UMA LINGUAGEM DE ESPECIFICAÇÃO PARA A

GRAMÁTICA DE ATRIBUTOS UPED

Dora Toma Taguata

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.).

Aprovada por:

Estevam Gilberto de Simone
(Presidente)

José Lucas Mourão Rangel Netto

Dóris Ferraz de Aragon

RIO DE JANEIRO, RJ - BRASIL

JANEIRO DE 1982

TOMA, TAGUATA DORA

Uma Linguagem de Especificação para a Gramática de Atributos

UPED (Rio de Janeiro) 1982.

VII , 127 p. 29,7 cm (COPPE-UFRJ, M. Sc., Engenharia de Sistemas e Computação) 1982

Tese - Univ. Fed. Rio de Janeiro. Fac. de Engenharia

1. Análise Semântica

I. COPPE/UFRJ II. Título(Série)

AGRADECIMENTOS

Ao Professor Estevam Gilberto de Simone pela orientação e experiências transmitidas.

Ao Professor José Lucas Mourão Rangel Netto pelo interesse e sugestões proporcionadas.

Aos colegas Jorge Kawahara e Ana Soares pelas suas contribuições.

A Daisy pelo trabalho da datilografia.

Ao CNPq pelo auxílio financeiro.

RESUMO

Uma notação que pode ser usada para especificar semântica na definição de linguagens de programação e na especificação de entradas em construção automática de compiladores, é a gramática de atributos apresentada por Knuth.

Neste trabalho é estudada uma classe de gramáticas de atributos e um método de análise semântica baseado no método de Bochmann "Cálculo Semântico da Esquerda para a Direita".

Apresenta-se também uma linguagem para especificação da referida classe de gramáticas. Os tipos de dados da linguagem são tomados do PASCAL e de definição de dados estruturados de HOARE.

Mediante vários exemplos usando a linguagem de especificação, ilustra-se como é aplicada a gramática de atributos.

ABSTRACT

Knuth's attribute grammar may be used as a formal definition of programming languages semantics in a translator writing system.

We introduce a particular class of attribute grammars and a semantic evaluation method based on Bochmann's "Semantic Evaluation from left to right". Our method is tailored to produce efficient one-pass top down compilers.

It is also defined a language for attribute grammars description with data types similar to PASCAL and HOARE's structured data definition.

Several examples show the use of attribute grammars and it description language.

ÍNDICE

Pág.

Pág.

CAPÍTULO I

INTRODUÇÃO

O presente trabalho descreve gramáticas de atributos e sua aplicação na definição de linguagens de programação.

Uma notação formal é necessária à definição de uma linguagem (e do seu compilador em sistemas geradores de compiladores) para uma interpretação rigorosa.

O método mais usado na especificação de linguagens e no projeto de seu compilador é estabelecer-se sua definição da sintaxe e semântica. Isto se deve ao fato de que a sintaxe que descreve a estrutura da linguagem pode ser expressa mediante uma gramática livre de contexto, com as seguintes vantagens:

- Notação formal simples;
- Métodos de análise sintática eficientes;
- Uso para especificação de entrada em sistemas geradores de compiladores.

A semântica define o "significado" associado ao programa na tradução para linguagem objeto. Uma notação precisa que pode ser usada na especificação da semântica é a gramática de atributos.

O método de cálculo das funções semânticas é um algoritmo que realiza a análise semântica de uma determinada gramática de atributos.

Estudaremos aqui uma classe de gramática de atributos e sua aplicação na definição de linguagens. Esta classe será definida pelo método de cálculo UPED, baseado no método de Bochmann[3]: cálculo semântico de uma passagem da esquerda para a direita.

Apresenta-se no Capítulo II uma descrição de gramática formal, semântica e gramática de atributos; que são conceitos necessários para a especificação de linguagens de programação.

Descreve-se no Capítulo III, as características da gramática de atributos UPED, e o método de cálculo UPED para realizar a análise semântica desta classe de gramáticas.

No Capítulo IV define-se a linguagem de especificação (LEGA) para a gramática de atributos UPED.

O Capítulo V oferece exemplos de aplicação usando a linguagem LEGA. O capítulo VI apresenta as conclusões.

CAPÍTULO II

ESPECIFICAÇÃO DE LINGUAGENS

II.1 - ESPECIFICAÇÕES DE SINTAXE

Uma linguagem se define sintaticamente como um conjunto de sentenças formadas por elementos de um alfabeto.

Uma forma de sua representação finita são as gramáticas |AHO|[1,2], |HOPCROFT|[9].

II.1.1 - Gramática Formal

Uma gramática G se define por:

$$G = (N, \Sigma, P, Z)$$

onde:

N = conjunto finito de símbolos não terminais

Σ = conjunto finito de símbolos terminais

obedecendo a:

$$N \cap \Sigma = \phi$$

O vocabulário ou alfabeto se denota por V, tal que:

$$V = N \cup \Sigma$$

P = conjunto de produções da forma:

$$\alpha \rightarrow \beta \ / \ \alpha \in V^{+} \text{ e } \beta \in V^{*}$$

V^{*} - denota o conjunto de todas as cadeias possíveis de elementos de V, inclui a sequência vazia (£).

$$V^{+} = V^{*} - \{£\}$$

Z = símbolo inicial | $Z \in N$

A partir deste símbolo são geradas as cadeias da linguagem.

Definições

1. Derivação - se $\alpha \rightarrow \beta$ é uma produção de P e $\gamma, \delta \in V^{*}$, então dizemos que a cadeia $\gamma \alpha \delta$ deriva $\gamma \beta \delta$ e se denota por $\gamma \alpha \delta \underset{G}{\Rightarrow} \gamma \beta \delta$ quando é aplicada a produção $\alpha \rightarrow \beta$.

O símbolo $\overset{*}{\underset{G}{\Rightarrow}}$ representa derivações em zero ou mais passos. Isto é:

se $\alpha_1, \alpha_2, \ldots, \alpha_m \in V^*$, então:

$$\alpha_1 \overset{}{\underset{G}{\Rightarrow}} \alpha_2,\ \alpha_2 \overset{}{\underset{G}{\Rightarrow}} \alpha_3,\ \ldots,\ \alpha_{m-1} \overset{}{\underset{G}{\Rightarrow}} \alpha_m \equiv \alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_m$$

2. Forma sentencial - dizemos que α é uma forma sentencial quando:

$$Z \overset{*}{\underset{G}{\Rightarrow}} \alpha \quad \text{e} \quad \alpha \in V^*$$

3. Linguagem - a linguagem L, sendo gerada por uma gramática G se define por:

$$L(G) = \{w \in \Sigma^* / Z \overset{*}{\underset{G}{\Rightarrow}} w\}$$

II.1.2 - Tipos de Gramática

Chomsky |CRAIG|[5], |HOPCROFT|[9] definiu quatro tipos de gramática, denominadas tipo 0, 1, 2 e 3, e serão determinadas pela forma das produções.

Uma gramática tipo 0, 1, 2 ou 3 gera uma linguagem que será do tipo 0, 1, 2 ou 3, respectivamente.

1. Gramática tipo 0 - Denominada também gramática de estrutura de frase.

Uma produção se caracteriza por ser da forma:

$$\alpha \rightarrow \beta \mid \begin{matrix} \alpha \in V^+ \\ \beta \in V^* \end{matrix} \quad e$$

Esta é uma gramática sem maiores restrições sobre as produções.

2. Gramática tipo 1 - Ou gramática sensível ao contexto.

Suas produções são da forma:

$$\mu_1 A \mu_2 \rightarrow \mu_1 \beta \mu_2 \mid \begin{matrix} A \in N \\ B \in V^+ \\ \mu_1, \mu_2 \in V^* \end{matrix}$$

Na gramática, o contexto em torno da estrutura é expresso explicitamente.

3. Gramática tipo 2 - Ou gramática livre de contexto.

Suas produções são da forma

$$A \rightarrow \beta \mid \begin{matrix} A \in N \\ \beta \in V^* \end{matrix}$$

Exemplo:

Seja a linguagem $L = \{a^n b^n \mid n \geq 1\}$. Sua gramática pode ser expressa por

$$G = (N, \Sigma, P, Z) \mid$$

$N = \{Z\}$

$\Sigma = \{a,b\}$

$P = \{Z \rightarrow a\ Z\ b,\ Z \rightarrow a\ b\}$

Definição: Uma árvore de derivação é uma forma de representação gráfica das derivações por $w \in L(G)$, sendo G uma gramática livre de contexto.

Esta árvore tem as seguintes características:

- os nós são elementos de $N \cup \Sigma \cup \{£\}$
- à raiz é Z
- por um nó interno k existe uma produção $p \in P$ da forma:

$$X_o \rightarrow X_1\ X_2\ \ldots\ X_n$$

O nó k é rotulado X_o e seus filhos da esquerda para a direita são rotulados $X_1, X_2, \ldots, X_n$. No caso de ser n=0, k tem apenas um filho rotulado £.

4. Gramática tipo 3 - Ou gramática regular. Suas produções são da forma:

$$A \rightarrow a$$

ou

$$A \rightarrow Aa \mid a \in \Sigma,\ A \in N$$

Exemplo:

Seja a linguagem $L = \{a^n \mid n \geq 1\}$ uma gramática G que gera L é a seguinte:

$G = (N, \Sigma, P, Z)$ |

$N = \{Z\}$

$\Sigma = \{a\}$

$P = \{Z \rightarrow Z\ a\ ,\ Z \rightarrow a\}$

II.2 - SEMÂNTICA

A tradução de uma cadeia em linguagem fonte a uma cadeia em linguagem objeto, a qual expressa o seu "significado", é requerida em casos como conversão à forma interna e geração de código.

Denominamos semântica o conjunto de regras específicas da linguagem que determinam essa tradução. Classificamos a semântica em estática e dinâmica |CRAIG|[5].

II.2.1 - Semântica estática

Para descrever a semântica estática, define-se contexto como:

"A porção do programa em torno da ocorrência de um objeto capaz de permitir a compreensão de seu significado".

Exemplo:

Seja o programa ALGOL seguinte:

```
BEGIN
  INTEGER A, B, C
  REAL L, M, N
  -
  -
  A: = A + B;
  -
  -
END
```

No ALGOL, são permitidos diferentes tipos de dados (INTEGER, REAL, BOOLEAN, etc.) e operações mistas (soma de inteiros, reais, inteiro e real, etc.).

Na sentença "A: = A + B" a estrutura da expressão corresponde a uma soma entre os objetos identificados por A e B; seus tipos serão obtidos na parte de declarações e determinam uma soma de inteiros.

A semântica estática define as dependências ao contexto no programa. Isto é as informações adicionais obtidas

no próprio programa, para definir seu significado.

Em linguagens compiladas com FORTRAN, ALGOL, PL/I onde é importante a velocidade de execução, é conveniente precisarmos ao máximo seu significado na etapa de tradução, para diminuir os controles ou verificações durante a execução.

Alguns exemplos de dependência ao contexto são:

- Em ALGOL, uma chamada de procedimento ou função deve ter o número de parâmetros efetivos igual ao dos formais.

- Na declaração de um tipo "subrange" em PASCAL, o valor da primeira constante não deve ser maior ao da segunda.

Verificação estática de tipo e escopo estático [PRATT][17] são termos usados em linguagens de programação e formam parte da semântica estática.

1. Regra de escopo:

É uma regra que determina a porção do programa, onde uma referência a um identificador tem o mesmo significado.

Uma regra de escopo estático se determina sobre a estrutura do programa (durante o tempo de tradução).

Exemplo:

Uma linguagem ALGOL, é uma linguagem com estrutura de blocos.

Um bloco é constituído de um grupo de sentenças delimitadas por BEGIN, END. Uma regra de escopo na linguagem é:

"Na referência a identificador, sua definição é dada pela declaração que aparece no bloco mais interno encerrando o ponto de referência".

No seguinte programa ALGOL:

```
BEGIN                              % BLOCO 1
     INTEGER M;
     PROCEDURE PR(Z, A, B);
     REAL Z, A, B;
     BEGIN                         % BLOCO 2
           Z:= (A * B) ** M
     END;
     FILE IMPR (KIND = PRINTER);
     BEGIN                         % BLOCO 3
           REAL J, K, L;
           J:= 5.8;
           K:= 7.2;
           M:= M+1;
           PR (L, J, K);
           WRITE (IMPR, L, J, K, M)
     END;
END.
```

A referência a um identificador num bloco vai corresponder a seguinte informação:

BLOCO	IDENTIFICADOR	INFORMAÇÃO
2	Z, A, B	Classe variável, tipo REAL
	M	Classe variável, tipo INTEGER
3	J, K, L	Classe variável, tipo REAL
	M	Classe variável, tipo INTEGER
	PR	Classe PROCEDURE
		Parâmetros formais
		1º - Classe variável, tipo REAL
		2º - Classe variável, tipo REAL
		3º - Classe variável, tipo REAL

2. Verificação Estática de Tipo:

Esta verificação se faz sobre a estrutura do programa, para comprovar o uso dos tipos de dados na linguagem.

As linguagens que necessitam fazer verificações do tipo, requerem declarações (explícitas ou implícitas).

Exemplo:

No ALGOL, as operações hetereogêneas ou mistas permitem combinações de tipos de dados, assumindo-se conversões implícitas na tradução. Isto facilita ao usuário da linguagem a especificação da expressão.

No seguinte quadro é mostrado o tipo do resultado para alguns operadores e tipos de dados.

		TIPO DO RESULTADO	
TIPO DO OPERANDO		OPERADOR	
ESQUERDO	DIREITO	+ - *	/
INTEGER	INTEGER	INTEGER	INTEGER
INTEGER	REAL	REAL	REAL
REAL	INTEGER	REAL	REAL
REAL	REAL	REAL	REAL

Verificações de tipo serão requeridas para determinar o tipo do resultado na operação. Seja o seguinte programa ALGOL:

```
BEGIN
   REAL X;
   INTEGER E, F;
   -
   -
       F: = E + X;
   =
END.
```

Na expressão da sentença "F: = E + X" verificamos que os tipos de operandos da operação soma são:

Esquerdo: INTEGER

Direito : REAL

Isto fornece um resultado REAL segundo o quadro anterior.

Para uma chamada de procedimento ou função, numa linguagem tipo ALGOL, verificamos se o tipo dos parâmetros efetivos é o mesmo dos parâmetros formais.

Seja o programa ALGOL seguinte:

```
BEGIN
      PROCEDURE A(B, C);
      INTEGER B;
      REAL C;
           BEGIN
           -
           -
           END;
      INTEGER L;
      REAL M;
      -
      -
      A (L, M);
      -
      -
END.
```

No programa tem-se declarado um procedimento A com passagem de parâmetros por nome ("by name"). Na chamada A(L, M) verificamos se os parâmetros atuais L e M são de tipo INTEGER e REAL que correspondem aos tipos dos parâmetros formais B e C respectivamente.

II.2.2 - Semântica Dinâmica

Refere-se à definição do significado do programa, quando existe uma dependência aos valores dos dados (que serão obtidos apenas, em tempo de execução).

Para linguagens interpretadas como SNOBOL, APL, o tipo das variáveis é determinado durante a execução, sendo necessário realizar nesta etapa os procedimentos para definir o significado. Estes procedimentos são tais como verificações de tipo, interpretação das operações requeridas (soma de inteiros, reais, etc.).

Alguns exemplos de semântica dinâmica para linguagens compiladas são:

- Na referência de um ARRAY, os valores de subíndices devem estar dentro do intervalo, segundo sua declaração.

  Estes valores poderão ser especificados mediante uma expressão, que se calculará durante a execução.

- O valor atribuído a uma variável tipo INTEGER deve estar em intervalo pré-definido na implementação.

II.3 - ESPECIFICAÇÕES DE SEMÂNTICA

Os sistemas que descrevem linguagens de programação, isto é, especificação da linguagem e do compilador, geralmente expressam a sintaxe mediante uma gramática livre de contexto e a semântica mediante uma outra notação. O uso da gramática livre de contexto tem vantagens tais como:

- A notação de gramática livre de contexto facilita a compreensão e verificação da estrutura do programa.

- Os métodos para realizar análise sintática são definidos e eficientes para aplicação automática. Exemplos destes métodos são as análises sintáticas descendente ("top-down") e ascendente ("bottom-up").

- Geradores automáticos de analisadores sintáticos podem ser construídos.

A semântica não pode ser definida pela gramática livre de contexto, pois ela especifica apenas a estrutura da linguagem que independe do contexto. Então uma outra forma de especificação deve ser empregada.

Diversas notações tem sido usadas para definir a semântica, tais como:

- Linguagem Natural

- Modelo de máquina, que é capaz de definir as operações do programa fonte ou interpretar seu significado mediante as operações nesta máquina. Exemplo: rotinas semânticas acopladas na definição da sintaxe |GRIES|[6] , |GRIFFITHS|[7].

- Gramáticas Van Winjgaarden, uma notação formal usada para definir semântica estática, apresentada por Van Winjgaarden[20] para definir a linguagem ALGOL 68.

- Gramáticas de Atributos, definidas por Knuth[14] e que tem sido usadas como notação para vários sistemas geradores de compiladores |BOCHMANN|[4], |LORHO|[16].

II.3.1 - Método Informal: Rotinas Semânticas

O processo mais comum para implementação da análise semântica e traduções é realizado por execução de procedimentos que são denominados rotinas semânticas, durante a análise sintática.

As rotinas semânticas fazem uso de estruturas de dados para armazenar informação associada à forma do programa, necessária para realizar verificações semânticas e traduções.

A organização da rotina semântica depende do método usado na análise sintática. Para o método descendente e as-

cendente |AHO|[1], as rotinas semânticas podem ser consideradas durante a análise sintática como segue:

Método Descendente

As rotinas semânticas Y_1, Y_2, ..., Y_{n+1} podem ser intercaladas numa produção $X_o \rightarrow X_1 \ X_2 \ \ldots \ X_n$:

$$X_o \rightarrow \ Y_1 \ X_1 \ Y_2 \ X_2 \ \ldots \ X_n \ Y_{n+1}$$

A rotina semântica durante a análise vai ser executada quando sua referência aparece no topo da pilha sintática.

Uma outra forma de organizar as rotinas semânticas é considerando uma inclusão no começo e fim da produção:

$$X_o \rightarrow \ Y_1 \ X_1 \ X_2 \ \ldots \ X_n \ Y_2$$

Método Ascendente (LR)

As rotinas semânticas são indicadas nas entradas por AÇÃO na tabela de análise sintática.

Na implementação da análise sintática, as rotinas se executam quando se faz uma transição.

ESTADO	AÇÃO (Σ)				GO TO (N)		
	T_1	T_2	...	T_g	NT_1	NT_2...	NT_i
0							
1							
2							
:							
m							

NT: não terminal

T : terminal

Fig. II.1 - Tabela do analisador sintático LR

Uma entrada AÇÃO (A/B) na tabela do analisador sintático pode ser:

A = AÇÃO: SHIFT - ESTADO

REDUCE - PRODUÇÃO

ACCEPT

B = Rotina semântica

Para o caso de ser A = REDUCE, a rotina semântica é associada à produção aplicada na redução.

REDUÇÃO: PRODUÇÃO	ROTINA SEMÂNTICA
1	Y_1
2	Y_2
3	Y_3
:	:
N	Y_N

Exemplo:

Tradução de um número binário a um número decimal.

A tradução se faz usando as rotinas semânticas, que são executadas durante a análise sintática.

O exemplo é apresentado usando dois métodos de análise sintática.

1. Análise Sintática LL(1)

As rotinas semânticas são chamadas no momento em que o analisador sintático determina uma derivação.

