

Fig. 5.13

5.6.2 ESTRUTURA DE DADOS DO GRAFO SINTÁTICO

Para programarmos o AS cujo funcionamento foi exemplificado em 5.6.1, devemos armazenar o grafo sintático na "memória" de um computador sob a forma de uma tabela, que denominaremos de TABGRAFO. Para isso, é necessário introduzir estruturas de dados que possam representar os grafos, com seus nós e arcos.

Cada nó do grafo será representado por um elemento de TABGRAFO que segue a estrutura de dados representada graficamente na fig. 5.14.

```

RECORD
  SIM : INTEGER; //ÍNDICE DO
  TER : BOOLEAN; //É TERMINAL?
  SEM : INTEGER; //ROTINA
  ALT : INTEGER; //ALTERNATIVO
  SUG : INTEGER; //PRÓXIMO
END;
  
```

SIM	TER	SEM
ALT		SUC

```

IF TER
  SIM APONTA TABT
ELSE
  SIM APONTA TABNT
  
```

Fig. 5.14

Os símbolos terminais e não-terminais da gramática são guardados em duas tabelas, que denominaremos de TABT e TABNT, respectivamente, cujas primeiras entradas têm índi-

ce 1 (um). A TABNT é uma tabela com dois campos (NOME e PRIM) por entrada: NOME contém uma cadeia de até 6 caracteres com o nome do não-terminal, ajustado à esquerda; PRIM contém um índice de uma entrada para a TABGRAFO e aponta para o elemento desta correspondente ao primeiro nó do grafo das produções desse não-terminal. A TABT é a própria tabela de símbolos reservados, descrita em 2.3 e cuja construção descrevemos em 2.4.

Na fig. 5.14, o campo SIM de cada elemento n_e da TABGRAFO contém um índice para a TABT ou para TABNT. Se o nó n_g correspondente do grafo contiver um símbolo terminal, o campo TER contém o valor true e SIM contém o índice da entrada da TABT onde está armazenado o terminal correspondente. Se o nó n_g contiver um não-terminal, o campo TER de n_e contém o valor false e SIM contém o índice da entrada da TABNT onde está armazenado o não-terminal correspondente. O campo ALT contém o índice para a TABGRAFO onde se encontra o elemento correspondente à alternativa do nó n_g ; se n_g não tem alternativa, ALT deve conter o número 0 (zero). Analogamente, SUC contém o índice do elemento da TABGRAFO correspondente ao sucessor de n_g . O campo SEM contém um número inteiro, indicando uma rotina "semântica" que deverá ser executada após o reconhecimento do nó, como veremos no item 6.2.

Se o nó n_g for um λ -nó, é gerado um elemento da TABGRAFO com TER=true, SIM=0. O mesmo se passa para cada λ -alternativa, a qual deve gerar um elemento da TABGRAFO com os campos adicionais ALT=0 e SUC=0, já que pela definição de ESLL(1)-gramáticas, cada λ -alternativa não tem alternativa e aponta para o vazio, isto é, não tem sucessor.

Na fig. 5.15, apresentamos um diagrama da estrutura de dados correspondente ao grafo da fig. 5.11. Nesse diagrama simplificamos os elementos como introduzidos na fig. 5.14; se o nó correspondente é terminal, o próprio símbolo terminal é colocado no elemento; se é né não-terminal, colocamos um apontador para um elemento da TABNT, que é representado por uma estrutura à parte. Omitimos, neste exemplo, o campo SEM. Desta maneira, recaímos nos diagramas usados por Wirth em /WIR 76/.

Na fig. 5.16 encontram-se as tabelas correspondentes ao grafo da fig. 5.11 e ao diagrama da fig. 5.15. Note-se que não é necessário construir-se esse diagrama como passo intermediário entre o grafo e a tabela, já que os grafos construídos a partir das ERE-gramáticas, segundo as regras dadas em 4.8, contêm diretamente todas as informações para a tabela, o que, como vimos, não acontece com o grafo sintático da PASCAL de /J-W 74/.

5.6.3 O CARREGADOR SINTÁTICO

Neste item descrevemos um procedimento que lê registros onde se dão informações sobre o grafo sintático, e que produz as três tabelas do item anterior. Esses registros não contêm a numeração absoluta dos nós, como exemplificado na fig. 5.11, mas sim um número relativo ao primeiro nó de cada subgrafo de um não-terminal, isto é, em cada um desses subgrafos iniciamos novamente a numeração dos nós a partir de 1 (um). Esses números relativos são usados também em ALT e SUC. Dessa maneira consegue-se uma grande flexibilidade na codificação do grafo. O carregador sintático deve transformar esses números relativos em absolutos. Cada subgrafo de um não-terminal é precedido de um registro especial, contendo o código 'C' (de "cabeça"), e que é usado para se introduzir o valor de PRIM em TABNT, já que, ao ser lido, o carregador pode deduzir a numeração absoluta de seu primeiro nó. Os nós dos grafos são introduzidos por registros contendo o código 'N' para não-terminais e 'T' para terminais. Na fig. 5.17 apresentamos os registros de entrada para o carregador, correspondentes ao grafo da fig. 5.11. O campo NUMNO contém o número relativo de cada nó. A numeração relativa dos nós 8, 9 e 10 do subgrafo de M é 1, 2 e 3, respectivamente. Novamente, deixamos de colocar os números das rotinas "semânticas". O λ -nó é representado por um registro cujo campo NOMER contém valor branco.

