6 mostra a árvore de derivação e as ações semânticas para a declaração **a: int ; b: real**.

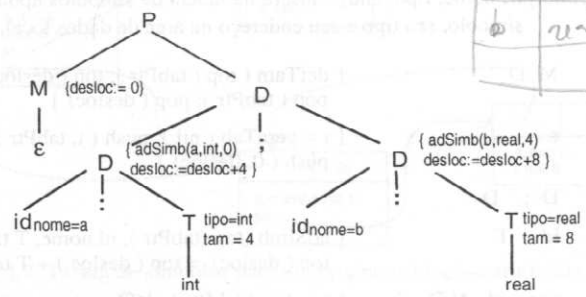

Figura 5.6 Árvore de derivação para a declaração **a: int; b: real** .

O esquema de tradução a seguir estende o esquema anterior, incluindo uma regra de produção para a declaração de procedimentos. Isso permitirá a geração de tabelas de símbolos para procedimentos embutidos.

EXEMPLO 5.6 Esquema de tradução para gerar uma árvore de tabelas de símbolos

Este esquema gera uma árvore de tabelas de símbolos para programas bloco–estruturados, obtendo-se uma tabela de símbolos para cada procedimento.

São usadas duas pilhas, *tabPtr* e *desloc*: a primeira contém um ponteiro para a tabela de símbolos de cada procedimento, e a segunda contém o próximo endereço (de memória local) disponível na área de dados do procedimento. Durante a compilação, o ponteiro no topo de *tabPtr* aponta para a tabela de símbolos do procedimento em análise. Cada tabela tem um ponteiro para a tabela do procedimento envolvente. Os símbolos não–terminais M e N servem para gerar, respectivamente, a tabela raiz (tabela de símbolos relativa ao programa principal, que contém as variáveis globais) e as tabelas referentes aos demais procedimentos. Observe que a redução N←ε (que poderia ser efetuada a qualquer momento) deverá ser efetuada logo após ter sido empilhada a seqüência de *tokens* "proc id ;".

As seguintes funções fazem parte do esquema de tradução:

- geraTab(ptr) - gera uma tabela de símbolos (filha da tabela apontada por ptr) e retorna um ponteiro para a tabela gerada;
- defTam(ptr, tam) - armazena na tabela de símbolos apontada por ptr o tamanho da área de dados local do procedimento correspondente;

- adProc(ptr, nome, pt) - insere na tabela de símbolos apontada por ptr o nome do procedimento e o ponteiro para a tabela de símbolos desse procedimento;
- adSimb(ptr, nome, tipo, end) - insere na tabela de símbolos apontada por ptr um novo símbolo, seu tipo e seu endereço na área de dados local.

P	→	M D	{ defTam (top (tabPtr), top (desloc)); pop (tabPtr); pop (desloc) }
M	→	ε	{ t = geraTab (nil); push (t, tabPtr); push (0, desloc) }
D	→	D ; D	
D	→	id : T	{ adSimb (top (tabPtr), id.nome, T.tipo, top (desloc)); top (desloc) = top (desloc) + T.tam }
D	→	proc id ; N D ; S	{ t = top (tabPtr); defTam (t, top (desloc)); pop (tabPtr); pop (desloc) ; adProc (top (tabPtr), id.nome, t) }
N	→	ε	{ t = geraTab (top (tabPtr)); push (t, tabPtr); push (0, desloc) }
T	→	int	{ T.tipo = int; T.tam = 4 }
T	→	real	{ T.tipo = real; T.tam = 8 }
T	→	array [num] of T1	{ T.tipo = matriz (num.val, T1.tipo); T.tam = num.val * T1.tam }
T	→	^ T1	{ T.tipo = ponteiro (T1.tipo); T.tam = 4 }

A Figura 5.7 mostra as tabelas de símbolos que seriam geradas para o seguinte programa bloco–estruturado:

```
a : real;
b : int;
proc p1;
    c : real;
    ---
end p1;
proc p2;
    d : array[5] of int;
    proc p3;
        e, f: real;
        ---
    end p3;
    ---
end p2;
---
```

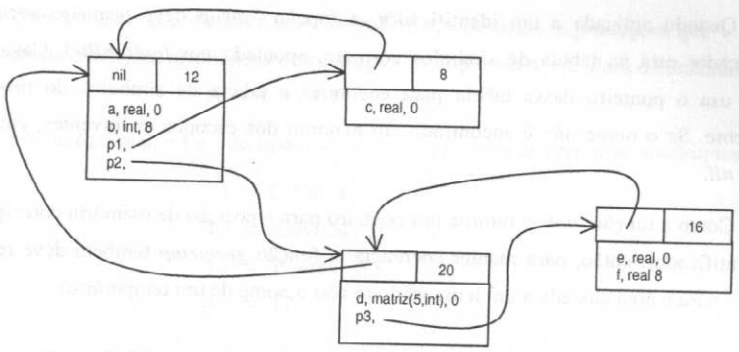

Figura 5.7 Tabela de símbolos para um programa bloco–estruturado.

5.3 Geração de Código para Comandos de Atribuição

Esta seção trata da geração de código para comandos de atribuição. São apresentados, também, esquemas de tradução correlatos, para conversão de tipos e para endereçamento de elementos de matrizes.

EXEMPLO 5.7 Esquema de tradução para comandos de atribuição

Este esquema de tradução gera código de três-endereços para comandos de atribuição. A função *lookup(id.nome)* procura o identificador armazenado *id.nome* na tabela de símbolos. Se existe uma entrada na tabela para esse identificador, a função retorna o índice correspondente ao mesmo; caso contrário, retorna *nil*. A ação semântica *geracod* grava comandos de três–endereços num arquivo de saída.

A diferença entre este esquema e o apresentado no Exemplo 5.3 é que aquele implementa uma gramática de atributos. Lá as ações envolvem apenas o cálculo de atributos (não há efeitos colaterais) e, no final, o código gerado fica armazenado no atributo *cod* de S.

S	→ id := E	{ p = lookup (id.nome); if p ≠ nil then geracod(p ":=" E.ptr) else erro}
E	→ E1 + E2	{ E.ptr = geratemp; geracod(E.ptr ":=" E1.ptr "+" E2.ptr) }
E	→ E1 * E2	{ E.ptr = geratemp; geracod(E.ptr ":=" E1.ptr "*" E2.ptr) }
E	→ - E1	{ E.ptr = geratemp; geracod(E.ptr ":=" "-u" E1.ptr)}
E	→ (E1)	{ E.ptr = E1.ptr }
E	→ id	{ p = lookup(id.nome); if p ≠ nil then E.ptr = p else erro }