No fim da análise, o número decimal se encontra numa variável RESULTADO.

a) Conjunto de produções da gramática LL(1) com as rotinas semânticas indicadas por um número no começo do lado direito da

produção.

```
0    Z   = NÚMERO
1    NÚMERO = (1) SINAL SEQUÊNCIA
2    SEQUÊNCIA = (2) DÍGITO SEQUÊNCIAB
3  - SEQUÊNCIAB  = SEQUÊNCIA
4                | (6)
5    DÍGITO      = (3) '1'
6                | (3) '0'
7    SINAL       = (4) ' +'
8                | (5)  '-'
```

b) Tabela preditiva

Não Terminal \ Terminal	1	0	+	-	$
Z			0	0	
NÚMERO			1	1	
SEQUÊNCIA	2	2			
SEQUÊNCIAB	3	3			4
DÍGITO	5	6			
SINAL			7	8	

c) Analisador Sintático e Tradutor.

```
%  PS = PILHA SINTÁTICA
%  SÍMBOLOS INICIAIS NA PS
   TOPO (PS): = $
   TOPO (PS): = Z
%  FIM = CONTROLE DO FIM DA ANÁLISE
   FIM: = FALSE
   GERA-TOKEN (TOKEN) % CONSEGUE TOKEN
   WHILE NOT FIM DO
   BEGIN
       SÍMBOLO (TOPO (PS), CLASSE)  % CONSEGUE INFORM. DE PS
       CASE  CLASSE  OF
       TERMINAL        : BEGIN
                         IF TOPO (PS) = $
                         THEN FIM: = TRUE
                         ELSE  BEGIN
                               POP (PS)
                               GERA-TOKEN (TOKEN)
                               END
                         END   TERMINAL
                         % TABELA-SINTAXE,CONSEGUE A PRODUÇÃO EM
                           TABELA PREDITIVA
        NÃO-TERMINAL   : BEGIN
                         % DERIVAÇÃO
                         POP (PS) % ELIMINA NÃO TERMINAL
                         % OBTER A PRODUÇÃO EM TABELA PREDITIVA
                         TABELA-SINTAXE (NÃO-TERMINAL, TOKEN,PRO
                         DUÇÃO)
                         FOR {CADA ELEMENTO EM PRODUÇÃO DA DIREI
                         TA PARA ESQUERDA} DO PUSH (PS,ELEMENTO);
                         END NÃO-TERMINAL
       AÇÃO-SEMÂNTICA : BEGIN
                         POP (PS) % ELIMINA NÚMERO DE ROTINA SEM.
                         ROTINA-SEMÂNTICA (AÇÃO-SEMÂNTICA)
                         END AÇÃO-SEMÂNTICA
   END  CASE
END     WHILE
```

```
% ROTINA DE TRATAMENTO DA TRADUÇÃO
PROCEDURE ROTINA-SEMÂNTICA (AÇÃO-SEMÂNTICA)
INTEGER AÇÃO-SEMÂNTICA
BEGIN  CASE AÇÃO-SEMÂNTICA OF
       1:   % INICIALIZA INDEX = POSIÇÃO DO ARRAY DE
            % DÍGITOS BINÁRIOS
            INDEX: = Ø
       2:   % PRESENÇA DE DÍGITO BINÁRIO
            INDEX: = INDEX + 1
       3:   % GUARDAR O DÍGITO BINÁRIO DE PS
            VALOR (INDEX): = TOPO (PS)
       4:   % NÚMERO BINÁRIO É POSITIVO
              SINAL: = FALSE
       5:   % NÚMERO BINÁRIO É NEGATIVO
              SINAL: = TRUE
       6:   % BEGIN CÁLCULO DO NÚMERO DECIMAL EM RESULTADO
              POTÊNCIA: = INDEX - 1
              RESULTADO:=0
              FOR    J:=1 STEP 1 UNTIL INDEX DO
                     BEGIN
                     RESULTADO:=RESULTADO+VALOR(J)*2**POTÊNCIA
                     POTÊNCIA:=POTÊNCIA - 1
                     END
            % ANALISAR SE POSITIVO OU NEGATIVO
            IF SINAL
            THEN RESULTADO: = - RESULTADO
            END
       END  CASE
END    ROTINA-SEMÂNTICA
```

d) Tradução da cadeia + 101

PILHA-SINTAXE	CADEIA DE ENTRADA	AÇÃO SEMÂNTICA
Z $	+ 101 $	
NÚMERO $	+ 101 $	
1 SINAL SEQUÊNCIA $	+ 101 $	
SINAL SEQUÊNCIA $	+ 101 $	INDEX = 0
4 + SEQUÊNCIA $	+ 101 $	
+ SEQUÊNCIA $	+ 101 $	SINAL = FALSE
SEQUÊNCIA $	101 $	
2 DÍGITO SEQUÊNCIAB $	101 $	
DÍGITO SEQUÊNCIAB $	101 $	INDEX = 1
3 '1' SEQUÊNCIAB $	101 $	
'1' SEQUÊNCIAB $	101 $	VALOR(1) = 1
SEQUÊNCIAB $	01 $	
SEQUÊNCIA $	01 $	
2 DÍGITO SEQUÊNCIAB $	01 $	
DÍGITO SEQUÊNCIAB $	01 $	INDEX = 2
3 '0' SEQUÊNCIAB $	01 $	
'0' SEQUÊNCIAB $	01 $	VALOR(2) = 0
SEQUÊNCIAB $	1 $	
SEQUÊNCIA $	1 $	
2 DÍGITO SEQUÊNCIAB $	1 $	
DÍGITO SEQUÊNCIAB $	1 $	INDEX = 3
3 '1' SEQUÊNCIAB $	1 $	
'1' SEQUÊNCIAB $	1 $	VALOR(3) = 1
SEQUÊNCIAB $	$	
6 $	$	
$	$	RESULTADO = 5

2. Análise Sintática SLR(1) |AHO|[1]

A chamada da rotina semântica é feita no momento que se tem uma redução na análise sintática. A rotina semântica é identificada pelo número de produção aplicado na redução.

No fim da análise, o número decimal se encontra na variável RESULTADO.

a) Produções da Gramática SLR

```
1   Z          = NÚMERO
2   NÚMERO     = S SEQUÊNCIA
3   SEQUÊNCIA  = DÍGITO SEQUÊNCIA
4              | DÍGITO
5   DÍGITO     = '1'
6              | '0'
7   S          = '+'
8              | '-'
```

b) Tabela do Analisador Sintático

ESTADO	AÇÃO					GO TO			
	1	0	+	-	$	NÚMERO	SEQUÊNCIA	DÍGITO	S
0			S3	S4		1			2
1					ACCEPT				
2	S7	S8					5	6	
3	R7	R7							
4	R8	R8							
5					R2				
6	S7	S8			R4		9	6	
7	R5	R5			R5				
8	R6	R6			R6				
9					R3				

Para AÇÃO uma entrada é: Si → S : SHIFT

i : ESTADO

Ri → R : REDUCE

i : PRODUÇÃO

ACCEPT - FIM DA ANÁLISE

PARA GO TO uma entrada é: i : ESTADO

c) Processo de Tradução

```
% PS = PILHA SINTÁTICA
% FIM = INDICADOR DO FIM DA ANÁLISE
  GERA - TOKEN (TOKEN) % GERA TOKEN
% ESTADO ZERO EM PS
PUSH (PS, 0)
FIM: = FALSE
WHILE NOT FIM DO
      BEGIN
      % ACTION: CONSEGUE O TIPO DE AÇÃO (ACCEPT, SHIFT,  REDUCE)
      % DE TABELA DO ANALISADOR SINTÁTICO
      ACTION (TOPO (PS), TOKEN, AÇÃO, ESTADO, PRODUÇÃO)
      % POR AÇÃO = SHIFT ENTÃO (SHIFT - ESTADO)
      % POR AÇÃO = REDUCE ENTÃO (REDUCE-PRODUÇÃO)
      CASE AÇÃO OF SHIFT: BEGIN
                PUSH (PS, ESTADO)
                GERA-TOKEN (TOKEN)
                END
      REDUCE   : BEGIN
                % X = Nº DE SÍMBOLOS DO LADO DIREITO DE PRODUÇÃO
                WHILE X > 0 DO BEGIN
                      POP (PS)
                      X: = X - 1
                      END
                % CONSEGUE O SEGUINTE ESTADO POR GO TO
                % EM TABELA DO ANALISADOR SINTÁTICO
                % SÍMBOLO ESQUERDO DE PRODUÇÃO = Y
                GO-TO (TOPO (PS), Y, ESTADO-SEGUINTE)
                PUSH (PS, ESTADO-SEGUINTE)
                ROTINA-SEMÂNTICA (PRODUÇÃO)
                END
      ACCEPT   : FIM: = TRUE
      END  CASE
END WHILE
```

```
% ROTINA DE TRATAMENTO DA TRADUÇÃO
PROCEDURE ROTINA-SEMÂNTICA (PRODUÇÃO)
INTEGER PRODUÇÃO
BEGIN  CASE  PRODUÇÃO OF
       1 :
       2 :   BEGIN
             % CÁLCULO DO NÚMERO DECIMAL EM RESULTADO
             RESULTADO: = 0
             POTÊNCIA: = INDEX-1
             FOR J:=1 STEP 1 UNTIL INDEX DO
                 BEGIN
                 RESULTADO:=RESULTADO+VALOR(J)*2**POTÊNCIA
                 POTÊNCIA: = * - 1
                 END
             IF SINAL
             THEN RESULTADO: = - RESULTADO
             END
       3,4:
       5 :   BEGIN
             % GUARDA O DÍGITO BINÁRIO : 1
             INDEX: = INDEX + 1
             VALOR (INDEX): = 1
             END
       6 :   BEGIN
             % GUARDA O DÍGITO BINÁRIO: 0
             INDEX: = INDEX + 1
             VALOR (INDEX): = 0
             END
       7 :   % INICIALIZA O INDICADOR DE POSIÇÃO DO ARRAY
             % QUE VAI CONTER OS DÍGITOS BINÁRIOS
             BEGIN
             INDEX: = 0
             SINAL: = FALSE
             END
       8 :   BEGIN
             INDEX:= 0
             SINAL:= TRUE
             END
       END CASE
END ROTINA-SEMÂNTICA
```

d) Exemplo de tradução da cadeia + 101

PILHA SINTAXE	ENTRADA	AÇÃO	AÇÃO SEMÂNTICA
0	+ 101 $	S3	-
3 0	101 $	R7	SINAL = FALSE INDEX = ∅
2 0	101 $	S7	
7 2 0	01 $	R5	INDEX = 1 VALOR(1) = 1
6 2 0	01 $	S8	
8 6 2 0	1 $	R6	INDEX = 2 VALOR(2) = 0
6 6 2 0	1 $	S7	
7 6 6 2 0	$	R5	INDEX = 3 VALOR(3) = 1
6 6 6 2 0	$	R4	
9 6 6 2 0	$	R3	
9 6 2 0	$	R3	
5 2 0	$	R2	RESULTADO = 5
1 0	$	ACCEPT	

II.3.2 - Método Formal: Gramáticas de Atributos

Gramática de atributos definida por Knuth[14] pode ser usada para especificar a semântica e tradução (forma interna, geração de código) de linguagens de programação.

II.3.2.1 - Definição

A gramática de atributos é constituida de uma gramática livre de contexto e um conjunto de atributos e funções

semânticas que definem a semântica.

1. Gramática livre de contexto

A gramática se define por G = (N, Σ, P, Z) como descrito em (II.1.2).

Uma produção p ε P se expressa por $X_o \rightarrow X_1 \; X_2 \; \ldots \; X_n \mid X_o \in N$ e $X_1, X_2, \ldots, X_n \in V$

2. Atributos

São informações associadas aos símbolos sintáticos.

Define-se um conjunto de atributos A para $X \in V$ por A(X) |

$$A(X) = I(X) \cup S(X)$$

onde:

I(X) = conjunto de atributos herdados de X

S(X) = conjunto de atributos sintetizados de X

e $I(X) \cap S(X) = \emptyset$

para o símbolo inicial Z e terminais se tem a condição:

$I(Z) = \phi$

$S(Z) = \phi$ para $X \in \Sigma$

Neste trabalho se segue a definição dada por Bochmann[3], onde a segunda condição é modificada por:

$I(X) = \phi$ para $X \in \Sigma$

Os atributos herdados, são transmitidos no sentido da raiz da árvore de derivação, para os nós terminais.

Os atributos sintetizados, são transmitidos no sentido contrário aos herdados.

Isto significa que os atributos herdados expressam uma dependência do significado de um certo símbolo aos símbolos ascendentes na árvore de derivação. Os atributos sintetizados exprimem o oposto.

3. Funções Semânticas

Estas funções expressam relações entre os valores de atributos associados aos símbolos sintáticos numa produção da gramática.

Uma ocorrência de atributos se denota por (a,j), e representa o valor de um atributo do tipo a, que corresponde ao símbolo X_j, $j = 0, 1, \ldots, n$, na produção $p \in P$, sempre que $a \in A(X_j)$. Uma ocorrência de atributos é especificada por uma função semântica, denotada por $f^p(a,j)$. Numa produção $X_o \rightarrow X_1 X_2 \ldots X_n$, uma função semântica $f^p(a,j)$ é especificada para definir uma ocorrência de atributos:

- herdado, quando $j = 1, 2, \ldots, n$ e $\underline{a} \in I(X_j)$

- sintetizado, quando $j = 0$ e $\underline{a} \in S(X_j)$

A função semântica $f^p(a,j)$, mapeia certas ocorrências de atributos associados aos símbolos sintáticos da produção $p \in P$ pela ocorrência de (a, j).

$$f^p(a,j)((a_1,j_1), (a_2,j_2),\ldots,(a_m,j_m)) = (a,j)$$

$(a_k, j_k)/k = 1, 2, \ldots, m$, são ocorrências de atributos para símbolos sintáticos X_{jk} na produção. A função semântica é definida para ter um valor de atributo $\underline{a}$ associado a X_j em p, se todas as ocorrências de atributos (argumentos da função) usadas para calcular $f^p(a,j)$ tiverem sido previamente calculadas.

II.3.2.2 - Processo de Cálculo das Funções Semânticas

Para realizar a análise de uma cadeia na linguagem, as funções semânticas são calculadas sobre a árvore de derivação que gera tal cadeia.

O processo de cálculo das funções semânticas consiste em, inicialmente, se obter uma ordem para seu cálculo de modo que cada função semântica seja definida.

Isto significa que, para que possamos calcular todas as funções semânticas poderá ser necessário percorrermos a

árvore diversas vezes.

A forma de evitarmos esse problema é definirmos um algoritmo de percurso da árvore e estabelecermos um conjunto de restrições às funções semânticas, de modo que seu cálculo seja sempre possível.

Numa gramática de atributos sem restrições, quanto a forma das funções semânticas é possível que nem todos os valores de atributos possam ser sempre definidos para todos os nós, em qualquer árvore de derivação. Este problema tem sido objeto de diferentes estudos |KNUTH|[14] , e é conhecido como o problema de circularidade em gramática de atributos.

Existem métodos de cálculo que determinam a ordem em que são calculadas as funções semânticas sobre a árvore de derivação. Estes métodos estabelecem o caminho a seguir na árvore para visitar os nós, e a seleção das funções semânticas que devem ser definidas.

Entre os métodos de cálculo estão:

Alternating Semantic |JAZAYERI|[10]
Treewalk Evaluator |KENNEDY|[13]
Ordered Attributed Grammar |KASTENS|[12]
Semantic Evaluation from Left to Right |BOCHMANN|[3]

O calculador é um programa que faz o processo de cálculo das funções semânticas, segundo o algoritmo dado por um dos métodos de cálculo.

II.3.2.3 - Método de Cálculo das Funções Semânticas da Esquerda para a Direita

O método, definido por Bochmann[3], consiste em percorrer a árvore de derivação desde a raiz até as saídas, visitando os nós no sentido da esquerda para a direita.

Dependendo da definição das funções semânticas, pode ser necessário realizar várias passagens sobre a árvore de derivação, até que todas as funções fiquem definidas.

A gramática de atributos não deve ser circular.

1. Método de cálculo das funções semânticas com uma passagem da esquerda para a direita

O método define o algoritmo de percurso sobre a árvore de derivação, determinando a sequência em que os nós devem ser visitados, e as funções semânticas que devem ser calculadas durante as visitas dos nós.

As funções semânticas são especificadas com base nas ocorrências dos atributos, que são previamente definidas no percurso da árvore, determinado pelo algoritmo.

A descrição do método consiste na definição do algoritmo de percurso sobre a árvore de derivação, e nas condições para escrever as funções semânticas.

a) Algoritmo de percurso sobre a árvore de derivação

Bochmann apresenta um procedimento recursivo cálculo-subárvore, para realizar o percurso sobre a árvore de derivação.

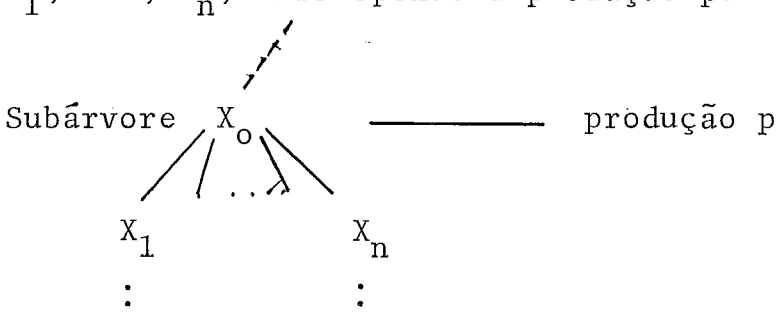
O percurso deve começar pela raiz

Procedimento cálculo-subárvore

Formato: Cálculo-subárvore (A)

A é um nó da árvore

Um nó da árvore de derivação, usado como argumento na chamada do procedimento, é denominado X_o, e seus filhos X_1, ..., X_n, corresponde à produção p.



```
PROCEDURE CÁLCULO-SUBÁRVORE (NÓ)      % X_o
BEGIN
FOR K:=1 TO    N  DO
    IF X_k  É  NÃO TERMINAL
       THEN    BEGIN
               CALCULAR AS FUNÇÕES SEMÂNTICAS DE ATRIBUTOS HERDA-
               DOS DE X_k ESPECIFICADOS EM p;
               CÁLCULO-SUBÁRVORE (X_k)
               END
       ELSE    % TERMINAL OU & % CALCULAR AS FUNÇÕES DE ATRIBUTOS
               SINTETIZADOS DE X_k ESPECIFICADOS EM p;
CALCULAR AS FUNÇÕES SEMÂNTICAS DE ATRIBUTOS SINTETIZADOS DE  NÓ
ESPECIFICADOS EM p.
END
```

b) Condições das funções semânticas

Definição: O conjunto de dependências $D^p(a,j)$ denota um conjunto que contém as ocorrências dos atributos, que serão empregados na avaliação de (a,j) pela função semântica $f^p(a,j)$.

As condições para escrever as funções semânticas são dadas pelo seguinte teorema.

Teorema:

Dada uma gramática de atributos que satisfaz:

$$D^p(a,j) \subset I_o \cup \bigcup_{j'=1}^{n} S_{j'},$$

Para: $j=0$ e $a \in S(X_o)$,

assim como $j = 1, 2, \ldots, n$ e $a \in I(X_j)$

onde: I_o = conjunto de ocorrências de atributos herdados de X_o em p

S_j = conjunto de ocorrências de atributos sintetizados de X_j em p

As funções semânticas em qualquer árvore de derivação, podem ser calculadas numa simples passagem da esquerda para direita, se e somente se se cumpre o seguinte:

$$D^p(i,j) \cap \bigcup_{j'=j}^{n} S_{j'} = \phi$$

Para todo $j = 1, 2, \ldots, n$ e $i \in I(X_j)$

2. Exemplo

Apresentamos o algoritmo Sethi Ullman[19], para obter o número de registros que demanda o cálculo de uma expressão.

As intruções são feitas com o operando do lado esquerdo num registro de uso geral.

O método de cálculo das funções semânticas demanda só uma passagem sobre a árvore de derivação.

a) Atributos

Posição - É um atributo herdado que dá informação sobre a posição do operando em relação ao operador. Seu tipo é "Boolean".

Número - É um atributo sintetizado que dá o número de registros necessários, para o cálculo das operações na expressão. Seu tipo é INTEIRO.

b) Símbolos sintáticos e seus atributos associados

SÍMBOLO	ATRIBUTO
Z	NÚMERO
E	NÚMERO, POSIÇÃO
T	NÚMERO, POSIÇÃO
F	NÚMERO, POSIÇÃO
PM, MD	-

C) Produções e funções semânticas

Quando numa produção aparece o mesmo símbolo sintático mais de uma vez, coloca-se um subíndice.