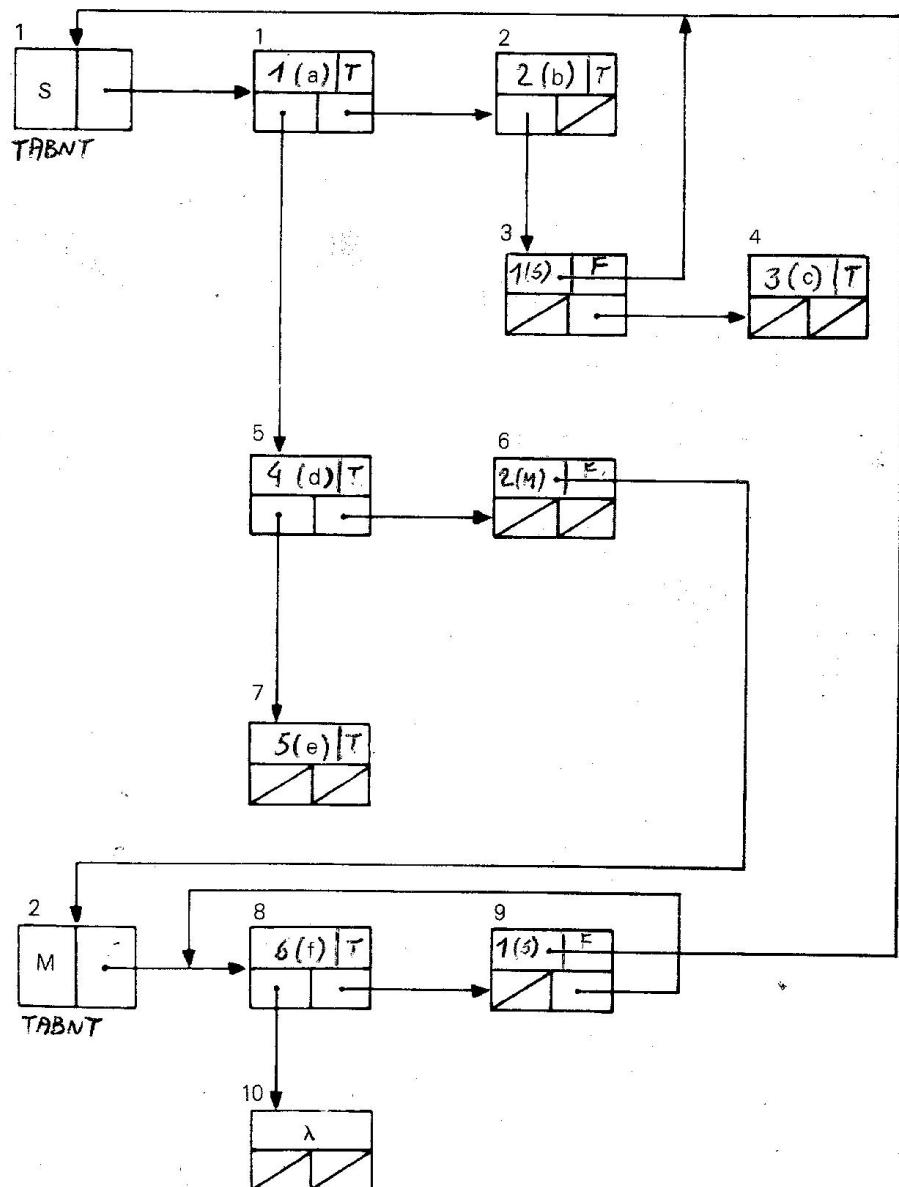


Fig. 5.15

Damos a seguir, em notação informal, uma descrição do carregador. Os significados de algumas variáveis são os seguintes: MAXT — índice do último elemento da TABT; MAXNT idem para a TABNT; INDPREM — índice do elemento correspondente ao primeiro nó de um

TABGRAFO					TABT		TABNT	
1	TER	SIM	ALT	SUC	SEM	1 2 3 4 5 6 7 8 9 10	a b c d e f	NOME PRIM 1 S 1 2 M 8

Fig. 5.16

TIPO	NOMER	NUMNO	ALTR	SUCR	SEMR
C	S				
T	A	1	5	2	
T	B	2	3	0	
N	S	3	0	4	
T	C	4	0	0	
T	D	5	7	6	
N	M	6	0	0	
T	E	7	0	0	
C	M				
T	F	1	3	2	
N	S	2	0	1	
T		3	0	0	

Fig. 5.17

subgrafo para um certo não-terminal; NOMAX — número do nó de maior número relativo de um subgrafo. Note-se que supomos não haver nenhum registro que não seja do tipo 'C', 'T' ou 'N'. Como se pode observar, a ordem dos registros que se seguem a um registro do tipo cabeça não é relevante. Indicaremos comandos compostos por $\lceil \rfloor$.