	PRODUÇÃO	FUNÇÃO SEMÂNTICA
(1)	$Z \rightarrow E$	NÚMERO (Z): = NÚMERO (E); POSIÇÃO (E): = TRUE;
(2)	$E_1 \rightarrow E_2$ PM T	IF NÚMERO (E_2) = NÚMERO (T) THEN NÚMERO (E_1): = NÚMERO(E_2) + 1 ELSE IF NÚMERO (E_2) > NÚMERO (T) THEN NÚMERO (E_1): = NÚMERO (E_2) ELSE NÚMERO (E_1): = NÚMERO (T); POSIÇÃO (E_2): = TRUE; POSIÇÃO (T): = FALSE;
(3)	$E \rightarrow T$	NÚMERO (E): = NÚMERO (T); POSIÇÃO (T): = POSIÇÃO (E);

	PRODUÇÃO	FUNÇÃO SEMÂNTICA
(4)	$T_1 \rightarrow T_2$ MD F	IF NÚMERO (T_2) = NÚMERO (F) THEN NÚMERO (T_1): NÚMERO (T_2) + 1 ELSE IF NÚMERO (T_2) > NÚMERO (F) THEN NÚMERO (T_1): = NÚMERO (T_2) ELSE NÚMERO (T_1): = NÚMERO (F); POSIÇÃO (T_2): = TRUE; POSIÇÃO (F) : = FALSE;
(5)	T → F	NÚMERO (T): = NÚMERO (F); POSIÇÃO (F): = POSIÇÃO (T);
(6)	F → 'IDENT'	IF POSIÇÃO (F) THEN NÚMERO (F): = 1 ELSE NÚMERO (F): = 0;
(7)	F → (E)	NÚMERO (F): = NÚMERO (E); POSIÇÃO (E): = POSIÇÃO (F);
(8)	PM → '+'	
(9)	PM → '-'	
(10)	MD → '*'	
(11)	MD → '/'	

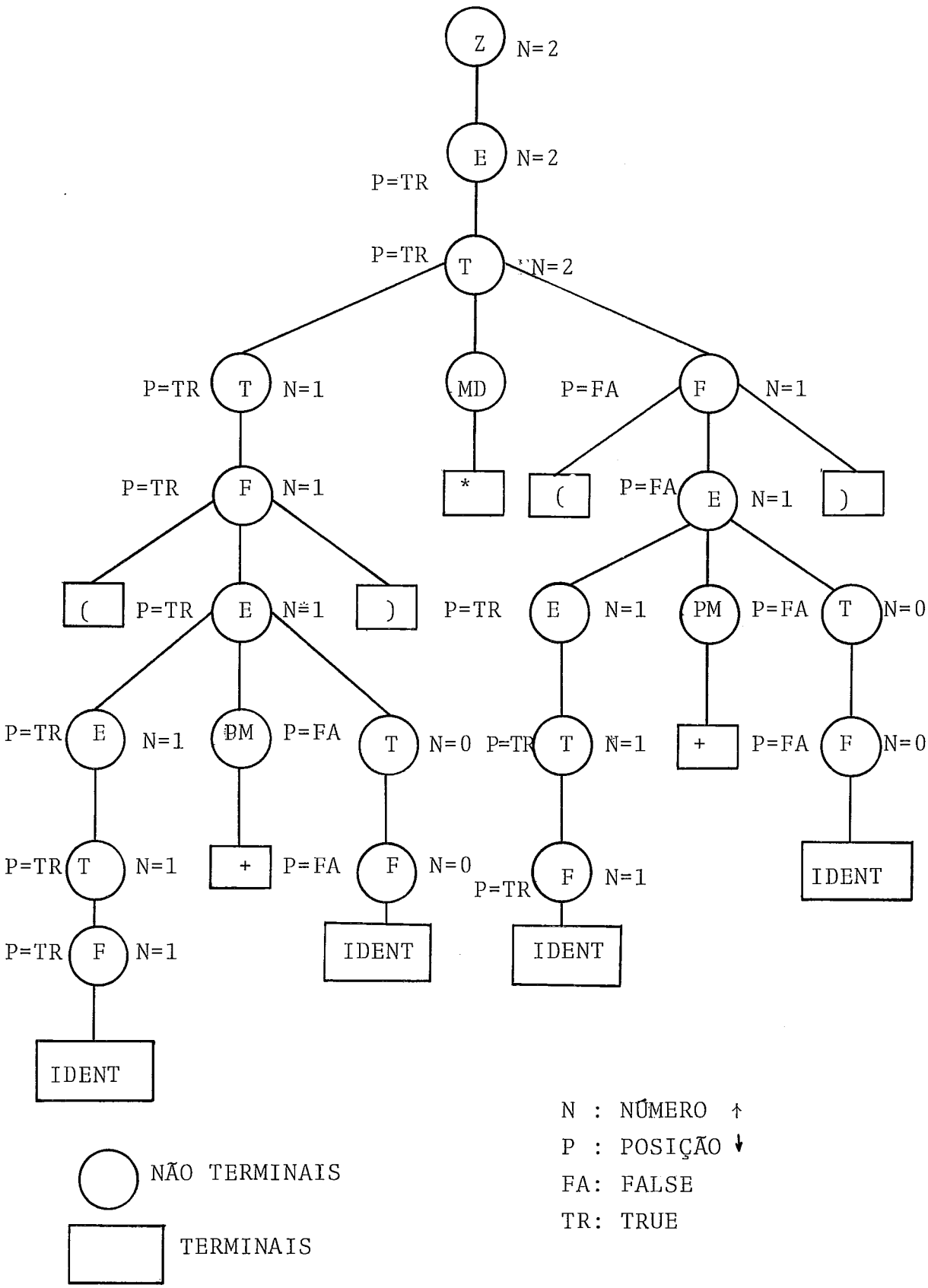
d) Exemplo de cálculo das funções semânticas para uma expressão

Seja a expressão (i+i) * (i+i), mostramos na Fig. III.1 a árvore de derivação com as funções semânticas definidas.

O número necessário de registradores máximo é 2, obtido por NÚMERO associado a Z.

Note que estamos usando diretamente a árvore de derivação, não nos importando com o tipo do analisador sintático que a construiu. No caso como a gramática tem produções recursivas à esquerda, deverá ser necessariamente ascendente.

Fig. II.2 - Árvore de derivação com as funções definidas para a expressão (i+1) * (i+i)



CAPÍTULO III

A GRAMÁTICA DE ATRIBUTOS UPED

Como vimos no capítulo anterior o método informal de efetuar a análise semântica é o uso de subprogramas escritos individualmente e que são chamados por controle do analisador sintático. Evidentemente deixa muito a desejar quanto a confiabilidade e facilidade de programação. Entretanto é o método correntemente ainda em utilização.

Gramáticas de atributos, que se constituem um modelo para análise semântica e tradução, cada vez são mais utilizadas para a leitura de compiladores. Entretanto, a complexidade da implementação de seu calculador torna seu uso, por vezes, inviável na construção de compiladores.

Neste trabalho iremos propor uma classe de gramáticas de atributos cujo método de cálculo será ainda mais simplificado que o de Bochmann[3], e nos preocuparemos com sua implementação, em particular seu casamento com um analisador sintático descendente, de modo a evitar a construção da árvore sintática e permitir sua aplicação em compiladores de um só passo.

A classe de gramática de atributos UPED é caracterizada pela forma em que se expressam as funções semânticas.

Algumas condições são estabelecidas para escrever as funções, devido à disponibilidade dos valores de atributos (argumentos) associados aos símbolos sintáticos na produção.

A análise semântica pode ser realizada pelo método de cálculo das funções semânticas UPED, que estabelece o algoritmo de percurso sobre a árvore de derivação e a definição das funções semânticas.

III.1 - CONDIÇÕES DAS FUNÇÕES SEMÂNTICAS NA GRAMÁTICA DE ATRIBUTOS UPED

Uma função semântica expressa o valor do atributo associado ao símbolo sintático numa produção.

As condições para escrever uma função semântica numa produção $X_o \rightarrow X_1 \ X_2 \ \ldots \ X_n$ são:

1. Uma função semântica correspondente a X_o, pode ser especificada com:

   - Valores de atributos sintetizados de X_j, $(j = 1,2,\ldots,n)$

   - Valores de atributos herdados de X_o.

   Uma função semântica vai definir o valor do atributo sintetizado associado a X_o.

2. Uma função semântica correspondente a X_j, (j = 1,2,...,n) pode ser especificado com:

   - Valores de atributos herdados de X_0

   - Valores de atributos sintetizados de $X_i \mid 0 < i < j$.

   Uma função semântica vai definir o valor do atributo herdado associado a X_j.

Esta proposta é uma simplificação do método de Bochmann, de modo que:

a) ao visitarmos pela primeira vez um certo nó (descendo), podemos apenas transmitir seus atributos herdados aos seus filhos;

b) ao visitarmos pela segunda vez um certo nó (subindo), podemos apenas transmitir os atributos sintetizados de seus filhos e os seus próprios atributos herdados.

Neste método todos os nós são visitados exatamente duas vezes.

III.2 - O MÉTODO DE CÁLCULO UPED

O método de cálculo das funções semânticas UPED, realiza a análise semântica de uma gramática de atributos UPED, com as seguintes vantagens:

- Eficiência e simplicidade
- Pode ser aplicado simultaneamente à análise sintática
- A árvore de derivação é mantida parcialmente, conseguindo menor uso de espaço de memória.

O algoritmo é baseado no método de cálculo de Bochmann[3] de uma simples passagem da esquerda para a direita sobre a árvore de derivação.

A árvore é criada na mesma ordem em que é feita na análise sintática descendente |AHO|[1], e durante sua formação se faz paralelamente o percurso para o cálculo das funções semânticas.

A informação de entrada é a cadeia da linguagem que deve ser analisada, e a sequência de produções obtida nas derivações mais a esquerda.

A saída são os valores de atributos definidos para o nó raiz.

Os valores dos atributos que são obtidos durante o procedimento de cálculo podem ser organizados em pilhas.

III.2.1 - Informação de Entrada

1. Cadeia da linguagem

É aquela que será analisada.

2. Sequência de produções

É a sequência correspondente às derivações mais a esquerda, que gera a cadeia analisada. As produções usadas são as especificadas na gramática de atributos.

O processo de derivação mais à esquerda começa pelo símbolo inicial Z; a cada derivação i recoloca-se o símbolo não terminal mais à esquerda (m.e.) na forma sentencial α_i, pela parte direita da produção aplicada. Obtemos a sequência de símbolos terminais w.

Desta forma, teríamos:

$$Z = \alpha_1 \overset{\text{m.e}}{\Rightarrow} \alpha_2 \overset{\text{m.e.}}{\Rightarrow} \cdots \overset{\text{m.e.}}{\Rightarrow} \alpha_n = w$$

III.2.2 - Informação de Saída

O método de cálculo finaliza com a segunda visita ao nó raiz, sendo disponíveis os valores dos atributos sintetizados associados ao símbolo inicial.

III.2.3 - Organização dos Valores de Atributos

Os valores de atributos são obtidos pela definição das funções semânticas sobre a árvore de derivação.

Usa-se uma estrutura de pilha por tipo de atributo e símbolo sintático, para organizar os valores de atributos.

No cálculo de uma função semântica, um valor de atributo, que é um argumento, encontra-se no topo da pilha correspondente.

Algumas restrições e observações, serão feitas para obter e armazenar informações nas pilhas de atributos durante o processo de cálculo das funções semânticas.

III.2.4 - Procedimento de Cálculo

Se da gramática livre de contexto pode-se obter um analisador sintático descendente determinístico, o processo de cálculo das funções semânticas pode ser feito conjuntamente com a análise sintática.

No caso de recuperação de erro deve ser verificado se não irá afetar o processo de cálculo, isto por que as funções semânticas ficam definidas em cada passo da análise sintática. O tratamento de recuperação de erro na análise semântica não será considerado no presente trabalho.

São usadas duas pilhas no processo de percurso sobre a árvore de derivação.

1. Pilha Sintática (PS)

Uma entrada pode ser:

- Um nó da árvore de derivação representado por um símbolo sintático.

- Um sinal (#) que identifica o final de uma produção.

- Um sinal ($) de fim da cadeia de entrada.

2. Pilha de Produções (PP)

Uma entrada contem um nó da árvore de derivação representado pelo número de produção associada. Esta informação vai ser usada no processo de subida na árvore de derivação.

No diagrama da Fig. III.1 mostra-se os passos que se seguem para o cálculo das funções semânticas sobre a árvore de derivação.

Antes disto, faremos observações sobre a organização das pilhas de atributos no cálculo das funções semânticas, segundo o sentido de visita dos nós na árvore durante o processo de cálculo.

OBSERVAÇÕES:

Considerar a produção $X_o \rightarrow X_1 \ X_2 \ \ldots \ X_n$

OBS. 1: No cálculo de uma função semântica f que define um valor de atributo sintetizado S associado ao símbolo terminal $X_i | 1 \leq i \leq n$ ou um valor de atributo herdado I associado ao não terminal $X_i | 1 \leq i \leq n$; o argumento de f que é um valor de atributo sintetizado associado a $X_j \mid 0 < j < i$ se encontra na pilha respectiva, numa profundidade determinada pela correspondência de símbolos iguais X_j da direita para a esquerda anteriores a X_i na produção.

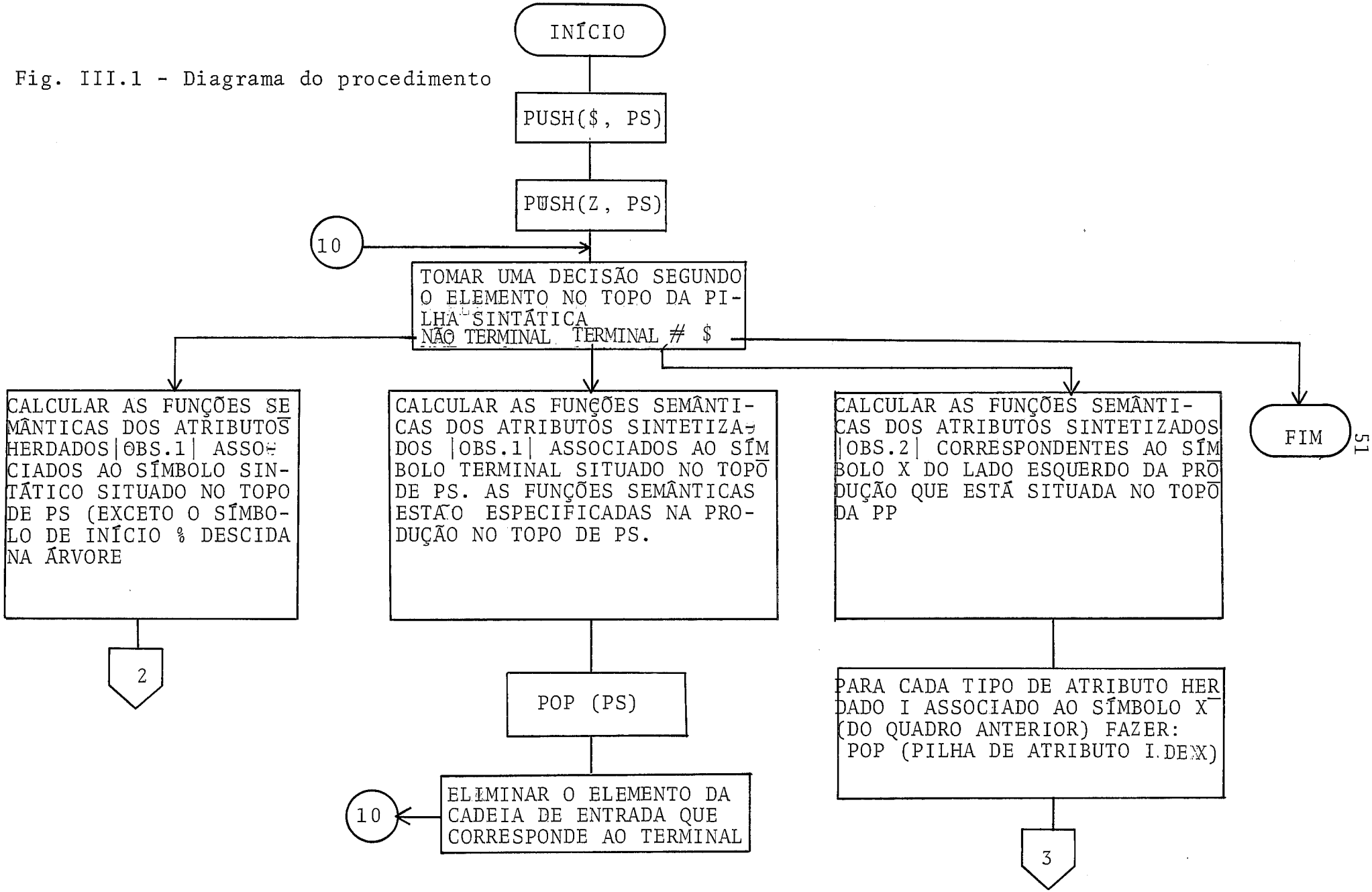
O valor de atributo definido pela função f se coloca no topo da pilha correspondente.

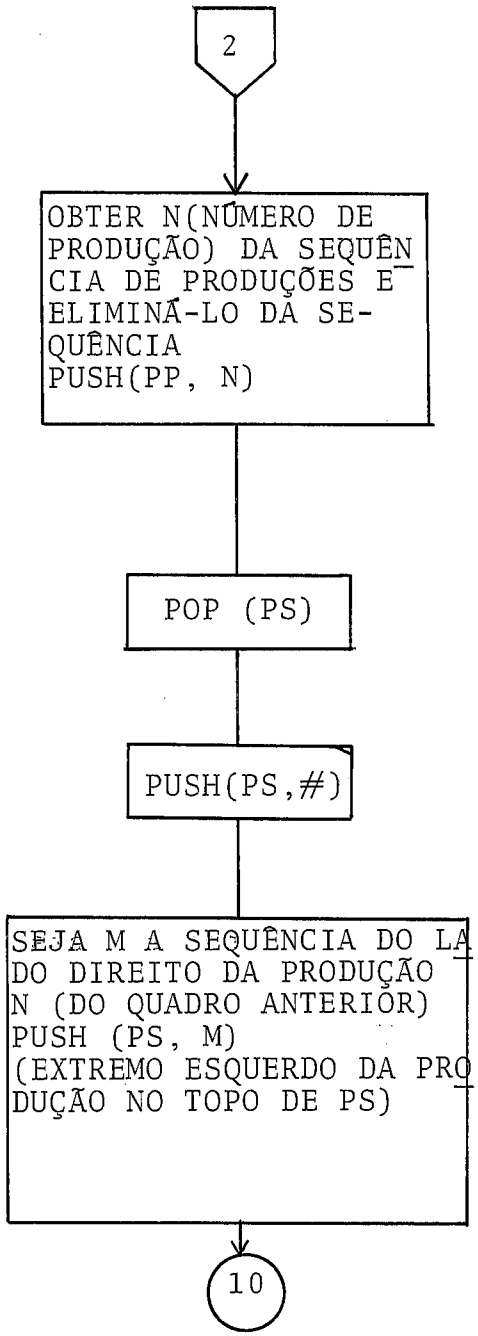
OBS. 2: No cálculo de uma função semântica f que define o valor do atributo sintetizado associado a X_o; um argumento de f que é um valor de atributo sintetizado associado a $X_j | 1 \leq j \leq n$ se encontra na pilha respectiva, numa profundidade determinada pela correspondência de símbolos iguais a X_j de direita para esquerda na produção.

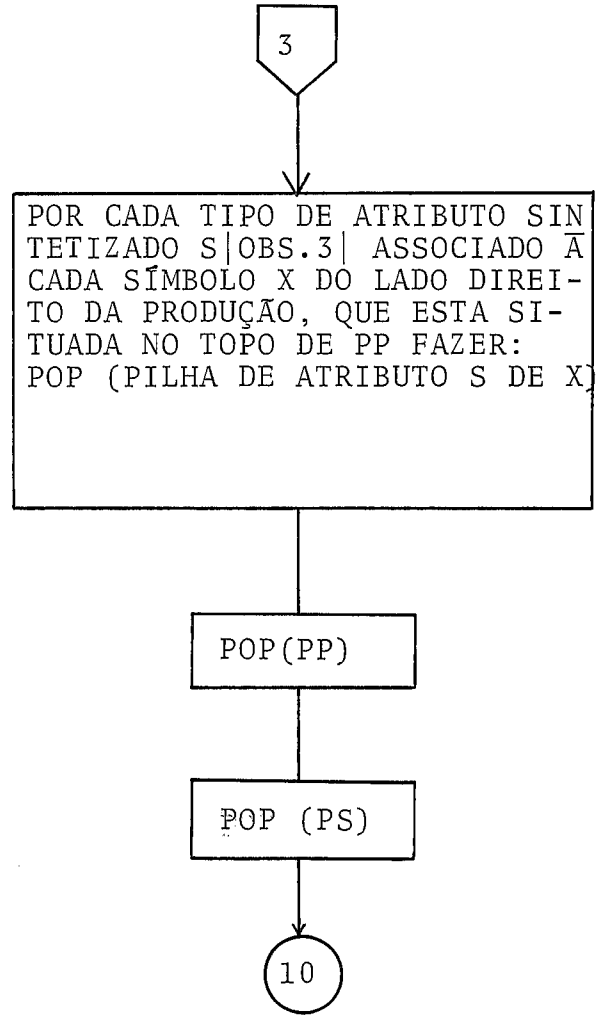
O valor de um atributo sintetizado definido pela função semântica pode ser armazenado num "buffer", até realizar todas as atualizações nas pilhas de atributos segundo OBS. 3.

OBS. 3: A atualização das pilhas de atributos sintetizados para os símbolos $X_j \mid (j = 1, 2, \ldots, n)$, no momento de visita de X_o no processo de subida, começa-se pelas pilhas de atributos dos símbolos de direita para esquerda na produção. Isto é X_n, X_{n-1}, $\ldots$, X_1.

Fig. III.1 - Diagrama do procedimento




2
OBTER N(NÚMERO DE PRODUÇÃO) DA SEQUÊNCIA DE PRODUÇÕES E ELIMINÁ-LO DA SEQUÊNCIA
PUSH(PP, N)
POP (PS)
PUSH(PS,#)
SEJA M A SEQUÊNCIA DO LADO DIREITO DA PRODUÇÃO N (DO QUADRO ANTERIOR)
PUSH (PS, M)
(EXTREMO ESQUERDO DA PRODUÇÃO NO TOPO DE PS)
10


3
POR CADA TIPO DE ATRIBUTO SINTETIZADO S|OBS.3| ASSOCIADO A CADA SÍMBOLO X DO LADO DIREITO DA PRODUÇÃO, QUE ESTA SITUADA NO TOPO DE PP FAZER:
POP (PILHA DE ATRIBUTO S DE X)
POP(PP)
POP (PS)
10

EXEMPLO:

Para o exemplo II.3.1, conversão de um número binário a um número decimal, se aplica o método de cálculo para o número binário + 101.

a) Gramática de atributos

- Atributos

Condição - atributo sintetizado que determina o sinal do número binário.

Valor - atributo sintetizado que contém o valor decimal.

Potência - atributo sintetizado que dá o peso para o dígito binário.

- Símbolos sintáticos e seus atributos associados:

Número	: Valor
Sinal	: Condição
Sequência, SequênciaB	: Valor, Potência
Dígito	: Valor

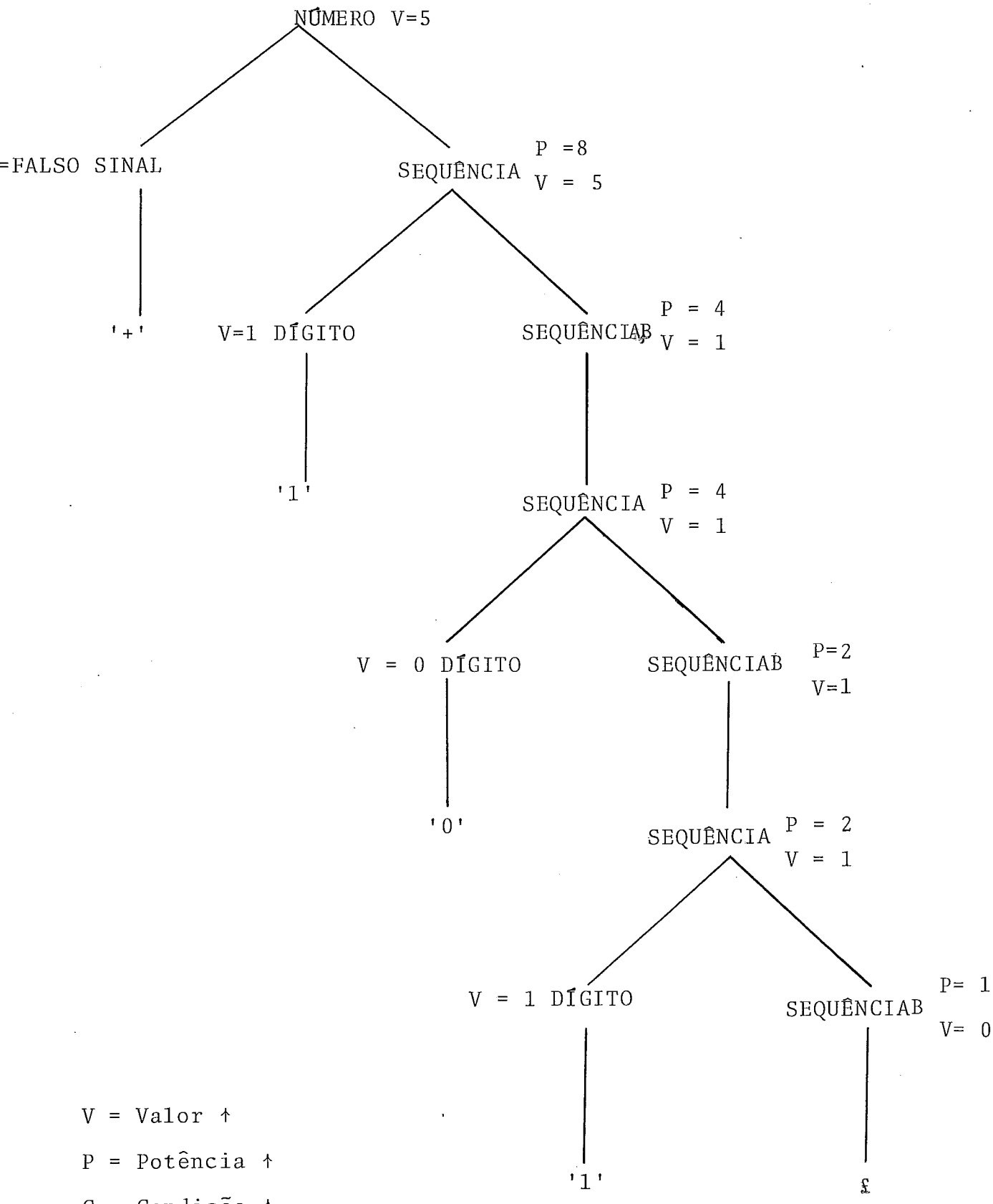
- A gramática:

PRODUÇÕES	FUNÇÕES SEMÂNTICAS
1 Número → Sinal Sequência	If Sinal. Condição Then Número..Valor: = - Sequência.Valor Else Número. Valor: = Sequência.Valor
2 Sinal → '+'	Sinal.Condição: = False
3 Sinal → '-'	Sinal.Condição: = True
4 Sequência → Dígito SequênciaB	Sequência.Valor:=SequênciaB.Potência * Dígito.Valor + SequênciaB .Valor Sequência.Potência:=SequênciaB .Potência * 2
5 SequênciaB → Sequência	SequênciaB .Potência:=Sequência.Potência SequênciaB .Valor:=Sequência.Valor
6 SequênciaB → £	SequênciaB .Potência:=1 SequênciaB .Valor :=0

7 Dígito → '0' Dígito.Valor: = 0

8 Dígito → '1' Dígito.Valor: = 1

b) Árvore de derivação com seus atributos, para a cadeia +101



V = Valor ↑

P = Potência ↑

C = Condição ↑

c) Procedimento de cálculo da cadeia + 101

PASSO	PP	PS	CADEIA	SEQUÊNCIA-PRODUÇÕES
0	-	N	$ + 1 0 1 $	1 2 4 8 5 4 7 5 4 8 6
1	1	S SA #	$ + 1 0 1 $	2 4 8 5 4 7 5 4 8 6
2	2 1	'+' # SA #	$ + 1 0 1 $	4 8 5 4 7 5 4 8 6
3	2 1	# SA #	$ 1 0 1 $	4 8 5 4 7 5 4 8 6
4	1	SA #	$ 1 0 1 $	4 8 5 4 7 5 4 8 6
5	4 1	D SB # #	$ 1 0 1 $	8 5 4 7 5 4 8 6
6	8 4 1	'1' # SB # #	$ 1 0 1 $	5 4 7 5 4 8 6
7	8 4 1	# SB # #	$ 0 1 $	5 4 7 5 4 8 6
8	4 1	SB # #	$ 0 1 $	5 4 7 5 4 8 6
9	5 4 1	SA # # #	$ 0 1 $	4 7 5 4 8 6
10	4 5 4 1	D SB # # # #	$ 0 1 $	7 5 4 8 6
11	7 4 5 4 1	'0' # SB # # # #	$ 0 1 $	5 4 8 6
12	7 4 5 4 1	# SB # # # #	$ 1 $	5 4 8 6
13	4 5 4 1	SB # # # #	$ 1 $	5 4 8 6
14	5 4 5 4 1	SA # # # # #	$ 1 $	4 8 6
15	4 5 4 5 4 1	D SB # # # # # #	$ 1 $	8 6
16	8 4 5 4 5 4 1	'1' # SB # # # # # #	$ 1 $	6
17	8 4 5 4 5 4 1	# SB # # # # # #	$ $	6
18	4 5 4 5 4 1	SB # # # # # #	$ $	6
19	6 4 5 4 5 4 1	# # # # # # #	$ $	
20	4 5 4 5 4 1	# # # # # #	$ $	
21	5 4 5 4 1	# # # # #	$ $	
22	4 5 4 1	# # # #	$ $	
23	5 4 1	# # #	$ $	
24	4 1	# #	$ $	
25	1	#	$ $	
26				

Informação das Pilhas de Atributos

PASSO	ATRIBUTO						
Nº	S.C.	N.V	SA.P	SA.V	SB.P	SB.V	D.V.
3	False						
8	False						1
12	False						0 1
17	False						1 0 1
19	False				1	0	1 0 1
20	False		2	1			0 1
21	False				2	1	0 1
22	False		4	1			1
23	False				4	1	1
24	False		8	5			
25		5					

C: Condição
V: Valor
P: Potência

N: Número
SA: Sequência
SB: SequênciaB
D: Dígito
S: Sinal

CAPÍTULO IV

LINGUAGEM DE ESPECIFICAÇÃO LEGA

Apenas o método definido no capítulo anterior não é suficiente para permitir uma utilização prática de gramáticas de atributos como especificação de semântica. É necessário, ainda, que uma forma de notação de alto nível permita expressar as funções semânticas e definir para cada atributo uma estrutura de dados correspondentes. Esta notação precisa poder ser, inclusive, usada separadamente como especificação de semântica de linguagens de programação.

Neste capítulo se define a linguagem de especificação para a gramática de atributos UPED.

A especificação da gramática é constituida de duas partes. Uma de declarações (atributos, variáveis, etc.) e outra de produções com suas funções semânticas.

Os tipos de dados para atributos e os comandos usados para especificar funções semânticas, são baseados na definição de PASCAL |JENSEN, WIRTH|[11] e conceitos de dados estruturados |HOARE|[8].

IV.1 - NOTAÇÃO

Para descrever a sintaxe da linguagem de especificação, usamos a notação Backus Naur e a definição de expressão regular |AHO|[2].

Uma construção sintática identifica-se por um nome em maiúsculo (símbolo não terminal).

Os elementos da linguagem se escrevem entre apóstrofos. (Símbolos terminais).

VAZIO representa uma cadeia sem símbolos. O símbolo ::= é usado para separar a identificação da construção sintática, de sua definição.

Na expressão regular são usados os seguintes operadores:

*: indica zero ou mais ocorrências do operando. (+: indica pelo menos uma ocorrência).

.: concatenação (a operação se especifica pela justaposição de operandos).

|: União.

A hierarquia da precedência de operadores, obede-

ce à seguinte ordem: *, .., | .

IV.2 - VOCABULÁRIO

O vocabulário básico da linguagem está formado por letras, dígitos e caracteres especiais.

CARACTER-ESPECIAL:::=, ';' | '.' | '=' | '(' | ')' | ','
|':' |'[' |']' |'}' |'{' |'→' |'↑' |'↓'
|'#' |'v' |'^' |''' |'"' |'+'|'-'|'*'|
'/'|'<'|'>'|'∅'

Com os elementos do vocabulário se representam os símbolos básicos da linguagem, que são os seguintes:

1. Identificadores - usados para definir variáveis, atributos , tipos de dados, constantes, símbolos sintáticos e cadeias de caracteres.

IDENTIFICADOR::= LETRA(LETRA|DÍGITO|'-')*

O identificador padrão é um identificador pré-definido. Tem-se os seguintes:

INTEIRO	VERDADEIRO
NOME	FALSO
BOOLEANO	INDEFINIDO
CARACTER	EMPTY

2. Número - é um número inteiro sem sinal

NÚMERO-INTEIRO::= (DÍGITO)$^{+}$

3. Cadeia de caracteres - é uma sequência de caracteres do vocabulário delimitado entre apóstrofos.

CADEIA-CARACTERES::=''' (LETRA|DÍGITO|CARÁCTER-ESPECIAL)$^{+}$'''

4. Nome constante - é uma sequência de caracteres alfanuméricos, inclui o símbolo '-'.

CONSTANTE-NOME:='"' LETRA(LETRA|DÍGITO|'-')*'"'

5. Palavras reservadas - são:

GRAMÁTICA	MAPA	ENTÃO	ENQUANTO
TEXTO	SEQUÊNCIA	INÍCIO	FAZER
TIPO	REGISTRO	IMPRIME	
ATRIBUTO	CASO	EM	
VARIÁVEL	FIM	E	
SÍMBOLOT	PRODUÇÕES	NÃO	
SÍMBOLONT	SE	OU	
DE	SENÃO		

6. Delimitadores e operadores - são:

;	+	[	/*
.	-	]	*/
=	*	{	∅
(	/	}	^
)	<>	→	
,	<	[*	
'	>	*]	
''	>=	#	
<<	<=		
>>	v		

Comentários são delimitados pelos símbolos /* e */.

IV.3 - ESTRUTURA DA LINGUAGEM DE ESPECIFICAÇÃO

A especificação da gramática de atributos consta da identificação da gramática, e de um bloco que contém declarações e regras (sintáticas e semânticas).

ESPECIFICAÇÃO-GRAMÁTICA::= IDENTIFICAÇÃO-GRAMÁTICA ';' BLOCO'.'

IDENTIFICAÇÃO-GRAMÁTICA::='GRAMÁTICA' IDENTIFICADOR-GRAMÁTICA

BLOCO::= DECLARAÇÕES REGRAS

IV.4 - DECLARAÇÕES

Os identificadores que são usados na especificação da gramática de atributos para definir cadeias de caracteres, tipos de dados, atributos, variáveis e símbolos sintáticos devem ser declarados.

DECLARAÇÕES::= DEF-TEXTO DEF-TIPO DEC-ATRIBUTO DEC-VARIÁVEL
DEC-SÍMBOLO-SINTÁTICO

IV.4.1 - Definição de Texto

Um texto é uma cadeia de caracteres, usado na impressão.

DEF-TEXTO::= 'TEXTO' LISTA-CADEIA (';' LISTA-CADEIA)*';'|VAZIO
LISTA-CADEIA::= IDENTIFICADOR '=' CADEIA-CARACTERES

IV.4.2 - Definição de Tipos de Dados

Define-se os tipos de dados que serão necessários na especificação de variáveis e atributos.

DEF-TIPO::='TIPO' LISTA-TIPO (';' LISTA-TIPO)* ';' | VAZIO
LISTA-TIPO::=IDENTIFICADOR'=' DESCRIP-TIPO

Em DESCRIP-TIPO descreve-se as características do tipo de dado.

IV.4.3 - Declaração de Atributos

Os atributos são considerados variáveis associadas aos símbolos sintáticos, e são locais numa produção.

```
DEC-ATRIBUTO ::= 'ATRIBUTO' DEFINIÇÃO-ATRIBUTO (';' DEFINIÇÃO-
                 ATRIBUTO)* ';'
DEFINIÇÃO-ATRIBUTO ::= IDENTIFICADOR CLASSE-ATRIBUTO (',' IDEN-
                       TIFICADOR CLASSE-ATRIBUTO)*
                       ':' DESCRIP-TIPO
CLASSE-ATRIBUTO ::= '↑'
                  | '↓'
```

O atributo é representado por IDENFICADOR. Se ele é herdado usamos o símbolo '↓', se é sintetizado o '↑'. O tipo de atributo é descrito por DESCRIP-TIPO.

IV.4.4 - Declaração de Variáveis

As variáveis são usadas para manipular informações dos atributos na definição das funções semânticas.

As variáveis são consideradas locais para um símbolo sintático na produção.

```
DEC-VARIÁVEL::= 'VARIÁVEL' DEFINIÇÃO-VARIÁVEL
                (';' DEFINIÇÃO-VARIÁVEL)* ';' | VAZIO
DEFINIÇÃO-VARIÁVEL::= IDENTIFICADOR (',' IDENTIFICADOR)* ':'
                      DESCRIP-TIPO
```

IV.4.5 - Declaração de Símbolos Sintáticos

Declara-se o símbolo sintático terminal ou não-terminal que possui atributos associados. Para cada símbolo dá-se a relação de seus atributos.

```
DEC-SÍMBOLO-SINTÁTICO::=(DEC-NÃO-TERMINAL|DEC-TERMINAL|VAZIO)
DEC-NÃO-TERMINAL::= 'SIMBOLONT' LISTA-DECLARAÇÃO
                    (';' LISTA-DECLARAÇÃO)* ';'
LISTA-DECLARAÇÃO::= LISTA-IDENTIFICADOR ':' LISTA-ATRIBUTOS
LISTA-IDENFIFICADOR::= IDENTIFICADOR (',' IDENTIFICADOR)*
LISTA-ATRIBUTOS::= IDENTIFICADOR-ATRIBUTO
                   (',' IDENTIFICADOR-ATRIBUTO)*
DEC-TERMINAL::= 'SIMBOLOT' LISTA-DECLARAÇÃO
                (';' LISTA-DECLARAÇÃO)* ';'
```

IV.5 - TIPOS DE DADOS

Os tipos de dados definem os possíveis valores que podem ser assumidos por uma variável, expressão ou atributo.

O conceito de tipo de dado segue a definição de dados estruturados |HOARE|[8], que são classificados em simples ou primários e estruturados.

Os estruturados são definidos a partir dos tipos simples e de outros já definidos.

DESCRIP-TIPO ::= TIPO-SIMPLES
| TIPO-CONJUNTO
| TIPO-SEQUÊNCIA
| TIPO-MAPA
| TIPO-REGISTRO
| TIPO-NOME

IV.5.1 - Tipos Simples

São tipos não estruturados, considerados como primários.

Os tipos simples são: o padrão, o escalar e o intervalo.

TIPO-SIMPLES ::= IDENTIFICADOR-TIPO|TIPO-ESCALAR|TIPO-INTERVALO

IV.5.1.1 - Tipo Padrão

É um tipo de dado pré-definido, denotado por INTEIRO, BOOLEANO, NOME, CARÁCTER.

1. INTEIRO.- Denota o subconjunto de números inteiros.

Aos operandos inteiros aplicam-se os seguintes operadores:

a) Aritméticos . *, /, +, -

b) Relacionais. =, <>, (teste de igualdade)
<, >, <=, >=

2. BOOLEANO - Denota o conjunto de valores lógicos verdadeiro e falso.

Os valores constantes são denotados pelos identificadores VERDADEIRO e FALSO.

Operadores lógicos que se aplicam aos operandos tipo BOOLEANO são:

OU (união lógica)
E (conjunção lógica)
NÃO (negação)

3. CARACTER.- Denota o conjunto ordenado de caracteres.

Os operadores relacionais que se aplicam aos operandos tipo CARÁCTER são:

=, <>, >, >=, <, <=

Uma constante especifica-se pelo carácter entre apóstrofos.

4. NOME.- Denota um nome constante de um número de carácteres determinado.

TIPO-NOME::= 'NOME''('NÚMERO-INTEIRO')"

Os operadores que se aplicam aos operandos tipo NOME são:

=, <> (teste igualdade)

IV.5.1.2 - Tipo Escalar

Define um conjunto ordenado de valores que são representados por identificadores.

Estabelece-se uma ordem ascendente com a sequência de identificadores definidos da esquerda para a direita.

TIPO-ESCALAR ::= '(' IDENTIFICADOR (';' IDENTIFICADOR)*')'

Os operadores relacionais que são aplicados aos operandos tipo escalar são:

=, <>, <, >, >=, <=

IV.5.1.3 - Tipo Intervalo

Denota um subconjunto de um tipo escalar ou padrão.

Será assumido o tipo dado pelas constantes na especificação.

```
TIPO-INTERVALO ::= CONSTANTE^A '..' CONSTANTE^B
CONSTANTEA     ::= CONSTANTE
CONSTANTEB     ::= CONSTANTE
CONSTANTE      ::= IDENTIFICADOR-CONSTANTE
                 | NÚMERO-INTEIRO
                 | ''' CARACTER '''
CARACTER       ::= LETRA
                 | DÍGITO
                 | CARACTER-ESPECIAL
```

O valor dado para CONSTANTEA deve ser $\leq$ ao dado para CONSTANTE^B.

IV.5.2 - Tipos Estruturados

A definição de um tipo de dado estruturado é realizada usando tipos de dados definidos previamente.

Os métodos básicos para definir os tipos estruturados são: MAPA, SEQUÊNCIA, CONJUNTO e REGISTRO.

IV.5.2.1 - Tipo CONJUNTO

Define um subconjunto de elementos que possuem seu tipo definido pelo tipo base, que é um escalar ou intervalo.

TIPO-CONJUNTO ::= 'CONJUNTO' 'DE' TIPO-BASE
TIPO-BASE ::= TIPO-SIMPLES

O conjunto denota-se pelos símbolos '[', ']', e seus elementos se separam por ','.

[ ] denota o conjunto vazio.

Os operadores que se aplicam ao tipo CONJUNTO são:

+ (união)
- (diferença)
* (interseção)
= (teste igualdade)
<> (teste diferente)
EM (pertence)
<=, < (contido)
>=, > (contém)

IV.5.2.2 - Tipo SEQUÊNCIA

O tipo SEQUÊNCIA define uma estrutura ordenada de componentes que são elementos do mesmo tipo.

TIPO-SEQUÊNCIA ::= 'SEQUÊNCIA' 'DE' DESCRIP-TIPO

Uma sequência se expressa pelo conteúdo delimitado por '[*' e '*]'.

Os operadores que se aplicam ao tipo SEQUÊNCIA são:

^ (concatenação)
# (desencadea um componente mais a esquerda da SEQUÊNCIA)
↓ (operação de acesso. Faz referência ao componente mais a esquerda da SEQUÊNCIA)
=, <> (testa igualdade, só no caso de comparação de um tipo SEQUÊNCIA com EMPTY = sequência vazia)

IV.5.2.3 - Tipo MAPA

O tipo MAPA é uma estrutura que consiste de componentes do mesmo tipo.