É interessante observar que a TABT, contendo os símbolos terminais, é na verdade a própria tabela de símbolos reservados (v. 2.3 e 2.4). Vemos assim que o carregador deve substituir a rotina programada para carregar essa tabela e descrita em 2.4. Para simplificar a descrição, usaremos no carregador uma construção linear de TABT, e não um método de "Hashing" como proposto em 2.4.

MAXT:=0; MAXNT:=0; INDPRIM:=1; NOMAX:=0;
enquanto houverem registros para serem lidos

```

    leia um registro com TIPO, NOMER, NUMNO, ALTR, SUCR, SEMR;
    se TIPO='C'
    então  $\lceil$ INDPRIM:=INDPRIM+NOMAX; NOMAX:=0;
    se NOMER não se encontra na TABNT
    então  $\lceil$ MAXNT:=MAXNT+1;
          TABNT[MAXNT].NOME:=NOMER;
          TABNT[MAXNT].PRIM:=INDPRIM $\rfloor$ ;

```

```

se existe E tal que  $1 \leq E \leq \text{MAXNT}$  e  $\text{TABNT}[E].NOME = \text{NOMER}$ 
então se  $\text{TABNT}[E].PRIM = 0$ 
    então  $\text{TABNT}[E].PRIM := \text{INDPRIM}$ 
    senão ERRO (dois cabeças para um mesmo não-terminal);
senão  $I := \text{INDPRIM} + \text{NUMO} - 1$ ;
    se  $\text{TIPO} = 'T'$  e  $\text{NOMER} \neq \emptyset$  (é terminal, diferente de  $\lambda$ -nó)
        então  $\text{se } \text{NOMER} \text{ não se encontra na TABT}$ 
            então  $\text{MAXT} := \text{MAXT} + 1;$ 
             $\text{TABT}[\text{MAXT}] := \text{NOMER};$ 
            seja E tal que  $1 \leq E \leq \text{MAXT}$  e  $\text{TABT}[E] = \text{NOMER}$ 
                faça  $\text{TABGRAFO}[I].TER := \text{true}$ ;
    se  $\text{TIPO} = 'N'$ 
        então  $\text{se } \text{NOMER} \text{ não se encontra na TABNT}$ 
            então  $\text{MAXNT} := \text{MAXNT} + 1;$ 
             $\text{TABNT}[\text{MAXNT}].NOME := \text{NOMER};$ 
             $\text{TABNT}[\text{MAXNT}].PRIM := 0;$ 
            seja E tal que  $1 \leq E \leq \text{MAXNT}$  e  $\text{TABNT}[E].NOME = \text{NOMER}$ 
                faça  $\text{TABGRAFO}[I].TER := \text{false}$ ;
    se  $\text{NOMER} = \emptyset$  ( $\lambda$ -nó)
        então  $\text{TABGRAFO}[I].SIM := 0$ 
        senão  $\text{TABGRAFO}[I].SIM := E;$ 
    se  $\text{ALTR} \neq 0$ 
        então  $\text{TABGRAFO}[I].ALT := \text{INDPRIM} + \text{ALTR} - 1$ 
        senão  $\text{TABGRAFO}[I].ALT := 0;$ 
    se  $\text{SUCR} \neq 0$ 
        então  $\text{TABGRAFO}[I].SUC := \text{INDPRIM} + \text{SUCR} - 1$ 
        senão  $\text{TABGRAFO}[I].SUC := 0;$ 
     $\text{TABGRAFO}[I].SEM := \$EMR;$ 
    se  $\text{NOMAX} < \text{NUMNO}$  então  $\text{NOMAX} := \text{NUMNO}$ 

```

5.6.4 O PROCEDIMENTO ANSIN

Neste item damos, em PASCAL, o procedimento do AS /SET 79/, que terá o nome ANSIN. Comentários inseridos entre os comandos devem tornar comprehensível o funcionamento do procedimento, principalmente seguindo-se as indicações dadas em 5.6.1. Algumas observações breves sobre esse procedimento:

— As constantes MAXG, MAXNT e MAXT, que indicam o índice máximo das tabelas TABGRAFO, TABNT e TABT, foram declaradas com valores apropriados para o grafo da linguagem PASCAL dado no apêndice I e devem ser alteradas para outras linguagens. MAXK dá o tamanho máximo da pilha do analisador; esse tamanho deve ser baseado em testes de programas típicos, para se obter um limite razoável. Note-se que o procedimento EMPILHA não testa a ultrapassagem desse limite, já que TOPO foi declarada do tipo 0..MAXK; teoricamente, esse intervalo ("range") deveria ser testado pelos programas-objeto produzidos pelos compiladores PASCAL, o que em geral não é o caso.

— Os procedimentos EMPILHA e DESEMPILHA manipulam a pilha do analisador. ANALEX devolve apenas o 1º parâmetro do AL conforme descrito em 3.4. CARREGADOR carrega as tabelas do grafo; não é necessário que ele siga os passos da rotina descrita em 5.6.3; ele poderia ler os registros de entrada, produzir as tabelas e gravá-las em um arquivo intermediário e deduzir o valor de MAXG, MAXNT e MAXT; o CARREGADOR leria essas tabelas diretamente do arquivo intermediário; infelizmente esse não é um processo simples em PASCAL, pela falta de limites variáveis de "arrays". Também não existe em PASCAL iniciali-

zação de "arrays" ou a declaração destes como constantes, o que poderia eliminar a necessidade da carga das tabelas.

— O procedimento TRATAERRO, que faz o tratamento de erros sintáticos, será descrito no item 5.8.

— Alguns pontos do procedimento ANSIN foram marcados com (+) e (++). Eles indicam os pontos onde deve ser manipulada a pilha sintática (ver menção à mesma em 2.3), o que será descrito em 5.7.

— O programa principal, que chama os procedimentos CARREGADOR e ANSIN, não é apresentado. O parâmetro OBJETIVO é o índice do símbolo inicial da gramática (ou do grafo) na TABNT. *Atenção:* para que ANSIN funcione perfeitamente, é necessário que se introduza em TABGRAFO[0] um nó inicial contendo, pela ordem, os seguintes campos: (false,m,0,0,n) onde m é o índice de OBJETIVO na TABNT e n é o número da rotina semântica a ser executada no fim da compilação; foi com esse nó em mente que inicializamos a primeira célula da pilha com 0 (zero).

```
type alpha = record SIMB: packed array [1:6] of char end;
const MAXG = 244; MAXNT = 13; MAXT = 66; MAXK = 50;
var TABGRAFO: array [0..MAXG] of
    record TER: Boolean;
        SIM, ALT, SUC: 0..MAXG;
        SEM: integer end;
    TABNT: array [1..MAXNT] of
        record NOME: alpha;
            PRIM: 1..MAXG end;
    TABT: array [1..MAXT] of alpha;
    K: array [1..MAXK] of 0..MAXG; /* pilha do analisador */
    TOPO: 0..MAXG; /* índice do topo de K */
procedure DESEMPILHA (var p: 0..MAXG);
begin p := K[TOPO]; TOPO := TOPO - 1 end;
procedure EMPILHA (p: 0..MAXG);
begin TOPO := TOPO + 1; K[TOPO] := p end;
procedure ANALEX (var PROX: alpha); /* analisador léxico; retorna o próximo símbolo */
procedure CARREGADOR; /* carrega as tabelas */
procedure ANSIN (OBJETIVO: 1..MAXNT; var SUCESSO: Boolean);
var CONTINUE: Boolean; /* controle da continuação da interpretação */
    I: 0..MAXG; /* índice para nós do grafo; indica o nó sendo interpretado */
    IU: 0..MAXG; /* índice do primeiro nó não encontrado; usado pela rotina de erro */
    ENT: alpha; /* próximo símbolo de entrada */
procedure TRATAERRO (IU: 1..MAXG); /* tratamento de erro; recebe o índice do primeiro nó
                                    não encontrado */
begin
    CARREGADOR; /* carrega estruturas */
    ANALEX (ENT); /* lê o primeiro símbolo */
    /* (+) */
    TOPO := 1; K[1] := 0; /* inicialização da pilha */
    I := TABNT[OBJETIVO].PRIM; /* aponta para o primeiro nó do símbolo inicial */
    IU := I;
    CONTINUE := true;
    while CONTINUE do
        if I ≠ 0 /* não é o fim de uma produção? */
            then if TABGRAFO[I].TER /* é um terminal? */
                then if TABGRAFO[I].SIM = 0 /* é um λ-nó? */
                    then begin /* vai para o sucessor */
                        I := TABGRAFO[I].SUC;
```

```

IU := I /* início da lista de erro */
end
else if TABTITABGRAFO[I].SIM = ENT /* reconheceu o terminal? */
then begin /* (++) */
    ANALEX(ENT); /* lê o próximo */
    I := TABGRAFO[I].SUC; /* vai para o sucessor */
    IU := I /* início da lista de erro */
end
else if TABGRAFO[I].ALT ≠ 0 /* há alternativa? */
then /* tome a alternativa */
    I := TABGRAFO[I].ALT
else /* TRATAERRO(IU) */
begin /* é um não-terminal */
    EMPILHA(I);
    /* vai para o primeiro nó do não-terminal */
    I := TABNT[TABGRAFO[I].SIM].PRIM
end
else /* é o fim de um lado direito de uma produção */
if TOPO ≠ 0 /* pilha não está vazia? */
then begin
    DESEMPILHA();
    /* vai para o sucessor do não-terminal */
    I := TABGRAFO[I].SUC;
    IU := I /* início da lista de erro */
end
else begin
    if ENT = '$' /* fim de arquivo? */
        then SUCESSO := true
        else SUCESSO := false; /* estão sobrando símbolos */
    CONTINUE := false /* encerra a análise */
end
end;

```

begin
sucesso := false;
continue := false;
end;
 - ENQUANTO NÃO HÁ
 ROTINA DE TRATAMENTO
 DE ERRO!