TIPO-MAPA ::= 'MAPA' '{' TIPO-DOMÍNIO '→' TIPO-CONTEÚDO '}'
TIPO DOMÍNIO ::= TIPO-NOME
| TIPO-SIMPLES
TIPO-CONTEÚDO ::= DESCRIP-TIPO

Um componente do MAPA é um par (a, b), onde a é um elemento do tipo dado por TIPO-DOMÍNIO e é usado para ter acesso ao elemento b ε ({TIPO-CONTEÚDO} $\cup$ {'INDEFINIDO'}).

Os operadores que se aplicam ao tipo MAPA são:

1. Acesso - seja M = MAPA {A → B}. Uma entrada ou acesso em M para a ε A especifica-se por M[a], sendo que:

   M[a] ε B ou é INDEFINIDO

2. União - Existem dois tipos de união com MAPA, a simples e a discriminada. Interpretamo-las como segue:

   Sejam M_1 e M_2 tipos: MAPA {A → B}

   a) União simples (+)

   $M_1 + M_2$ = para todo a ε A fazer
   (Se M_1[a] <> indefinido
   então se M_2[a] <> indefinido
   então ERRO
   senão M_1[a]
   senão M_2[a])

   b) União discriminada (/)

   M_1/M_2 = Para todo a ε A fazer
   (se M_2[a] <> indefinido
   então M_2[a]
   senão M_1[a])

IV.5.2.4 - Tipo REGISTRO

O tipo REGISTRO é uma estrutura constituída de campos definidos por tipos de dados.

O tipo REGISTRO é equivalente ao tipo "RECORD" do PASCAL [JENSEN, WIRTH][11]

```
TIPO-REGISTRO ::= 'REGISTRO' LISTA-CAMPOS 'FIM'
LISTA-CAMPOS  ::= PARTE-FIXA
                | PARTE-FIXA ';' PARTE-VARIÁVEL
                | PARTE-VARIÁVEL
PARTE-FIXA    ::= CAMPO (';' CAMPO)*
CAMPO         ::= IDENTIFICADOR-SELEÇÃO (';' IDENTIFICADOR-SE-
                  LEÇÃO)*
                  ':' DESCRIP-TIPO
PARTE-VARIÁVEL::= ['CASO' DETEÇÃO TIPO-SIMPLES 'DE'
                  VARIANTE (';' VARIANTE)*
VARIANTE      ::= CABEÇALHO ':' '('LISTA-CAMPOS')'
CABEÇALHO     ::= CONSTANTE (',' CONSTANTE)*
DETEÇÃO       ::= IDENTIFICADOR-DISCRIMINAÇÃO ':'
                | VAZIO
```

IV.6 - EXPRESSÃO

A expressão é uma combinação de operadores, constantes, variáveis, etc., para obter um valor de um certo tipo.

```
EXPRESSÃO ::= SIMPLE-EXP (VAZIO|OP-RELAÇÃO SIMPLE-EXP)
SIMPLE-EXP::= OP-UNÁRIO TÉRMINO
            | TÉRMINO
            | SIMPLE-EXP OP-ADIÇÃO TÉRMINO
TÉRMINO    ::=FATOR (OP-MULTIPLICAÇÃO FATOR)*
FATOR      ::= CONSTANTE
            | CONSTANTE-NOME
            | VARIABLE
            | ATRIBUTO
            | '[' (VAZIO|EXPRESSÃO (';' EXPRESSÃO)*)']'
            | '{' CORPO-MAPA '}'
            | '(' EXPRESSÃO ')'
            | OP-NEGAÇÃO FATOR
            | '[*' (EXPRESSÃO|VAZIO)'*]'
CORPO-MAPA::= EXPRESSÃO '→' EXPRESSÃO
            | VAZIO
```

A seguir dá-se as regras de cálculo das expressões, definidas pela precedência de operadores e a ação dos operadores sobre os operandos.

IV.6.1 - Precedência de Operadores

Os operadores que intervêm em expressão, são classificados em:

1. Operador de negação

   OP-NEGAÇÃO ::= 'NÃO'

2. Operadores de multiplicação

OP-MULTIPLICAÇÃO ::= '*'|'/'|'E'

3. Operadores de adição

OP-ADIÇÃO ::= '+'|'-' |'OU' |'^'

4. Operadores unários

OP-UNÁRIO ::= '+'|'-'|'# '

5. Operadores relacionais

OP-RELAÇÃO ::= '=' |'<>'|'>'|'<'|'>='|'<='|'EM"

No seguinte quadro indica-se a precedência dos operadores :

OPERADOR	PRECEDÊNCIA
NEGAÇÃO	+ ALTA
MULTIPLICAÇÃO	↓
ADIÇÃO,UNARIO	↓
RELACIONAL	+ BAIXA

IV.6.2 - Ação dos Operadores

Para os operadores permitidos em expressão se indica os tipos de operandos e do resultado.

1. Operador de negação

Operador: NÃO
Tipo do operando: BOOLEANO
Tipo do resultado: BOOLEANO

2. Operador de multiplicação

OPERADOR	TIPOS DOS OPERANDOS	TIPO DO RESULTADO
*	INTEIRO	INTEIRO
	C:CONJUNTO COM TIPO BASE T	C
/	INTEIRO	INTEIRO
	MAPA {A → B}	MAPA {A → B}
E	BOOLEANO	BOOLEANO

3. Operadores de adição

OPERADOR	TIPOS DOS OPERANDOS	TIPO DO RESULTADO
+	INTEIRO	INTEIRO
	C:CONJUNTO COM TIPO BASE T	C
	MAPA {A → B}	MAPA {A → B}
-	INTEIRO	INTEIRO
	C:CONJUNTO COM TIPO BASE T	C

OU	BOOLEANO	BOOLEANO
^	SEQUÊNCIA	SEQUÊNCIA

4. Operadores de relação

OPERADOR	TIPOS DOS OPERANDOS	TIPO DO RESULTADO
=, <>	ESCALAR	BOOLEANO
	PADRÃO	BOOLEANO
	NOME(N); N=N^{O} DE CARAC	
	TERES	BOOLEANO
	SEQUÊNCIA	BOOLEANO
	CONJUNTO	BOOLEANO
<,>,<=,>=	ESCALAR	BOOLEANO
	PADRÃO	BOOLEANO
	CONJUNTO	BOOLEANO
EM	T:TIPO BASE DO CONJUNTO C	BOOLEANO
	(LADO ESQUERDO) e	
	CONJUNTO C COM TIPO BASE T	
	(LADO DIREITO)	

5. Operadores unários

OPERADOR	TIPO DE OPERANDO	TIPO DO RESULTADO
+	INTEIRO	INTEIRO
-	INTEIRO	INTEIRO
#	SEQUÊNCIA COM COMPONEN-	TIPO COMPONENTE: S
	TES TIPO S	

IV.7 - COMANDOS

Os comandos são usados para expressar as funções semânticas.

```
COMANDO ::= ATRIBUIÇÃO
          | CONDICIONAL
          | IMPRESSÃO
          | COMPOSTO
          | REPETIÇÃO
          | VAZIO
```

1. ATRIBUIÇÃO

```
ATRIBUIÇÃO ::= VARIABLE-ATRIBUIÇÃO ' :=' EXPRESSÃO
```

2. CONDICIONAL

```
CONDICIONAL ::= 'SE' EXPRESSÃO 'ENTÃO' COMANDO OPÇÃO-CONDI-
                CIONAL
OPÇÃO-CONDICIONAL ::= 'SENÃO' COMANDO
                    | VAZIO
```

3. IMPRESSÃO

```
IMPRESSÃO ::= 'IMPRIME''('ELEMENTO-IMP (',' ELEMENTO-IMP)*')'
ELEMENTO-IMP ::= IDENTIFICADOR-CADEIA
               | CADEIA-CARACTERES
```

4. COMPOSTO

```
COMPOSTO ::= 'INÍCIO' COMANDO (';' COMANDO)*'FIM'
```

5. REPETIÇÃO

REPETIÇÃO ::= 'ENQUANTO' EXPRESSÃO 'FAZER' COMANDO

IV.8 - REGRAS SINTÁTICAS E SEMÂNTICAS

As regras correspondem ás produções e funções semânticas.

REGRAS ::= 'PRODUÇÕES' LISTA-PRODUÇÕES 'FIM'

LISTA-PRODUCÕES ::= PRODUÇÃO (';' PRODUÇÃO)*

IV.8.1 - Produções

As produções são escritas levando em consideração as condições da gramática UPED.

Usa-se a notação Backus Naur para escrever as produções. O símbolo '=' separa a parte esquerda da direita na produção.

```
PRODUÇÃO ::= SÍMBOLO-NÃO-TER '=' PARTE-DIR
PARTE-DIR::= FUNÇÕES (SÍMBOLO-FUNÇÕES)*
SÍMBOLO-FUNÇÕES::= SÍMBOLO-SINTÁTICO FUNÇÕES
FUNÇÕES ::= LISTA-FUNÇÕES
          | VAZIO
SÍMBOLO-NÃO-TER ::= IDENTIFICADOR-NÃO-TER
SÍMBOLO-SINTÁTICO ::= SÍMBOLO-NÃO-TER
                    | SÍMBOLO-TER
SÍMBOLO-TER ::= CADEIA-CARACTERES
```

Nas produções, para cada símbolo sintático, escreve-se logo a seguir as suas funções semânticas.

IV.8.2 - Funções Semânticas

As funções semânticas expressam-se mediante os comandos propostos, que são associados a cada símbolo sintático na produção.

```
LISTA-FUNÇÕES ::= '<<' COMANDO (';' COMANDO)*'>>'
```

IV.8.2.1 - Uso de Variáveis e Atributos

1. No comando atribuição, do lado esquerdo do sinal, pode ter-se uma variável ou atributo. Para o atributo é opcional a especificação do símbolo sintático ao qual é associado.

```
VARIABLE-ATRIBUIÇÃO ::= VARIABLE
                      | ATRIBUTO-ATRIBUIÇÃO
ATRIBUTO-ATRIBUIÇÃO ::= IDENTIFICADOR-ATRIBUTO
                CLASSE-ATRIBUTO ('('IDENTIFICADOR-SÍMBOLO')'|
                VAZIO)
                RESTO-VAR
RESTO-VAR ::= (CLASSE-VAR)*
CLASSE-VAR::= VAR-MAPA
            | VAR-CAMPO-REGISTRO
            | VAR-ACESSO-SEQUÊNCIA
VAR-MAPA ::= '[' EXPRESSÃO ']'
```

VAR-CAMPO-REGISTRO ::= '.' IDENTIFICADOR-CAMPO

VAR-ACESSO-SEQUÊNCIA ::= 'v'

2. Numa expressão, um atributo é associado a um símbolo sintático, e se denota pelo nome do atributo seguido pelo símbolo sintático entre parênteses.

No caso de ter vários símbolos sintáticos iguais na produção, eles vão ser referenciados com índices (0, 1,2,...) entre parênteses angulares. A correspondência com os índices e os símbolos sintáticos iguais se dá da direita para a esquerda da produção. O símbolo sintático do lado esquerdo da produção é especificado sem índice.

ATRIBUTO ::= IDENTIFICADOR-ATRIBUTO CLASSE-ATRIBUTO
'(' IDENTIFICADOR-SÍMBOLO
('<' NÚMERO-INTEIRO '>'|VAZIO)')'RESTO-VAR

3. A variável não precisa nenhuma referência (do símbolo sintático) por que é considerada local para cada símbolo na produção.

VARIABLE ::= IDENTIFICADOR-VARIABLE RESTO-VAR

CAPÍTULO V

EXEMPLOS DE APLICAÇÃO

V.1 - A LINGUAGEM S

Apresentamos uma linguagem simples S, derivada do Pascal |JENSEN|[11]. A sintaxe e semântica estática é especificada usando a linguagem LEGA.

V.1.1- CARACTERÍSTICAS DA LINGUAGEM S

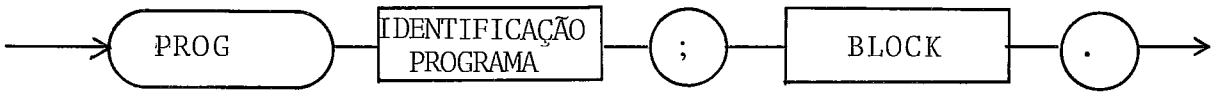
Um programa que se escreve na linguagem,consta de declarações e comandos.

As declarações fazem-se para constantes e variáveis.

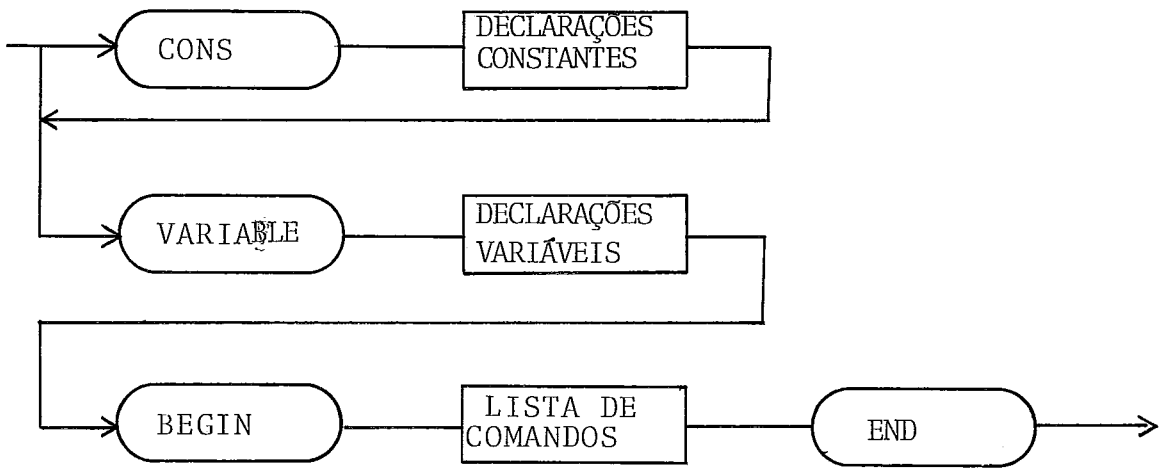
O diagrama da sintaxe se mostra no Apêndice B.

1. Estrutura de um programa

PROGRAMA



BLOCK



2. Tipos de dados

São permitidos os seguintes tipos de dados:

INTEGER

BOOLEAN

CHAR

SUBRANGE

Os três primeiros são do tipo padrão.

SUBRANGE denota um subconjunto de valores INTEGER ou CHAR.

Especificação: CONSTANTE-1 .. CONSTANTE-2
CONSTANTE-1 e CONSTANTE-2 devem ser ambas tipo INTEGER ou CHAR e |CONSTANTE-1| $\leq$ |CONSTANTE-2|. O tipo de SUBRANGE é assumido do tipo das constantes que o definem.

3. Expressão

Os operadores que são permitidos numa expressão , são dados a seguir:

Operador	Tipo de Operando Esquerdo-Direito	Tipo do Resultado
(+,-,*,/)	INTEGER INTEGER	INTEGER
LÓGICO (AND,OR)	BOOLEAN BOOLEAN	BOOLEAN

RELAÇÃO	INTEGER	INTEGER	BOOLEAN
(<>,=,<,>,<=,>=)	CHAR	CHAR	BOOLEAN
	BOOLEAN	BOOLEAN	BOOLEAN

4. Comandos

São considerados os seguintes comandos:

a) Atribuição - Se realiza a atribuição quando a variável do lado esquerdo do sinal de atribuição e a expressão tem os mesmos tipos.

TIPO DO LADO ESQ. DO SINAL DE ATRIB:	TIPO DE EXPRESSÃO
INTEGER	INTEGER
BOOLEAN	BOOLEAN
CHAR	CHAR

b) Repetição e condicional - O tipo da expressão usado nestes comandos, deve ser BOOLEAN.

c) Impressão - Só se faz impressões de variáveis.

d) Composto.

5. Regras de escopo

a) As variáveis e constantes são globais. Isto é, elas vão conservar seu tipo durante todo o programa.

b) Um identificador só pode ser declarado uma vez como constante ou variável.