Antes de deixarmos este item, é interessante observar que as gramáticas ESLL(1), com as restrições vistas (v. 5.5), foram definidas baseando-se no algoritmo acima. Essas restrições é que permitem que o algoritmo analise corretamente a cadeia de entrada. Por exemplo, a gramática $S \rightarrow ab \mid ac$ geraria um grafo (e uma tabela) o qual, se interpretado pelo analisador, não seria capaz de reconhecer a cadeia 'ac'. De fato, o primeiro 'a' forçaria sempre a escolha da primeira alternativa da gramática. As gramáticas ESLL(1) são tais que o próximo símbolo de entrada determina univocamente o caminho a ser seguido no grafo correspondente. Lembra-mos que isso implica no fato de elas serem do tipo "forte" (v. 5.4): não é necessário usar-se o passado da análise (isto é, o trecho da forma sentencial já analisado) para tomar a decisão de qual alternativa do grafo deve ser seguida.

É importante notar também que existem grafos que seguem as restrições dadas em 5.5, sendo portanto adequados para o algoritmo descrito, mas que não provêm de gramáticas ESLL(1), segundo as regras de construção dos grafos dadas em 4.8. Na fig. 5.18 apresentamos um trecho de grafo que se encaixa dentro dessa situação. Ele corresponde a comandos separados por um número qualquer de ';'. Esse efeito é conseguido na PASCAL através da existência do comando vazio. Note-se que a produção `block ::= begin [statm] ; ;* { end } +` é ESLL(1) e a ela corresponde um grafo como o da fig. 5.18 trocando-se as posições do `end` e dos ';.

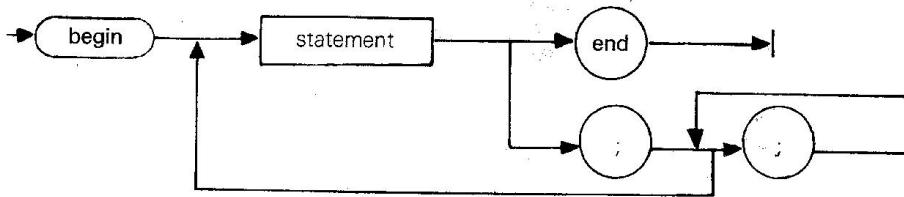


Fig. 5.18

Casos como o da fig. 5.18 sugerem a técnica de projetar grafos diretamente, satisfazendo às restrições de 5.5, sem que se passe pela fase de escrever uma gramática ESLL(1). O uso dessa classe de gramáticas neste texto deveu-se primordialmente a motivos didáticos, tendo-se com isso feito uma introdução às gramáticas formais e à geração de analisadores sintáticos a partir das mesmas.

5.7 A PILHA SINTÁTICA

Como já mencionamos anteriormente (v. 5.6.1), o AS aqui introduzido não efetua, a rigor, uma análise sintática. Para cada novo símbolo de entrada, o AS percorre simplesmente um trecho do grafo sintático, ignorando inclusive todos os símbolos já reconhecidos. Assim, não são construídas as formas sentenciais intermediárias entre a cadeia de entrada e o símbolo inicial da gramática, o que constituiria realmente uma análise sintática. Vejamos como se deve introduzir algumas modificações a fim de que o procedimento ANSIN de 5.6.4 construa as formas sentenciais.

Seja uma gramática ESLL(1) $G = (V_N, V_T, S, P)$, e $F_1 = e_1 \dots e_{i-1} e_i \dots e_n \in L(G)$. Sejam as formas sentenciais $S \xrightarrow{*} F_{q+1} \xrightarrow{*} F_q \xrightarrow{*} F_1$. Vejamos como se pode encarar a passagem de F_q para F_{q+1} , durante o reconhecimento efetuado pelo AS. Seja $F_q = \gamma_1 \gamma_2 \dots \gamma_m e_i \dots e_n$, com $\gamma_1, \dots, \gamma_m \in V_N \cup V_T$.

Suponhamos que no próximo reconhecimento direto uma parte de F_q seja reconhecida como o objetivo atual, o não-terminal N , isto é, queremos representar o passo $\gamma_1 \dots \gamma_m e_i \dots e_{i+p} \xrightarrow{*} N$, onde $1 \leq j \leq m+1$ e $i-1 \leq p \leq n$; $j = m+1$ e $p = i-1$ indicam, respectivamente, a ausência de símbolos de $\gamma_j \dots \gamma_m$ e de $e_{i+p} \dots e_n$.

Nesse caso, N está em um nó apontado necessariamente pelo topo da pilha K do analisador. Após esse passo, teremos $F_{q+1} = \gamma_1 \gamma_2 \dots \gamma_{j-1} N e_{i+p+1} \dots e_n$.

Vê-se portanto que, para se obter as formas sentenciais, é preciso efetuar o seguinte:

- Guardar toda a parte $\gamma_1 \dots \gamma_m$ já reconhecida.
- O reconhecimento direto de um não-terminal N pode englobar os símbolos mais à direita de $\gamma_1 \dots \gamma_m$ e símbolos mais à esquerda da cadeia de entrada $e_i \dots e_n$. Todos esses símbolos devem ser substituídos por N .
- Depois dessa substituição, N torna-se o símbolo mais à direita da parte já reconhecida.

Essas considerações sugerem uma estrutura de pilha que denominaremos de *pilha sintática*, abreviada por PS, para armazenar a parte já reconhecida $\gamma_1 \dots \gamma_m$ das formas sentenciais, já que os reconhecimentos implicam em substituição, sempre, dos símbolos mais à direita dessa subcadeia, por um não-terminal N que é colocado na posição mais à direita da parte já reconhecida. Pode-se também ver que cada novo símbolo terminal e reconhecido durante o percurso do grafo deve ser empilhado na PS. Por outro lado, sempre que se chegar ao fim do lado direito de uma produção (isto é, quando um nó reconhecido não tiver sucessor ou, em termos do ANSIN, $I=0$ no começo da malha `while`), devemos desempilhar da PS todo esse lado direito e, em seguida, empilhar o não-terminal N do lado esquerdo da produção. Es-

te N está no nó apontado pelo topo da pilha K. Aqui surge um interessante problema: qual o tamanho do lado direito da produção de N que devemos reconhecer, isto é, quantos símbolos de uma forma sentencial devem ser substituídos por N? Note-se que as gramáticas ESLL(1) podem ter expressões regulares (estendidas) nos lados direitos das produções. Com isso, alguns lados direitos deixam de ter tamanho fixo. Por exemplo, ao querermos aplicar a produção $N \rightarrow ab^*c$ em um reconhecimento direto, podemos ter um número qualquer de 'b's na forma sentencial. Esse problema pode ser resolvido com muita simplicidade em nosso método de análise sintática. Para isso, basta introduzir um campo adicional em cada célula da pilha K do analisador. Se atingirmos um nó n do grafo, com um não-terminal N, n é empilhado em K, como anteriormente, agora junto com um número r que é o índice da próxima célula livre da PS. Isso é compreensível: a partir dessa célula da PS é que serão colocados os símbolos que formarão a subcadeia $\gamma_j \dots \gamma_m e_i \dots e_{i+p}$, como já vimos. Quando chegamos ao fim de um lado direito de N, desempilhamos de K o par (n, r) ; desempilhamos todas as células da PS desde a célula r inclusive, até o topo; vamos ao nó n e lá encontramos o apontador para a tabela de não-terminais, indicando N; empilhamos N (na célula r da PS). A inicialização de K deve ser, agora, feita com o par $(0, 1)$, se a primeira célula de PS tem índice 1.

Na fig. 5.19 apresentamos um exemplo de funcionamento do AS, agora com a PS, para a gramática G_{17} do item 5.6.1, cujo grafo está na fig. 5.11. Usamos a cadeia 'dfaec', cuja análise já foi exemplificada naquele item. A ordem das colunas indica a ordem de execução das respectivas ações. Elementos em branco indicam a repetição do elemento anterior na mesma coluna.

Note-se que esse método funciona perfeitamente quando é reconhecida uma produção com lado direito igual a λ . Este seria o caso da cadeia 'dfadc', que deixamos para o leitor verificar.

As alterações a serem feitas no programa do AS são as seguintes:

- Declarations adicionais e mudanças nas declarações já feitas.

```

type...
const...; MAXPS = 100;
var...
  K: array[1..MAXK] of record NO: 0..MAXG; R:1..MAXPS end;
  TOPO...
  PS: array [1..MAXPS] of alpha; /* pilha sintática */
  TOPPS: 0..MAXPS; /* índice do topo da PS */
procedure DESEMPILHA (var p: 0..MAXG);
begin p := K[TOPO].NO; TOPPS := K[TOPO].R; TOPO := TOPO - 1;
  /* empilha o não-terminal na PS */
  PS[TOPPS] := TABNT[TABGRAFO[p].SIM].NOME end;
procedure EMPILHA (p: 0..MAXG);
begin TOPO := TOPO + 1; K[TOPO].NO := p; K[TOPO].R := TOPPS + 1 end;

b) Alterações no procedimento ANSIN. Os pontos onde essas alterações devem ser feitas são aqueles marcados em ANSIN com (+) e (++).

(+):
begin
  TOPO := 1; K[1].NO := 0; K[1].R := 1; /* inicialização da pilha do AS */
  TOPPS := 0;
  I := TABNT...
(++):

begin
  TOPPS := TOPPS + 1; PS[TOPPS] := ENT; /* empilha em PS o símbolo reconhecido */
  ANALEX...

```

símbolo lido	cadeia restante	nó visitado	pilha do analisador	pilha sintática 1 2 3 4 5
d	faec	0 1 5	(0,1)	
f	aec	6 8	(0,1) (6,2)	d
a	ec	9 1	(0,1) (6,2) (9,3)	d f
e	c	2 3 1 5 7	(0,1) (6,2) (9,3) (3,4)	d f a
c		0 4	(0,1) (6,2) (9,3)	d f a e d f a S
\$		0 8 10 0 0	(0,1) (6,2)	d f a S c d f S d M S

Fig. 5.19

Um exercício interessante, que deixamos para o leitor, é a construção de árvores sintáticas a partir das informações da pilha sintática e da pilha do analisador.