V.1.2- - A Gramática de Atributos da Linguagem S

```
GRAMATICA LINGUAGEM-S;
TEXTO
      ERRO1 = 'IDENTIFICADOR MULTIDECLARADO';
      ERRO2 = 'IDENTIFICADOR NAO RECONHECIDO';
      ERRO4 = 'IDENTIFICADOR NAO E DO TIPO ESPERADO';
      ERRO7 = 'ERRO EM CONSTANTES EM SUBRANGE';
      ERRO8 = 'O TIPO DE EXPRESSAO DEVE SER BOOLEAN';
      ERRO10 = 'EM ATRIBUICAO OS TIPOS DE VARIAVEL E
                EXPRESSAO NAO SAO COMPATIVEIS';
      ERRO12 = 'ID NAO E DA CLASSE ESPERADA';
      ERRO16 = 'EM EXPRESSAO OS TIPOS DE OPERANDOS USADOS
                COM O OPERADOR DE RELACAO NAO SAO COMPATIV.';
      ERRO17= 'EM EXPRESSAO, OS TIPOS DE OPERANDOS USADOS
               COM O OPERADOR LOGICO OU ARIT. NAO SAO
               COMPATIVEIS';
TIPO
      CLASSDADO = (TYPE,VAR,CONST,INDEFC);
      ORD = (INT,CHARACTER,BOOL,INDEFT);
      TIPORANGO = (SUBRANGE,SIMPLES,TOTAL);
      TIPOTYPE = REGISTRO
                 ORDINAL : ORD;
                 CASO RANGO : TIPORANGO DE
                      SUBRANGE : (V1,V2 : INTEIRO);
                      SIMPLES : (VALOR : INTEIRO);
                      TOTAL : (VALORTOT : (ALL));
                 FIM;
      OBJETOTIPO = REGISTRO
                   CLASSE : CLASSDADO;
                   TYPEP : TIPOTYPE;
                   FIM;
      ENVIR =  MAPA{NOME(10) -> OBJETOTIPO};
ATRIBUTO
      /*                                                  */
      /*   USED,ORIG                                      */
      /*             COLECAO DE INFORMACAO DE IDENT.      */
      /*   SEQNOME                                        */
      /*             SEQUENCIA DE NOMES DE IDENT.         */
      /*   TYPES                                          */
      /*             CARACTERISTICAS DO TIPO DO IDENT     */
      /*   TYPEVALOR                                      */
      /*             TIPO DE DADO                         */
      /*   OP                                             */
      /*             TIPO DE OPERADOR                     */
      /*   CLASSVALOR                                     */
      /*             CLASSE DO IDENT.                     */
      /*   NUMERO                                         */
      /*               NUMERO INTEIRO                     */
      /*   NAME                                           */
      /*               NOME DO IDENT                      */
      /*                                                  */
```

```
      USED↓,ORIG↑ : ENVIR;
      SEQNOME↑ : SEQUENCIA DE NOME(10);
      TYPES↑ : TIPOTYPE;
      TYPEVALOR↑ : ORD;
      OP↑ : (ARIT,LOG);
      CLASSVALOR↑: CLASSDADO;
      NUMERO↑ : INTEIRO;
      NAME↑ : NOME(10);
      CONDICAO↑ : BOOLEANO;
VARIAVEL
      OBJETO : OBJETOTIPO;
      ENV : ENVIR;
      NAMEVAR : NOME(10);
      CLASS1,CLASS2 : CLASSDADO;
      TYPE1,TYPE2 : ORD;
SIMBOLONT
      PROGRAM : ORIG;
      BLOCK : USED,ORIG;
      CONST-DEC,VAR-DEC : USED,ORIG;
      STAT : USED;
      LISTA-CONST,LISTA-VAR : USED,ORIG;
      CONSTANT,TYPE : USED;TYPES;
      INTERVALO : CONDICAO,USED,TYPES;
      LISTA-ID : SEQNOME;
      COMANDO-LIS,COMANDO,ASSIGN,IF-STAT,WRITE-STAT : USED;
      COMP-STAT,WHILE-STAT : USED;
      LISTA-ID-WR :USED;
      EXP,VARIABLE,SEX,TERM,FATOR : USED,TYPEVALOR;
      IDENT-FAT : CLASSVALOR,USED,TYPEVALOR;
      OP-ARIT : OP;
      CONSTANT-ID : USED,TYPES;
      CONSTANT-FA : TYPES;
      OP-MULT : OP;
      IF-REST : USED;
      LISTA-CON : SEQNOME;
      LISTA-WR-R : USED;
      EXP-RES : USED,CONDICAO,TYPEVALOR;
      SEX-RES,TERM-R : USED,CONDICAO,TYPEVALOR,OP;
SIMBOLOT
      IDENT : NAME;
      CHAR  : NUMERO;
      UNSIG-CONS : NUMERO;
PRODUCOES
PROGRAM     =          << ORIG↑ := ORIG↑(BLOCK)              >>
            'PROG'
            'IDENT'    << NAME↑ := IDENT                     >>
            ';'
            BLOCK      << /*COLOCAR OS ID PRE-DEF EM USED */
                          OBJETO.CLASSE := TYPE;
                          OBJETO.TYPEP.ORDINAL := INT;
```

```
                        OBJETO.TYPEP.RANGO := TOTAL;
                        OBJETO.TYPEP.VALORTOT := ALL;
                        ENV := {"INTEGER" ->OBJETO};
                        OBJETO.TYPEP.ORDINAL := BOOL;
                        ENV := ENV + {"BOOLEN" ->OBJETO};
                        OBJETO.TYPEP.ORDINAL := CHARACTER;
                        ENV := ENV + {"CHAR" ->OBJETO};
                        OBJETO.CLASSE := CONST;
                        OBJETO.TYPEP.ORDINAL := BOOL;
                        OBJETO.RANGO := SIMPLES;
                        OBJETO.VALOR := 0;
                        ENV := ENV + {"FALSE" ->OBJETO};
                        OBJETO.VALOR := 1;
                        ENV := ENV + {"TRUE" ->OBJETO};
                        USED↓ := ENV;                          >>
        '.'
        ;
BLOCK   =           <<ORIG↑ := ORIG↑(CONST-DEC)                >>
        CONST-DEC   <<USED↓ := USED↓(BLOCK)                    >>
        ;
CONST-DEC =         <<ORIG↑ := ORIG↑(VAR-DEC)                  >>
        'CONS'
        LISTA-CONST<<USED↓ := USED↓(CONST-DEC)                 >>
        VAR-DEC     <<USED↓ := ORIG↑(LISTA-CONST)              >>
        ;
CONST-DEC =         <<ORIG↑ := ORIG↑(VAR-DEC)                  >>
        VAR-DEC     <<USED↓ := USED↓(CONST-DEC)                >>
        ;
LISTA-CONST =        << ORIG↑ := ORIG↑(LISTA-RES)              >>
        'IDENT'      <<NAME↑ := IDENT                          >>
        '='
        CONSTANT     <<USED↓ := USED↓(LISTA-CONST)             >>
        ';'
        LISTA-CONST   <</*DECLARACAO DE CONSTANTE  */
                      SE USED↓(LISTA-CONST)[NAME↑(IDENT)]
                         <> INDEFINIDO
                      ENTAO    /* ID MULTIDEFINIDO      */
                          SE USED↓(LISTA-CONST)[NAME↑
                             (IDENT)].CLASSE <> INDEFC
                          ENTAO /*   ERRO               */
                              INICIO
                              OBJETO.CLASSE := INDEFC;
                              OBJETO.TYPEP.ORDINAL :=
                                              INDEFT;
                              IMPRIME ERRO1;
                              USED↓ := USED↓(LISTA-CONST)
                               / {NAME↑(IDENT) ->OBJETO}
                               FIM;
                      SENAO /* INCLUIR CONST EM ORIG*/
                          INICIO
```

```
                                OBJETO.CLASSE := CONST;
                                OBJETO.TYPEP :=
                                      TYPES↑(CONSTANT);
                                 USED↓ := USED↓(LISTA-CONST)
                                  + {NAME↑(IDENT) ->OBJETO }
                                FIM                                    >>
            ;
LISTA-CONST =           <<ORIG↑ := USED↓(LISTA-CONST)                  >>
            ;
VAR-DEC     =            <<ORIG↑ := ORIG↑(LISTA-VAR)                   >>
            'VARIABLE'
            LISTA-VAR    <<USED↓ := USED↓(VAR-DEC)                     >>
            STAT        <<USED↓ := ORIG↑(LISTA-VAR)                    >>
            ;
LISTA-VAR   =            <<ORIG↑ := ORIG↑(LISTA-V-R)                   >>
            LISTA-ID
            ':'
            TYPE         <<USED↓ := USED↓(LISTA-VAR)                   >>
            ';'
            LISTA-VAR    <</* DECLARACAO DE VARIAVEIS  */
                           ENV := USED↓(LISTA-VAR);
                           /* LISTA DE ID DECLARADOS EM ENV*/
                           ENQUANTO SEQNOME↑(LISTA-ID) <>
                                 EMPTY
                           FAZER
                                INICIO
                                NAMEVAR := #SEQNOME↑(LISTA-ID);
                                /* VERIFICAR SE O ID EXISTE  */
                                SE ENV[NAMEVAR] <> INDEFINIDO
                                ENTAO   /* MULTIDECLARADO  */
                                     SE ENV[NAMEVAR].CLASSE <>
                                          INDEFC
                                     ENTAO  /*  ERRO      */
                                          INICIO
                                          OBJETO.CLASSE := INDEFC;
                                          OBJETO.TYPEP.ORDINAL :=
                                                INDEFT;
                                          IMPRIME ERRO1;
                                          ENV := ENV /
                                            {NAMEVAR ->OBJETO }
                                          FIM;
                                SENAO /*DAR O MODO DO ID*/
                                     INICIO
                                     OBJETO.CLASSE := VAR;
                                     OBJETO.TYPEP :=
                                          TYPES↑(TYPE);
                                     ENV := ENV +
                                        { NAMEVAR ->OBJETO }
                                     FIM;
                               FIM;
```

```
                    USED↓ := ENV                              >>
                ;
LISTA-VAR       =   <<ORIG↑ := USED↓(LISTA-VAR)               >>
                ;
TYPE            =   <<SE USED↓(TYPE)
                      [NAME↑(IDENT)] <> INDEFINIDO
                    ENTAO /* ANALIZAR IDENT */
                         INICIO
                         OBJETO := USED↓(TYPE)
                           [NAME↑(IDENT)]
                         SE NAO CONDICAO↑
                         ENTAO /* E IDENT. DE TIPO */
                              SE (OBJETO.CLASSE = TYPE)
                               OU (OBJETO.CLASSE = INDEFC)
                              ENTAO /* OK     */
                                   TYPES↑ := OBJETO.TYPEP
                              SENAO /*   ERRO  $*/
                                   INICIO
                                   TYPES↑.ORDINAL := INDEFT
                                   IMPRIME ERRO12
                                   FIM
                         SENAO /* INTERVALO  */
                              SE (OBJETO.CLASSE = CONS)
                              OU (OBJETO.CLASSE = INDEFC)
                              ENTAO /*  OK CONSTANTE*/
                                   INICIO
                                   TYPE1 :=
                                    OBJETO.TYPEP.ORDINAL;
                                   TYPE2 :=
                                    TYPES↑(INTERVALO).ORDINAL
                                   SE(TYPE1 <> INDEFT) E
                                     (TYPE2 <> INDEFT)
                                   ENTAO /* LIMITES*/
                                     SE(OBJETO.TYPEP.VALOR)
                                     <= (TYPES↑(INTERVALO)
                                      .VALOR) E
                                     (((TYPE1 = INT) E
                                       (TYPE2 = INT) OU
                                      ((TYPE1 = CHAR) E
                                       (TYPE2) = CHAR)))
                                     ENTAO /* DAR MODO */
                                       INICIO
                                       TYPES↑.RANGO :=
                                        SUBRANGE;
                                       TYPES↑.V1 :=
                                       TYPES↑(INTERVALO)
                                         .VALOR;
                                       TYPES↑.V2 :=
                                        OBJETO.TYPEP.VALOR;
                                       TYPES↑.ORDINAL :=
```

```
                                        TYPE1;
                                      FIM
                                    SENAO /* ERRO  */
                                      INICIO
                                      TYPES↑.ORDINAL :=
                                       INDEFT;
                                      IMPRIME ERRO7
                                      FIM
                                  SENAO /* CONST INDEF.*/
                                    TYPES↑.ORDINAL := INDEFT
                              SENAO /* ERRO  */
                                  INICIO
                                  TYPES↑.ORDINAL := INDEFT;
                                  IMPRIME ERRO12
                                  FIM
                         SENAO /* ID NAO RECONHECIDO*/
                             INICIO
                             TYPES↑.ORDINAL := INDEFT;
                             IMPRIME ERRO2
                             FIM                                    >>
        'IDENT'    << NAME↑ := IDENT                                >>
        INTERVALO <<USED↓ :=  USED↓(TYPE)                           >>
        ;
TYPE    =           << /* TIPO TYPE              */
                      TYPE1 :=
                           TYPES↑(CONSTANT-FA).ORDINAL;
                      TYPE2 :=
                           TYPES↑(CONSTANT).ORDINAL;
                      SE (TYPE1 <> INDEFT) E
                         (TYPE2 <> INDEFT)
                      ENTAO /* VERIFICAR LIMITES  */
                          SE(TYPES↑(CONSTANT-FA).VALOR <=
                             TYPES↑(CONSTANT).VALOR) E
                             (((TYPE1 = INT) E
                               (TYPE2 = INT) OU
                              ((TYPE1 = CHAR) E
                               (TYPE2 = CHAR)))
                          ENTAO   /* DAR O MODO  */
                              INICIO
                              TYPES↑.RANGO := SUBRANGE;
                              TYPES↑.V1 :=
                              TYPES↑(CONSTANT-FA).VALOR;
                              TYPES↑.V2 :=
                              TYPES↑(CONSTANT).VALOR;
                              TYPES↑.ORDINAL := TYPE1;
                              FIM
                          SENAO /*  ERRO    */
                              INICIO
                              TYPES↑.ORDINAL  := INDEFT;
                              IMPRIME ERRO7;
```

```
                                    FIM
                              SENAO /* ALGUMA CONST E INDEF   */
                                  TYPES↑.ORDINAL := INDEFT      >>
                    CONSTANT-FA   <<USED↕ := USED↓(TYPE)        >>
                    '..'
                    CONSTANT   <<USED↓ := USED↓(TYPE)           >>
                    ;
INTERVALO           =           <<TYPES↑ := TYPES↑(CONSTANT)
                                  CONDICAO↑ := VERDADEIRO          >>
                    '..'
                    CONSTANT    <<USED↓ := USED↓(INTERVALO)         >>
                    ;
INTERVALO           =           <<CONDICAO↑ := FALSO                >>
                    ;
STAT                =
                    'BEGIN'
                    COMANDO-LIS<</* AMBIENTE DE STAT PARA COM */
                                  USED↓ := USED↓(STAT)          >>
                    'END'
                    ;
COMANDO-LIS         =
                    COMANDO     <<USED↓ := USED↓(COMANDO-LIS)   >>
                    COMANDO-CON<<USED↓ := USED↓(COMANDO-LIS)    >>
                    ;
COMANDO-CON         =
                    ';'
                    COMANDO     <<USED↓ := USED↓(COMANDO-CON)   >>
                    COMANDO-CON<<USED↓ := USED↓(COMANDO-CON)    >>
                    ;
COMANDO-CON         =
                    ;
COMANDO             =
                    ASSIGN      <<USED↓ := USED↓(COMANDO)       >>
                    ;
COMANDO             =
                    IF-STAT     <<USED↓ := USED↓(COMANDO)       >>
                    ;
COMANDO             =
                    WRITE-STAT <<USED↓ := USED↓(COMANDO)        >>
                    ;
COMANDO             =
                    COMP-STAT   <<USED↓ := USED↓(COMANDO)       >>
                    ;
COMANDO             =
                    WHILE-STAT <<USED↓ := USED↓(COMANDO)        >>
                    ;
COMANDO             =
                    ;
COMP-STAT           =
                    'BEGIN'
```

```
                COMANDO-LIS<<USED↓ := USED↓(COMP-STAT)          >>
                'END'
                ;
WHILE-STAT      =               <<(*VERIFICA-SE O TIPO DE EXP  *)
                                  SE(TYPEVALOR↑(EXP) <> BOOL) OU
                                    (TYPEVALOR↑(EXP) <> INDEFT)
                                  ENTAO
                                      IMPRIME ERRO8             >>
                'WHILE'
                EXP             <<USED↓ := USED↓(WHILE-STAT)    >>
                'DO'
                COMANDO         <<USED↓ := USED↓(WHILE-STAT)    >>
                ;
ASSIGN          =               <<(* VERIFICA-SE COMPATIB. *)
                                  (* DE TIPO                 *)
                                  TYPE1 := TYPEVALOR↑(VARIABLE);
                                  TYPE2 := TYPEVALOR↑(EXP);
                                  SE NAO(((TYPE1 = INT) E
                                          (TYPE2 = INT))
                                  OU ((TYPE1 = BOOL) E (TYPE2 = BOOL))
                                  OU(TYPE1 = INDEFT)
                                  OU (TYPE2 = INDEFT))
                                  ENTAO
                                      IMPRIME ERRO10            >>
                VARIABLE        <<USED↓ := USED↓(ASSIGN)        >>
                ':='
                EXP             <<USED↓ := USED↓(ASSIGN)        >>
                ;
IF-STAT         =               <<(* VERIFICA-SE O TIPO DE EXP  *)
                                  SE(TYPEVALOR↑(EXP) <> BOOL) OU
                                    (TYPEVALOR↑(EXP) <> INDEFT)
                                  ENTAO
                                      IMPRIME ERRO8             >>
                'IF'
                EXP             <<USED↓ := USED↓(IF-STAT)       >>
                'THEN'
                COMANDO         <<USED↓ := USED↓(IF-STAT)       >>
                IF-REST         <<USED↓ := USED↓(IF-STAT)       >>
                ;
IF-REST         =
                'ELSE'
                COMANDO         <<USED↓ := USED↓(IF-REST)       >>
                ;
IF-REST         =
                ;
WRITE-STAT      =
                'WRITE'
                '('
                LISTA-ID-WR<<USED↓ := USED↓(WRITE-STAT)         >>
                ')'
```

```
          ;
LISTA-ID-WR =          << /* VERIFICA SE O ID EXISTE  */
                         SE USED↓(LISTA-ID-WR) [NAME↑(IDENT)]
                            <> INDEFINIDO
                         ENTAO
                              INICIO
                              CLASS1 := USED↓(LISTA-ID-WR)
                               [NAME↑(IDENT)].CLASSE
                              SE(CLASS1 <> VAR) OU
                                 (CLASS1 <> INDEFC)
                              ENTAO /* ID NAO E VAR  */
                                   IMPRIME ERRO12
                              FIM
                         SENAO /* ID NAO E RECONHECIDO  */
                              IMPRIME ERRO2                  >>
          'IDENT'        <<NAME↑ := IDENT                    >>
          LISTA-WR-R     <<USED↓ := USED↓(LISTA-ID-WR)       >>
          ;
LISTA-WR-R =           <<
                         SE USED↓(LISTA-WR-R)[NAME↑(IDENT)]
                            <> INDEFINIDO
                         ENTAO
                              INICIO
                              CLASS1 := USED↓(LISTA-WR-R)
                                 [NAME↑(IDENT)].CLASSE
                              SE (CLASS1 <> VAR) OU
                                 (CLASS1 <> INDEFC)
                              ENTAO
                                   IMPRIME ERRO12
                              FIM
                         SENAO
                              IMPRIME ERRO2                  >>
          ','
          'IDENT'        <<NAME↑ := IDENT                    >>
          LISTA-WR-R     <<USED↓ := USED↓(LISTA-WR-R)        >>
          ;
LISTA-WR-R =
          ;
VARIABLE   =           <</* VERIFICA SE EXISTE O ID   */
                         SE USED↓(VARIABLE)[NAME↑(IDENT)]
                            <> INDEFINIDO
                         ENTAO  /* CONSEGUIR SEU TIPO */
                              INICIO
                              TYPEVALOR↑ := USED↓(VARIABLE)
                               [NAME↑(IDENT)].TYPEP.ORDINAL;
                              CLASS1 := USED↓(VARIABLE)
                               [NAME↑(IDENT)].CLASSE;
                              SE(CLASS1 <> VAR) OU
                                (CLASS1 <> INDEFC)
                              ENTAO
```

```
                                IMPRIME ERRO12;
                            FIM
                        SENAO /* ID NAO RECONHECIDO  */
                            INICIO
                            IMPRIME ERRO2;
                            TYPEVALOR↑ := INDEFT
                            FIM                                    >>
            'IDENT'     <<NAME↑ := IDENT                           >>
            ;
EXP         =           <</* VERIFICA SE O TIPO DE EXP SIM*/
                          SE NAO CONDICAO↑(EXP-RES)
                          ENTAO
                            TYPEVALOR↑ := TYPEVALOR↑(SEX)
                          SENAO
                            INICIO
                            TYPE1 := TYPEVALOR↑(SEX);
                            TYPE2 := TYPEVALOR↑(EXP-RES);
                            SE((TYPE1 = INT)E (TYPE2 = INT)
                              OU ((TYPE1 = BOOL) E
                                  (TYPE2 =BOOL))
                              OU ((TYPE1 = CHARACTER) E
                                  (TYPE2 = CHARACTER))
                            ENTAO
                              TYPEVALOR↑ := BOOL
                            SENAO /* VERIFICAR SE E INDEF*/
                              INICIO
                              SE (TYPE1 <> INDEFT) E
                                 (TYPE2 <> INDEFT)
                              ENTAO /* ERRO INCOMPAT */
                                 IMPRIME ERRO16;
                              TYPEVALOR↑ := INDEFT;
                              FIM                                  >>
            SEX         <<USED↓ := USED↓(EXP)                      >>
            EXP-RES     <<USED↓ := USED↓(EXP)                      >>
            ;
EXP-RES     =           <<TYPEVALOR↑ := TYPEVALOR↑(SEX);
                          CONDICAO↑ := VERDADEIRO                  >>
            OP-REL
            SEX         <<USED↓ := USED↓(EXP-RES)                  >>
            ;
EXP-RES     =           <<TYPEVALOR↑ := INDEFT;
                          CONDICAO↑ := FALSO                       >>
            ;
OP-REL      =
            '='
            ;
OP-REL      =
            '<>'
            ;
OP-REL      =
```

```
                '>'
                ;
OP-REL          =
                '<'
                ;
OP-REL          =
                '<='
                ;
OP-REL          =
                '>='
                ;
SEX             =           <</* VERIFICA-SE OS TIPOS DE EXP*/
                              SE CONDICAO↑(SEX-RES)
                              ENTAO /* VERIFICA-SE COMP. */
                                   TYPE1 := TYPEVALOR↑(TERM);
                                   TYPE2 := TYPEVALOR↑(SEX-RES);
                                   SE(TYPE1 = INT) E (TYPE2 = INT)
                                     E (OP↑(SEX-RES) = ARIT)
                                   ENTAO  /*    OK          */
                                       TYPEVALOR↑ := INT
                                   SENAO
                                       SE(TYPE1 = BOOL) E
                                         (TYPE2 = BOOL)
                                        E(OP↑(SEX-RES) = LOG)
                                       ENTAO /*     OK     */
                                           TYPEVALOR↑ := BOOL
                                       SENAO

                                           INICIO
                                           SE(TYPE1 <> INDEFT) E
                                             (TYPE2 <> INDEFT)
                                           ENTAO /* INCOMPATIB.*/
                                               IMPRIME ERRO17;
                                           TYPEVALOR↑ := INDEFT;
                                           FIM
                              SENAO
                                   TYPEVALOR↑ := TYPEVALOR↑(TERM)
                                                              >>
                TERM          <<USED↓ := USED↓(SEX)            >>
                SEX-RES       <<USED↓ := USED↓(SEX)            >>
                ;
SEX-RES         =           <<
                              CONDICAO↑:= VERDADEIRO;
                              OP↑ := OP↑(OP-ARIT);
                              SE CONDICAO↑(SEX-RES<0>)
                              ENTAO /* VERIFICA-SE COMPATIB*/
                                   TYPE1 := TYPEVALOR↑(TERM);
                                   TYPE2 :=
                                       TYPEVALOR↑(SEX-RES<0>);
                                   SE(TYPE1 = INT) E (TYPE2 = INT)
```

```
                                    E (OP↑(SEX-RES<0>) = ARIT)
                                  ENTAO  /*      OK    */
                                      TYPEVALOR↑ := INT
                                  SENAO
                                      SE(TYPE1 = BOOL) E
                                        (TYPE2 = BOOL)
                                       E (OP↑(SEX-RES<0>) = LOG)
                                      ENTAO /*    OK   */
                                          TYPEVALOR↑ := BOOL
                                      SENAO
                                          INICIO
                                          SE(TYPE1 <> INDEFT) E
                                            (TYPE2 <> INDEFT)
                                          ENTAO
                                              IMPRIME ERRO17;
                                          TYPEVALOR↑ := INDEFT;
                                          FIM
                              SENAO
                                  TYPEVALOR↑ :=
                                           TYPEVALOR↑(TERM)      >>
        OP-ARIT
        TERM          <<USED↓ := USED↓(SEX-RES)                  >>
        SEX-RES       <<USED↓ := USED↓(SEX-RES)                  >>
        ;
SEX-RES =             <<CONDICAO↑ := FALSO                      >>
        ;
OP-ARIT =             <<OP↑ := ARIT                             >>
        '+'
        ;
OP-ARIT =             <<OP↑ := ARIT                             >>
        '-'
        ;
OP-ARIT =             <<OP↑ := LOG                              >>
        'OR'
        ;
TERM    =              <<
                         SE CONDICAO↑(TERM-R)
                         ENTAO
                             INICIO
                             TYPE1 := TYPEVALOR↑(FATOR) ;
                             TYPE2 := TYPEVALOR↑(TERM-R);
                             SE(TYPE1 = INT) E (TYPE2 = INT)
                              E (OPI(TERM-R) = ARIT)
                             ENTAO /* OK    */
                                 TYPEVALOR↑ := INT
                             SENAO
                                 SE (TYPE1 = BOOL) E
                                   (TYPE2 = BOOL)
                                  E(OP↑(TERM-R) = LOG)
                                 ENTAO
```

```
                                        TYPEVALOR↑ := BOOL
                                    SENAO /* VERIFICA SE IND*/
                                        INICIO
                                        SE(TYPE1 <> INDEFT) E
                                          (TYPE2 <> INDEFT)
                                        ENTAO /* INCOMPATIB.*/
                                            IMPRIME ERRO17;
                                        TYPEVALOR↑ := INDEFT
                                        FIM
                                SENAO
                                    TYPEVALOR↑ := TYPEVALOR↑(FATOR)
                                                                >>
        FATOR       <<USED↓ := USED↓(TERM)                      >>
        TERM-R      <<USED↓ := USED↓(TTERM)                     >>
        ;
TERM-R  =           <<
                    OP↑ := OP↑(OP-MULT);
                    CONDICAO↑ := VERDADEIRO;
                    SE CONDICAO↑(TERM-R<0>)
                    ENTAO
                        INICIO
                        TYPE1 := TYPEVALOR↑(FATOR);
                        TYPE2 := TYPEVALOR↑(TERM-R<0>);
                        SE (TYPE1 = INT) E
                           (TYPE2 = INT) E
                           (OP↑(TERM-R<0>) = ARIT)
                        ENTAO /*    OK    */
                            TYPEVALOR↑ := INT
                        SENAO
                            SE(TYPE1 = BOOL) E
                              (TYPE2 = BOOL) E
                              (OP↑(TERM-R<0>) = LOG)
                            ENTAO
                                TYPEVALOR↑ := BOOL
                            SENAO /* VERIFICAR SE IND*/
                                INICIO

                                SE(TYPE1 <> INDEFT) E
                                  (TYPE2 <> INDEFT)
                                ENTAO /* INCOMPAT.*/
                                    IMPRIME ERRO17;
                                TYPEVALOR↑ := INDEFT
                                FIM
                     SENAO
                         TYPEVALOR↑ := TYPEVALOR↑(FATOR)
                                                        >>
        OP-MULT
        FATOR       <<USED↓ := USED↓(TERM-R)                    >>
        TERM-R      <<USED↓ := USED↓(TERM-R)                    >>
        ;
```

```
TERM-R      =           <<CONDICAO↑ := FALSO                    >>
            ;
OP-MULT     =           <<OP↑ := ARIT                           >>
            '*'
            ;
OP-MULT     =           <<OP↑ := ARIT                           >>
            '/'
            ;
OP-MULT     =           <<OP↑ := LOG                            >>
            'AND'
            ;
FATOR       =           << /* VERIFICA-SE A CLASSE DO ID*/
                         CLASS1 := CLASSVALOR↑(IDENT-FAT);
                         SE(CLASS1 = VAR) OU
                           (CLASS1 = CONST) OU
                           (CLASS1 = INDEFC)
                         ENTAO /*    OK    */
                              TYPEVALOR↑ :=
                                   TYPEVALOR↑(IDENT-FAT)
                         SENAO    /*     ERRO    */
                              INICIO
                              IMPRIME ERRO12;
                              TYPEVALOR↑ := INDEFT
                              FIM                               >>
            IDENT-FAT    <<USED↓ := USED↓(FATOR)                 >>
            ;
FATOR       =           << /*  CONSTANTE        */
                         TYPEVALOR↑ :=
                              TYPES↑(CONSTANT-FA).ORDINAL       >>
            CONSTANT-FA
            ;
FATOR       =           <<TYPEVALOR↑ := TYPEVALOR↑(EXP)         >>
            '('
            EXP          <<USED↓ := USED↓(FATOR)                 >>
            ')'
            ;
IDENT-FAT   =           <</* CONSEGUIR A CLASSE DO ID   */
                         SE USED↓(IDENT-FAT)[NAME↑(IDENT)]
                              <> INDEFINIDO
                         ENTAO    /* OBTER CLASSE    */
                              INICIO
                              CLASSVALOR↑ :=USED↓(IDENT-FAT)
                                [NAME↑(IDENT)].CLASSE;
                              TYPEVALOR↑ := USED↓(IDENT-FAT)
                               [NAME↑(IDENT)].TYPEP.ORDINAL
                              FIM
                         SENAO /* ERRO ID NAO RECONHECIDO*/
                              INICIO
                              CLASSVALOR↑ := INDEFC;
                              TYPEVALOR↑ := INDEFT;
```

```
                              IMPRIME ERRO2
                              FIM                                    >>
            'IDENT'     <<NAME↑ := IDENT                              >>
            ;
CONSTANT    =           <</* TRANSMITE O TIPO DA CONST */
                          TYPES↑ := TYPES↑(CONSTANT-ID)              >>
            CONSTANT-ID<<USED↓ := USED↓(CONSTANT)                    >>
            ;
CONSTANT    =           <<TYPES↑ := TYPES↑(CONSTANT-FA)              >>
            CONSTANT-FA
            ;
CONSTANT-FA =           <<
                          TYPES↑.ORDINAL := CHARACTER;
                          TYPES↑.RANGO := SIMPLES;
                          TYPES↑.VALOR := NUMERO↑(CHAR)              >>
            '''
            'CHAR'      <<NUMERO↑ := CHAR                            >>
            '''
            ;
CONSTANT-FA =           <</* DAR O TIPO PARA INTEIRO     */
                          TYPES↑.ORDINAL := INT;
                          TYPES↑.RANGO := SIMPLES;
                          TYPES↑.VALOR :=
                              NUMERO↑(UNSIG-CONS)                    >>
            'UNSIG-CONS'<<NUMERO↑ := UNSIG-CONS                      >>
            ;
CONSTANT-ID =           <</* VERIFICA-SE A CLASSE DO ID */
                          SE USED↓(CONSTANT-ID)
                            [NAME↑(IDENT)] <> INDEFINIDO
                          ENTAO /* VERIFICA-SE CONST     */
                              INICIO
                              CLASS1 := USED↓(CONSTANT-ID)
                                [NAME↑(IDENT)].CLASSE;
                              SE (CLASS1 = CONST) OU
                                 (CLASS1 = INDEFC)

                              ENTAO  /*  OK     */
                                   TYPES↑ := USED↓(CONSTANT)
                                     [NAME↑(IDENT)].TYPEP
                              SENAO   /*   ERRO CLASSE  */
                                   INICIO
                                   IMPRIME ERRO12;
                                   TYPES↑.ORDINAL := INDEFT
                                   FIM
                              FIM
                          SENAO    /* ID NAO RECONHECIDO*/
                              INICIO
                              TYPES↑.ORDINAL := INDEFT
                              IMPRIME ERRO2
                              FIM                                    >>
```

```
            'IDENT'     <<NAME↑ := IDENT                          >>
            ;
LISTA-CON   =           <</* LISTA DE NOMES */
                          SEQNOME↑ := SEQNOME↑(LISTA-CON<0>)
                                    ^ [*NAME↑(IDENT)*]            >>
            ','
            'IDENT'     <<NAME↑ := IDENT                          >>
            LISTA-CON
            ;
LISTA-CON   =           <<SEQNOME↑ := [* *]                       >>
            ;
LISTA-ID    =           <<SEQNOME↑ := SEQNOME↑(LISTA-CON)
                                  ^ [*NAME↑(IDENT)*]              >>
            'IDENT'     <<NAME↑ := IDENT                          >>
            LISTA-CON
            ;
  FIM.
```