5.8 TRATAMENTO AUTOMÁTICO DE ERROS SINTÁTICOS

Como vimos no item 2.2 o tratamento de erros sintáticos é uma parte fundamental do algoritmo de análise sintática. (Usaremos a denominação "tratamento" em lugar de "recuperação" como é por vezes empregada a tradução literal de "error recovery".) Trata-se, em síntese, de dois processos distintos: a *detecção* do erro e a *correção sintática* da cadeia de entrada, de modo que a análise sintática possa prosseguir até o fim dessa cadeia. Este último aspecto é essencial para os compiladores de uso prático: o usuário quer conhecer o maior número de erros que podem ser detectados no programa-fonte, para que ele possa posteriormente corrigi-los todos a um só tempo. Seria altamente indesejável se o compilador parasse a análise sintática no primeiro erro sintático detectado, ignorando o restante do programa.

Uma noção importante nessa área é a do *tratamento automático de erros sintáticos*. Essa expressão é usada quando os processos de tratamento independem do analisador, sendo dependentes exclusivamente da gramática analisada. Isto é, no caso do analisador ser fixo e receber uma gramática (ou tabela que a representa, como em nosso caso) como entrada não deve ser necessário reprogramar algumas de suas partes para tratar os erros de gramática diferentes. Assim, uma vez programado o processo de tratamento, ele se aplica a qualquer gramática aceita pelo analisador. Este será o caso dos algoritmos a serem descritos neste item.

5.8.1 DETECÇÃO DE ERROS SINTÁTICOS

Há duas possibilidades de ocorrerem erros sintáticos:

- a) O símbolo inicial da gramática foi encontrado, e ainda sobram símbolos da cadeia de entrada, ainda não analisados. Essa situação é deduzida pelo procedimento ANSIN nos últimos comandos deste: a pilha do analisador está vazia e não foi detectado um "fim de arquivo". Note-se que a pilha fica vazia somente quando o símbolo inicial é reconhecido, devido à inicialização do analisador, a qual coloca na pilha o apontador para o nó inicial contendo

esse símbolo (v. 5.6.4). O programa principal que chama ANSIN pode saber que esse erro ocorreu, pelo valor "false" do parâmetro de saída SUCESSO.

- b) O nó sendo visitado, de índice I_i , é do tipo terminal, não é λ -nó, mas seu conteúdo não coincide com o símbolo lido, e não há alternativa para esse nó, isto é:
 $TABGRAFO[I].TER = true, TABGRAFO[I].SIM \neq 0,$
 $TAB[TABGRAFO[I].SIM] \neq ENT$ e $TABGRAFO[I].ALT = 0$

Essa situação ocorre exatamente onde se dá a chamada de TRATAERRO(IU) no procedimento ANSIN. Vejamos em detalhes as ações que devem ser tomadas nesse caso.

Logo após a detecção de um erro sintático do tipo (b), é necessário emitir uma mensagem de erro, mostrando ao usuário exatamente em que símbolo t da cadeia de entrada deu-se o erro. Isso pode ser conseguido emitindo-se um caractere como ']' no registro de saída imediatamente abaixo do registro onde saiu t , e exatamente embaixo deste símbolo, como mostramos em exemplos no fim deste item. O AL tem um apontador para o registro de entrada, que pode ser usado para esse fim. Em seguida ao caractere ']' deve ser emitida, no mesmo registro, mensagem indicando qual o tipo de erro que ocorreu. Em nosso caso, podemos listar todos os símbolos terminais t_1, t_2, \dots, t_n esperados, com $n \geq 1$, nenhum dos quais coincide com t . Note-se que o nó com t_n não tem nó alternativo, como já foi dito; o nó de t_{n-1} pode ser um nó cuja alternativa é t_n ; eventualmente, algum t_i pode ter um não-terminal N como alternativa; neste caso t_{i+1} deve ser o primeiro terminal do subgrafo de N , com $1 \leq i \leq n$, se o primeiro nó de N não for não-terminal. É necessário percorrer a seqüência de alternativas t_1, t_2, \dots, t_i, N continuando com a seqüência $t_{i+1}, t_{i+2}, \dots, t_n$, emitindo todos os terminais que forem aparecendo. Evidentemente, $\{t_{i+1}, t_{i+2}, \dots, t_n\} = FIRST(N)$ (v. 5.5). Se algum desses terminais coincidisse com t , ou fosse λ , não teria ocorrido a condição de erro. Naturalmente, a segunda seqüência pode também ser interrompida por um não-terminal N' , e assim por diante. Conhecendo-se o nó inicial da primeira seqüência não é difícil percorrer as seqüências de alternativas, pulando para o primeiro nó de cada não-terminal que ocorrer no meio do percurso. Este se encerra com um nó terminal que não tem alternativa, e que será obrigatoriamente t_n . Denominaremos essa seqüência dos nós t_1, \dots, t_n de *percurso de alternativas*.

Formalmente podemos definir os símbolos terminais desse percurso para o sucessor de um nó m contendo o símbolo $x \in V_N \cup V_T$, da produção $M \rightarrow \alpha x \beta \in P$ $\alpha, \beta \in (V_N \cup V_T)^*$, como sendo o conjunto

$$PERALT(x, M \rightarrow \alpha x \beta) = \{t \in V_T \mid S \xrightarrow{*} \alpha' M \beta' \xrightarrow{*} \alpha' \alpha x \beta \beta' \xrightarrow{*} \alpha' \alpha x t \beta'' \beta', \alpha', \beta', \beta'' \in (V_N \cup V_T)^*\}$$

O procedimento ANSIN já foi programado guardando-se o índice IU do nó de t_1 , que é o nó inicial do percurso. Como se pode ver em ANSIN, IU assume um novo valor nos seguintes casos, pela ordem: primeiro nó do símbolo inicial da gramática; sucessor de um λ -nó; sucessor de um terminal que acabou de ser reconhecido; sucessor de um não-terminal que acabou de ser reconhecido. Damos a seguir o trecho do procedimento TRATAERRO que emite a mensagem dos terminais esperados e não encontrados. Suponhamos que o AL guarde na variável global IP a posição do primeiro caractere do símbolo t onde ocorreu o erro. Se no percurso de alternativas a partir de t_1 for encontrado um nó não-terminal, seu índice é desempilhado da pilha do analisador onde ele foi erroneamente colocado.

```

procedure TRATAERRO (IU: 1..MAXG);
var IX: 0..MAXG; /* percorre a lista de alternativas */
    IT: 1..80; /* contador para IP */
begin
    IX := IU; /* início do percurso */
    writeln('ERROR!'); /* muda de registro e emite início da mensagem */
    for IT := 1 to IP - 1 do write (' ');
    write ('[');
    while IX < 0 do /* percorre t_i */
        if TABGRAFO[IX].TER /* é terminal? */
            then begin /* emite terminal esperado */
                write ('"', TAB[TABGRAFO[IX].SIM], '"');
            end;
    write (']');
end;

```

```

    /* vai para a próxima alternativa */
    IX := TABGRAFO[IX].ALT
end
else begin /*nó não-terminal*/
    IX := TABNT[TABGRAFO[IX].SIM].PRIM;
    TOPO := TOPO - 1 /*desempilha um não-terminal*/
end;
write ('ESPERADO(S)');

```

Como exemplo, tomemos a gramática G_17 , do item 5.6.1 cujo grafo está na fig. 5.11. A cadeia 'aadfgecc' (o 'g' foi colocado a mais) provocaria a seguinte mensagem de erro:

```

A A D F G E C C
ERRO      |: 'A', 'D', 'E', ESPERADO(S)
No caso da cadeia de entrada 'ag' teríamos
A G
ERRO      |: 'B', 'A', 'D', 'E', ESPERADO(S)

```

Note-se que no exemplo seguinte a cadeia de entrada 'adfec', onde 'g' foi escrito em lugar de 'f', provocará a mensagem:

```

A D G E C
ERRO      |: 'C', ESPERADO(S)

```

Essa mensagem é devida ao fato de que 't' tem uma λ -alternativa, portanto M é sempre reconhecido quando procurado. Esse mesmo problema ocorre com o comando (errado) em PASCAL

IF A = B THEN THEN C:= D ELSE C:= E.

Neste caso, um comando vazio será reconhecido pelo analisador entre os doi then (v. grafo no apêndice I). Em seguida, será emitida a mensagem

|: ';', 'END', ESPERADO(S)

Se o erro for corrigido sintaticamente assumindo-se ';' em lugar do segundo THEN haverá detecção de novo erro no ELSE

|: ';', 'END' ESPERADO(S)

Vemos que o projeto de uma gramática deveria levar em conta o tratamento de erros feito pelo analisador sintático; de maneira geral, podemos recomendar que não se utilize λ -nós, alternativas vazias ou lados direitos consistindo exclusivamente da cadeia vazia. Um exemplo do primeiro caso é alternativa de ELSE e, do terceiro, o caso de "statement" vazio, ambos na gramática da linguagem PASCAL /J-W 74/ e conservados no grafo apresentado no apêndice I. No item 5.8.2e propomos uma detecção e correção de erros causados pelo reconhecimento indevido de alternativas vazias, como nos exemplos vistos acima.

5.8.2 CORREÇÃO SINTÁTICA DE ERROS

É importante notar o título deste item: a correção a ser tratada aqui é exclusivamente sintática. Supomos que ao encontrar o primeiro erro (sintático ou "semântico") o compilador deixe de gerar código-objeto. Tanto a análise sintática como a de contexto devem continuar, para que sejam detectados erros dessas naturezas no restante do texto, sendo o usuário informado dos mesmos.

Seja uma ESLL(1)-gramática $G = (V_N, V_T, S, P)$ e uma cadeia de entrada $e_1 \dots e_i e_{i+1} \dots e_n$. Suponhamos que a subcadeia $e_1 \dots e_{i-1}$ já tenha sido reconhecida pelo AS, e que houve uma detecção de erro (cf. 5.8.1.) no símbolo e_i . Lembremos que foi tentado o percurso de alternativas partindo de nó apontado por IU, passando pelos nós contendo os terminais t_1, \dots, t_n :

A correção será feita através de uma de quatro estratégias, que passaremos a descrever a seguir. Essas estratégias são testadas seqüencialmente; se nenhuma conseguir corrigir o erro, ignora-se o símbolo e_i e passa-se ao próximo símbolo e_{i+1} , voltando a testar as quatro es-