V.2 - A Linguagem $\{a^n b^n c^n / n \geq 1\}$

A linguagem $L = \{a^n b^n c^n / n \geq 1\}$ |CRAIG|[5] é uma linguagem sensível ao contexto ou tipo 1, que pode ser definido pela gramática sensível ao contexto G seguinte:

$G = (N, \Sigma, P, S)$ /

$N = \{S, T, B, C, D\}$

$\Sigma = \{a, b, c\}$

$P = \{S \rightarrow T,$

$T \rightarrow a\,TBD,$

$T \rightarrow a\,bD,$

$DB \rightarrow CB$

$CB \rightarrow CD,$

$CD \rightarrow BD,$

$bB \rightarrow bb,$

$D \rightarrow c\ \}$

Uma definição da linguagem L usando gramática de atributos será:

```
GRAMATICA ABC;
TEXTO
      ERRO = 'NAO E UMA CADEIA DA LINGUAGEM';
ATRIBUTO
      /*                                                          */
      /* CONTA                                                    */
      /*       CONTA CONTADOR DE CARACTERES DA CADEIA             */
      /* TOTAL                                                    */
      /*       NUMERO TOTAL DE CARACTERES DA CADEIA               */
      /*                                                          */
      CONTA↓,TOTAL↑ : INTEIRO;
SIMBOLONT
      S : TOTAL;
      A,B,C : CONTA,TOTAL;
      A-RESTO,B-RESTO,C-RESTO : CONTA,TOTAL;
PRODUCOES

   S        =        << /* VERIFICA-SE A CADEIA E DA LING.*/
                       SE NAO ((TOTAL↑(A) = TOTAL↑(B)  E
                                (TOTAL↑(A) = TOTAL↑(C))
                       ENTAO
                           IMPRIME ERRO;
                       TOTAL↑ := TOTAL↑(A)                  >>
            A        << CONTA↓ := 0                          >>
            B        << CONTA↓ := 0                          >>
            C        << CONTA↓ := 0                          >>
            ;

   A        =        << TOTAL↑ := TOTAL↑(A-RESTO)            >>
            'A'
            A-RESTO  << CONTA↓ := CONTA↓(A) + 1              >>
            ;

   A-RESTO  =        << TOTAL↑ := TOTAL↑(A)                  >>
            A        << CONTA↓ := CONTA↓(A-RESTO)            >>
            ;

   A-RESTO  =        << TOTAL↑ := CONTA↓(A-RESTO)            >>
            ;

   B        =        << TOTAL↑ := TOTAL↑(B-RESTO)            >>
            'B'
            B-RESTO  << CONTA↓ := CONTA↓(B) + 1              >>
            ;

   B-RESTO  =        << TOTAL↑ := TOTAL↑(B)                  >>
            B        << CONTA↓ := CONTA↓(B-RESTO)            >>
            ;

   B-RESTO  =        << TOTAL↑ := CONTA↓(B-RESTO)            >>
```

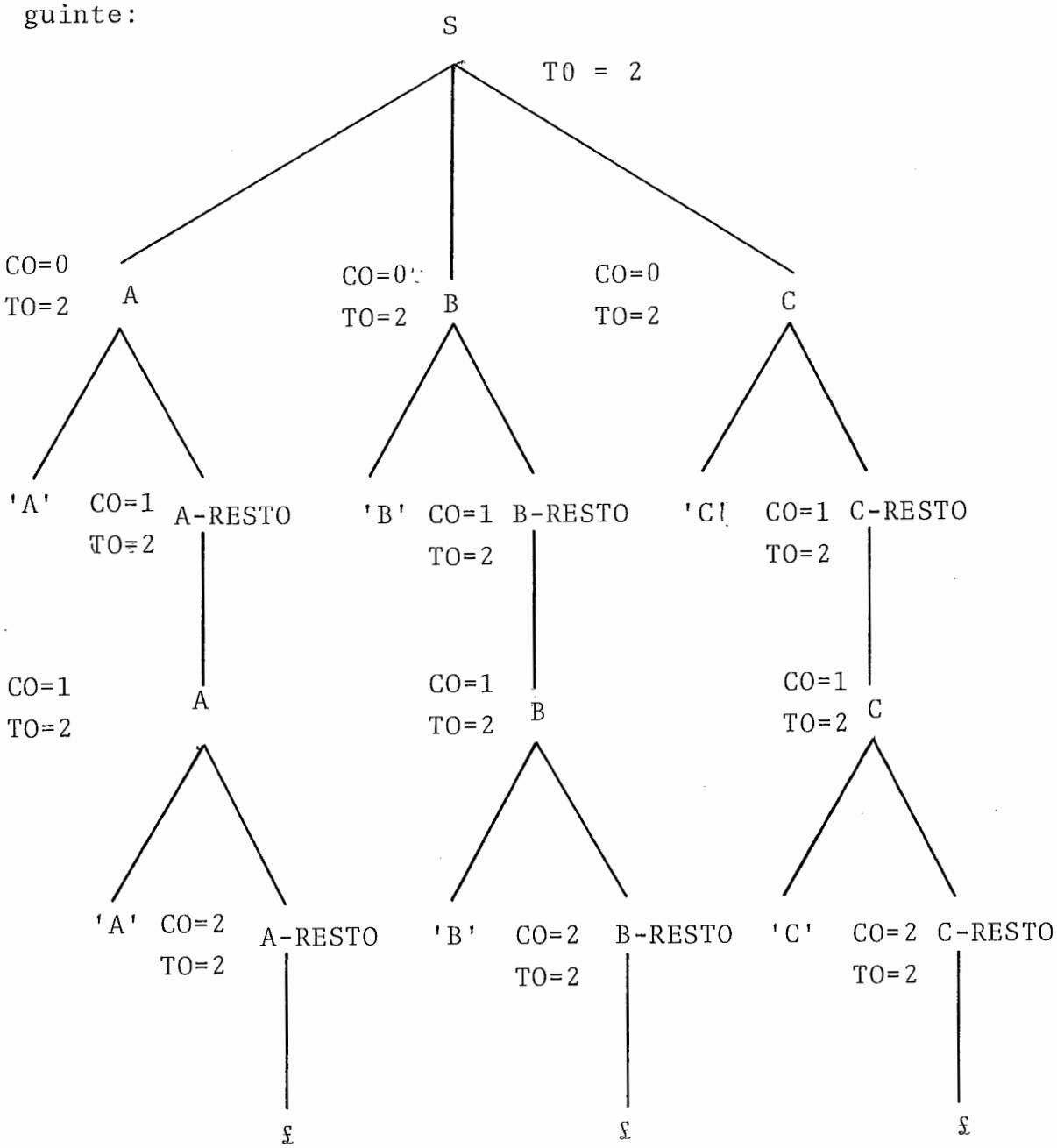
```
            ;

    C       =       << TOTAL↑ := TOTAL↑(C-RESTO)          >>
            'C'
            C-RESTO << CONTA↓ := CONTA↓(C) + 1            >>
            ;

    C-RESTO =       << TOTAL↑ := TOTAL↑(C)                >>
            C       << CONTA↓ := CONTA↓(C-RESTO)          >>
            ;

    C-RESTO =       << TOTAL↑ := CONTA↓(C-RESTO)          >>
            ;
FIM.
```

Uma árvore de derivação com seus atributos para a cadeia $a^2b^2c^2$, quando aplicado o método de cálculo UPED é a seguinte:



TO = TOTAL ↑

CO = CONTA ↓

CAPÍTULO VI

CONCLUSÕES

A transformação de uma árvore sintática numa árvore sintática abstrata ("abstract syntax tree"), segundo McKeeman[23], é um dos passos do compilador.

Na aplicação em geração automática de compiladores, um ponto por resolver foi a análise de semântica estática como um passo anterior à geração desta árvore, e sobre o qual se centralizou o tema.

A gramática de atributos de Knuth tem sido aplicada à definição formal da semântica e geração de códigos. Alguns geradores de calculadores de funções semânticas tem sido propostos |LORHO|[16], |KENNEDY|[13], |KASTENS|[12]. Os métodos de cálculo mais gerais necessitam manter toda a árvore de derivação durante a análise.

A gramática de atributos UPED contida na classe de gramática de atributos de Bochmann[3] implica em condições na definição das funções semânticas, porém o calculador UPED requer a árvore de derivação apenas parcialmente durante a análise, como é determinado pelo percurso e a ordem de cálculo.

Na linguagem de especificação para a gramática de atributos UPED, os tipos de dados e comandos foram estudados para facilitar a definição da semântica estática de linguagens de programação. Outro tipo de aplicações são orientados à tradução, definição de linguagens sensíveis ao contexto, etc., como é mostrado nos exemplos.

Um calculador para a gramática de atributos UPED pode ser construído resolvendo a organização das pilhas de atributos.

Algumas idéias para desenvolvimento futuro são apresentadas a seguir.

1. Ampliações para a linguagem LEGA

a) Definição de tipos de dados que facilitem a especificação de conversão de código.

b) Declaração de procedimentos para facilitar a especificação das funções semânticas.

c) Definição de uma notação simplificada para o caso de identidades, isto é, para atribuições dos mesmos valores de atributos sintetizados (herdados) associados ao símbolo sintático da parte direita (esquerda) de uma produção, para os associados ao símbolo sintático da parte esquerda (direita) |KASTENS|[12].

No exemplo III.2.1 tem-se uma identidade na produção nº 5 da gramática de atributos.

2. Aplicação em construção automática de compiladores

Construção de um tradutor que aceite como entrada a especificação de uma gramática de atributos na linguagem LEGA e entregue como saída, um programa em PASCAL.

As instruções no programa PASCAL calculam os valores de atributos e resolvem a administração das pilhas dos atributos.

O tradutor deve verificar as condições para escrever as funções semânticas para a gramática de atributos UPED.

A gramática livre de contexto da gramática de atributos deve ser LL(1), por que é possível conseguir um gerador de analisador sintático LL(1). Usando o analisador sintático obtém-se a sequência de produções da derivação mais a esquerda, o que é requerido no cálculo das funções semânticas.

Na aplicação de um outro tipo de gerador de analisador sintático, será necessária uma transformação intermediária para se conseguir a sequência de produções da derivação mais à esquerda.

BIBLIOGRAFIA

1. AHO, Alfred, ULLMAN, Jeffrey - Principles of Compiler Design, Addison Wesley, 1977.

2. AHO, Alfred, ULLMAN Jeffrey - The Theory of Parsing, Translation and Compiling, Prentice-Hall, Vol. 1, 1972.

3. BOCHMANN,G. V. - Semantic Evaluation from Left to Right, CACM, Vol. 19, (55-62), 1976.

4. BOCHMANN, G. V., WARD, P. - Compiler Writing System for Attribute Grammars, The Computer Journal 21, (144-148), Maio 1978.

5. CRAIG, J., CLEAVELAND, UZGALIS - Grammars for Programming Languages, Elsevier North Holland, 1977.

6. GRIES, David - Compiler Construction for Digital Computer, John Wiley Sons, 1976.

7. GRIFFITHS, M. - LL(1) Grammars and Analysers, Laboratoire d' Informatique, Université de Grenoble, France, Compiler Construction, Bauer & Eickel, (55-82), 1976.

8. HOARE, C., DIJKSTRA, E. W., DAHL O. J. - Structured Programming, Academic Press, 1972.

9. HOPCROFT, John, ULLMAN, Jeffrey - Formal Languages and Their Relation to Automata, Addison Wesley, 1969.

10. JAZAYERI, M., WALTER, K. - Alternating Semantic Evaluator, Proceedings of the ACM, (230-234), Outubro 1975.

11. JENSEN, Kathleen, WIRTH, Niklaus - Pascal, User Manual and Report, Springer-Verlag, 1978.

12. KASTENS - Ordered Attributed Grammar, Acta Informatica, Springer-Verlag, (229-256), 1980.

13. KENNEDY, Ken - Automatic Generation of Efficient Evaluators for Attribute Grammars, Conference Record of the Third ACM, Janeiro 1976.

14. KNUTH, Donald - Semantics of Context Free Languages, Mathematical Systems Theory Journal, Vol. 2, (127-145), 1968.

15. LEWIS, P. M., ROSENKRANT, D. J., STEARNS, R. E. - Compiler Design Theory, Addison Wesley, 1976.

16. LORHO, Bernard - Semantic Attributes Processing in the System Delta, Methods of Algorithmic Language, Lecture Notes in Computer Science, Springer-Verlag, Vol. 17, (21-40), 1977.

17. PRATT, Terrence - Programming Languages Design and Implementation, Prentice-Hall, 1975

18. RAIHA, K. J. - Bibliography on Attribute Grammars, University of Helsinki, Finland, Sigplan Notices, Vol. 15 N (3), Março 1980.

19. SETHI, R., ULLMAN, J. D. - The Generation of Optimal Code for Arithmetic Expressions, Journal of the ACM, (715-728), 1970.

20. VAN, Winjgaarden - Revised Report on the Algorithmic Language Algol 68, Acta Informatica, Vol. 5, 1975.

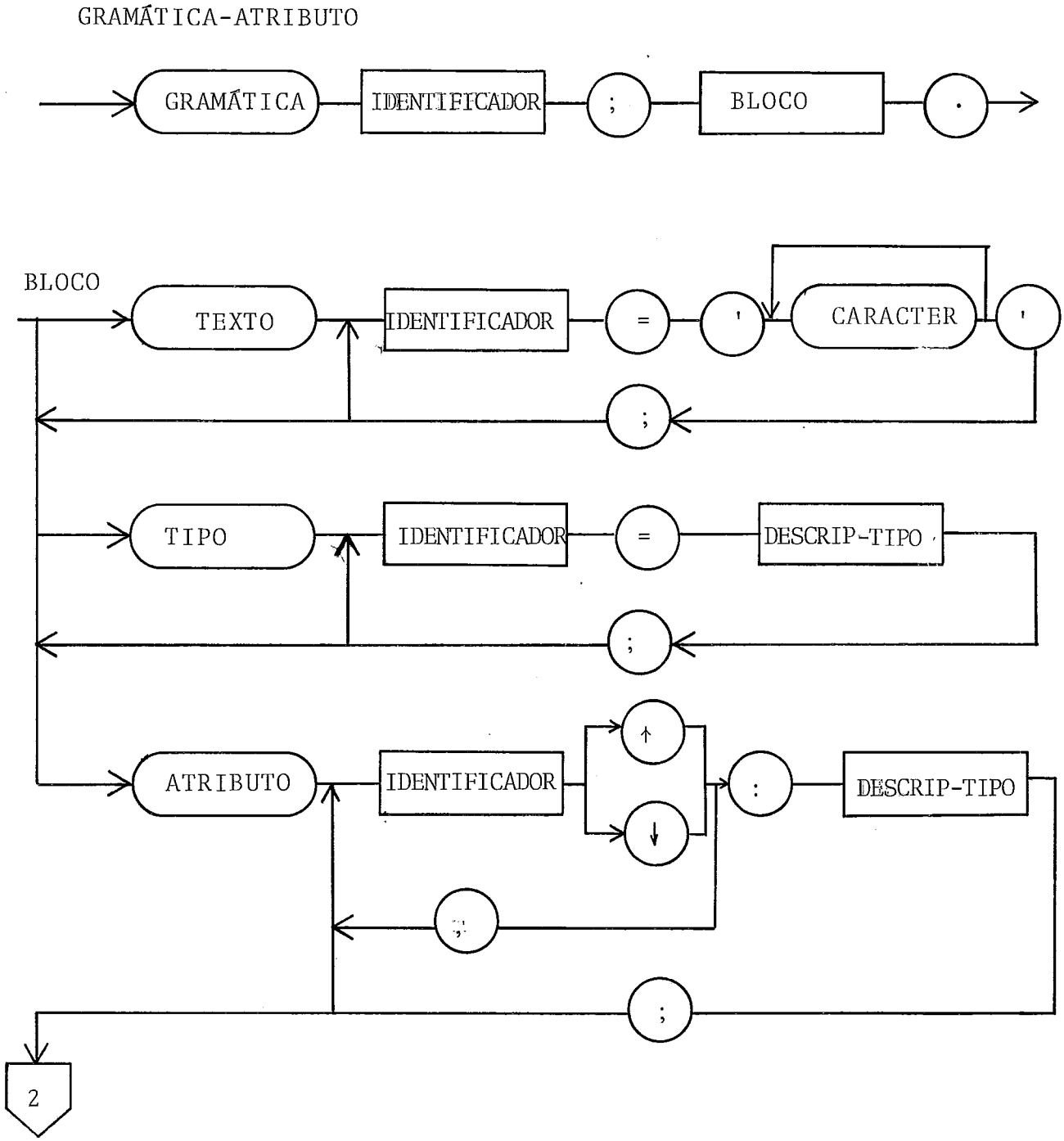
21. WATT, David - An Extended Attributed Grammar for Pascal, Sigplan Notices, (60-74), Fev. 1977.

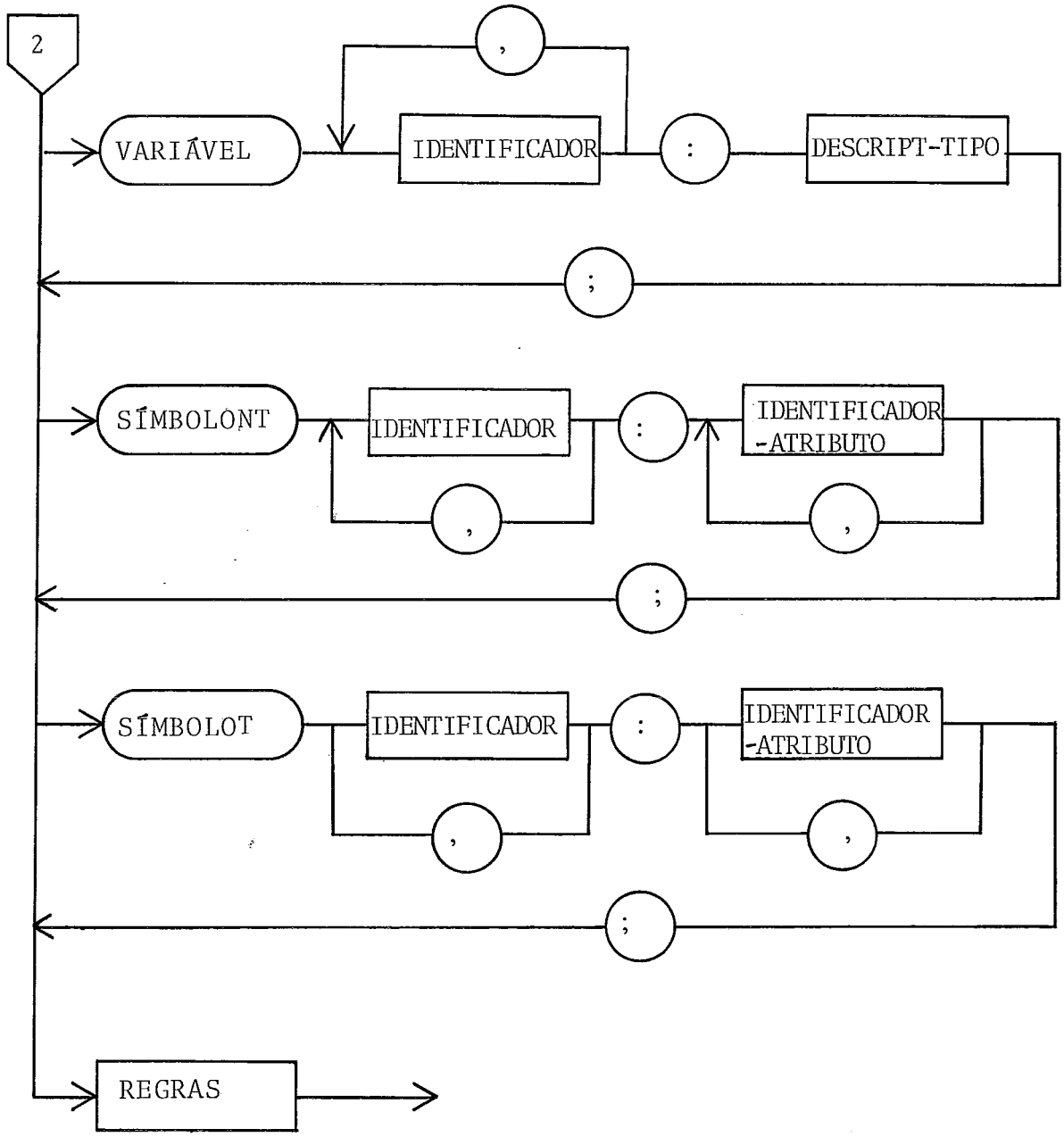
22. WATT, David, MADSEN, Ole - Extended Attributed Grammar, Report Nº 10, Computing Department, University of Glasgow, Julio 1975.

23. McKEEMAN, W. M. Compiler Construction, University of California, U.S.A. Compiler Construction, Bauer & Eickel, (1-36), 1976.

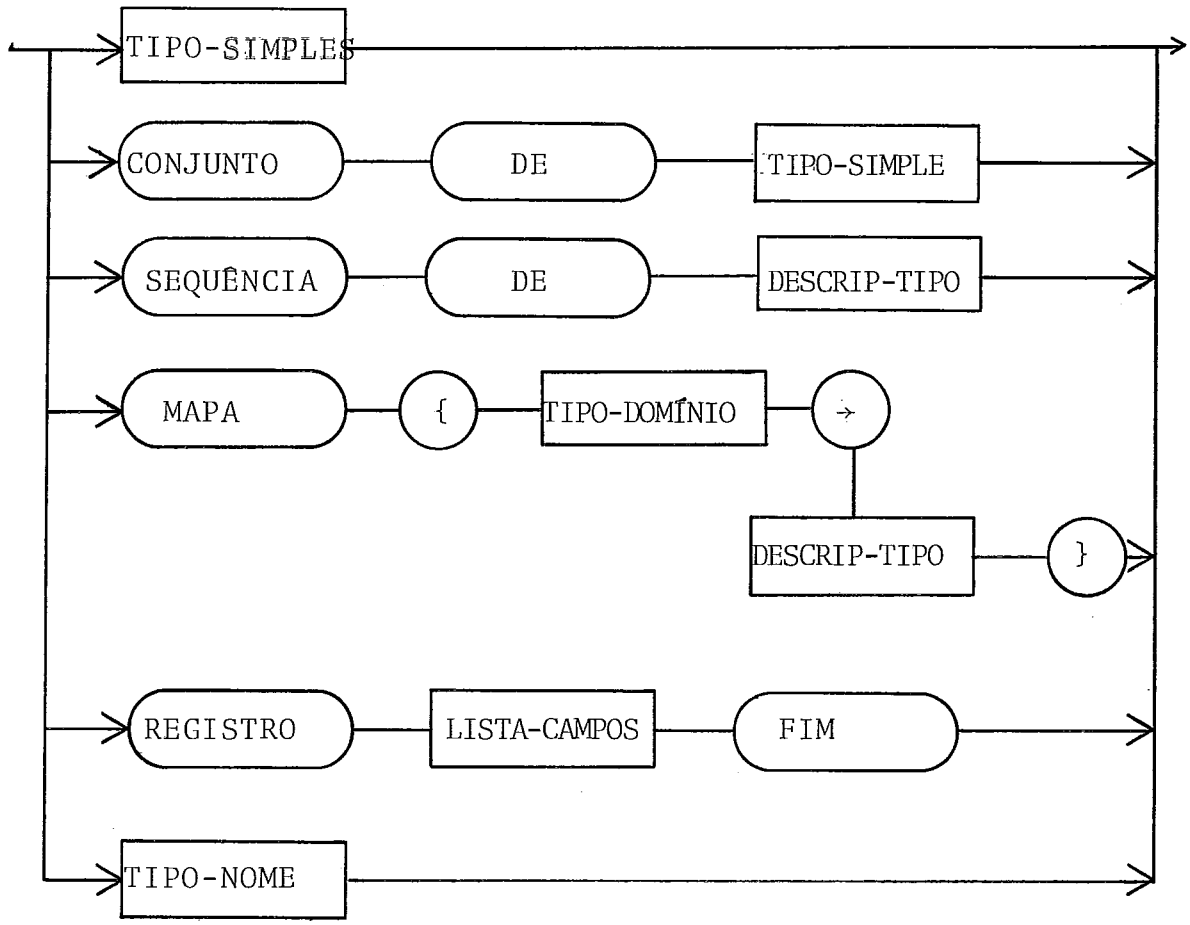
APÊNDICE A

DIAGRAMA DE SINTAXE DA LINGUAGEM LEGA

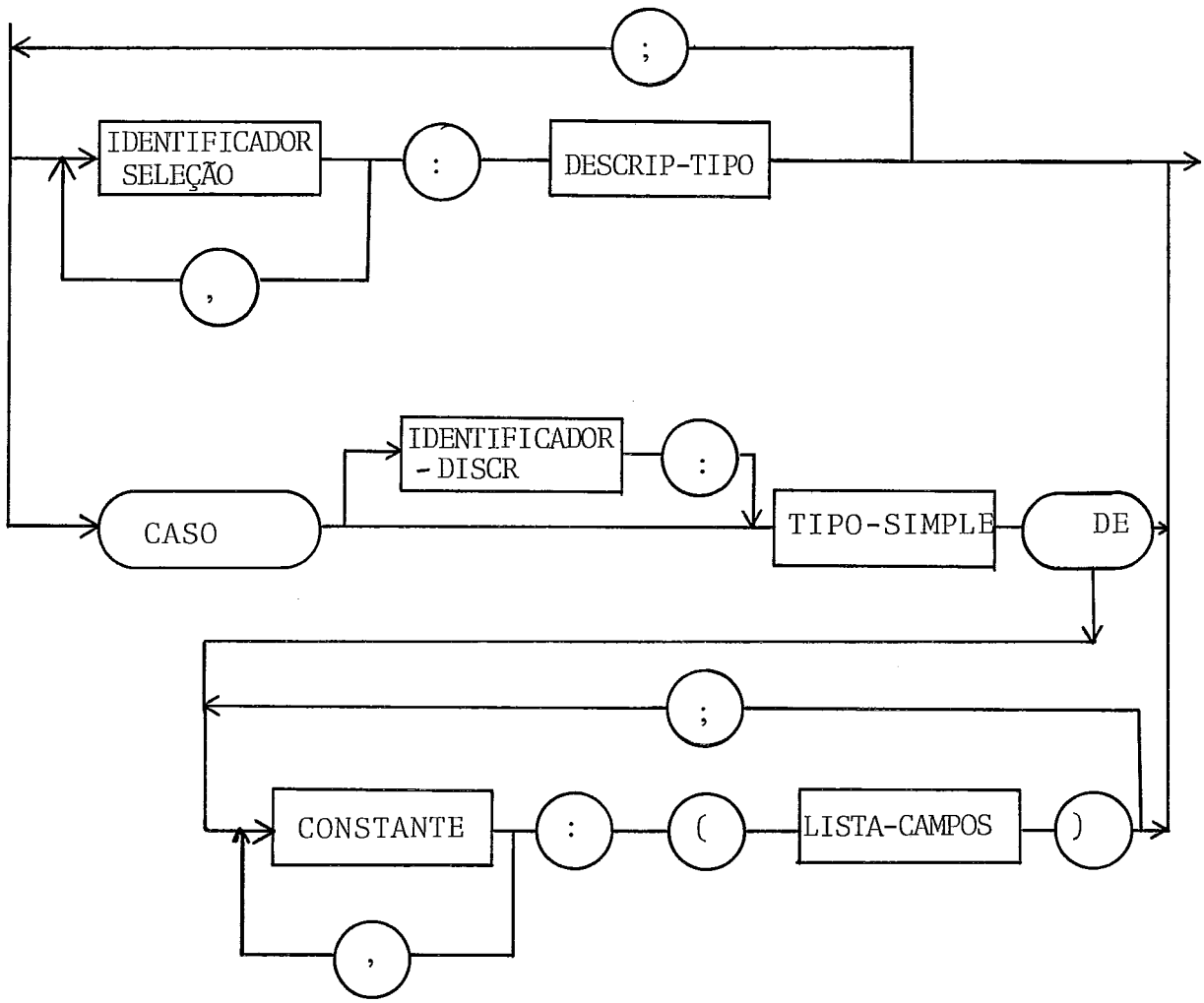



2
VARIÁVEL
IDENTIFICADOR
,
:
DESCRIPT-TIPO
;
SÍMBOLONT
IDENTIFICADOR
,
:
IDENTIFICADOR
-ATRIBUTO
,
;
SÍMBOLOT
IDENTIFICADOR
,
:
IDENTIFICADOR
-ATRIBUTO
,
;
REGRAS

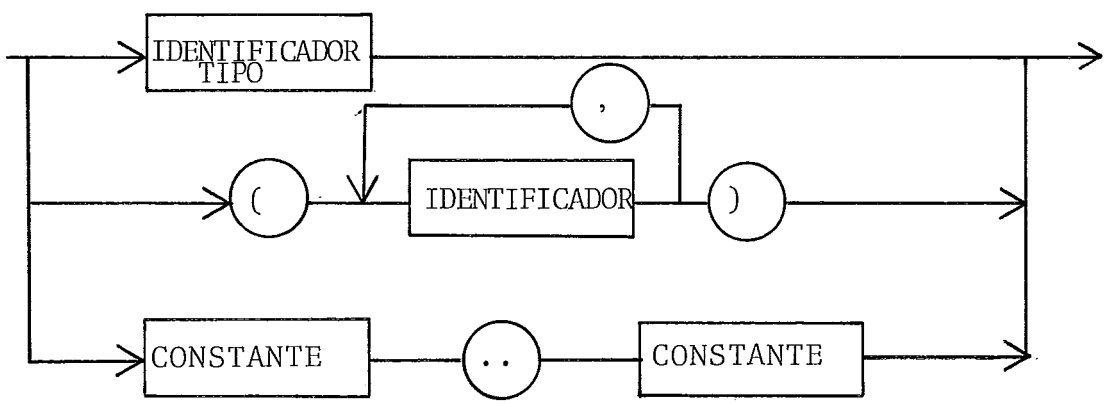
DESCRIP-TIPO



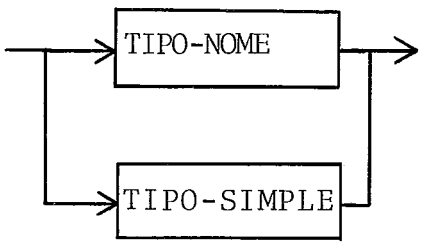
LISTA CAMPOS



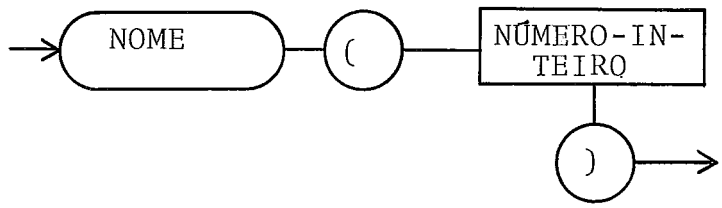
TIPO-SIMPLE



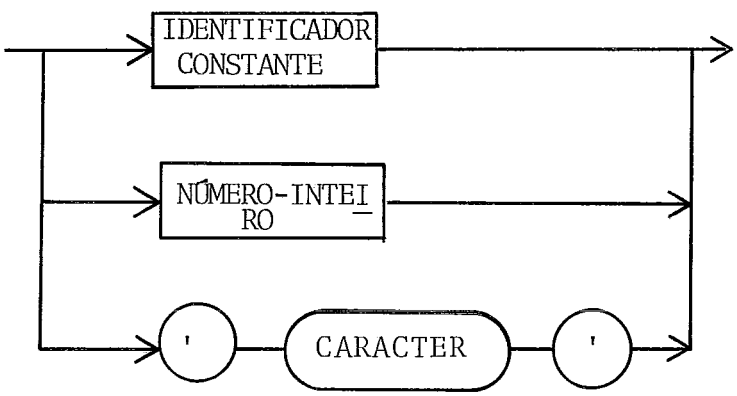
TIPO DOMÍNIO



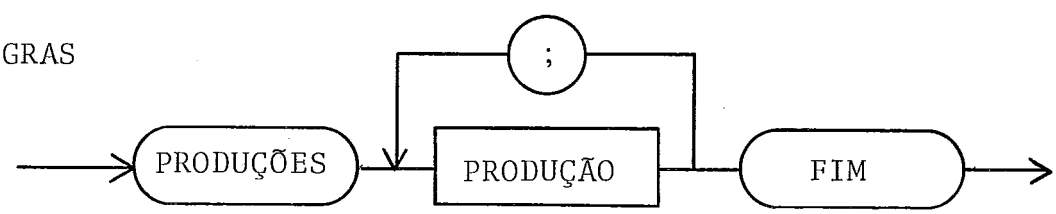
TIPO NOME



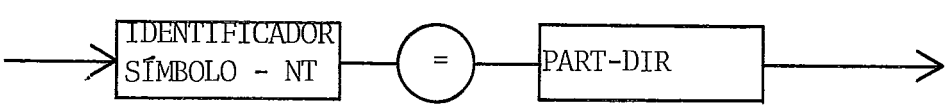
CONSTANTE



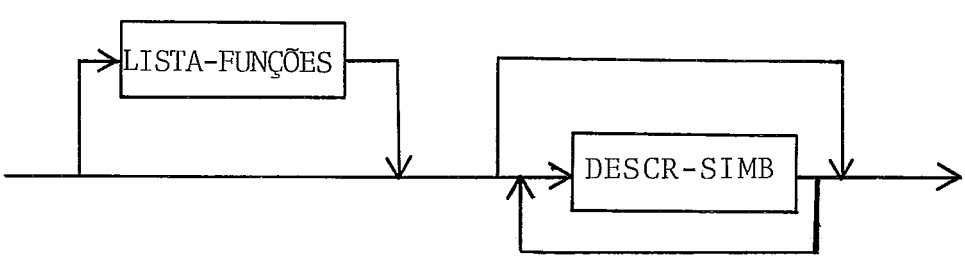
REGRAS



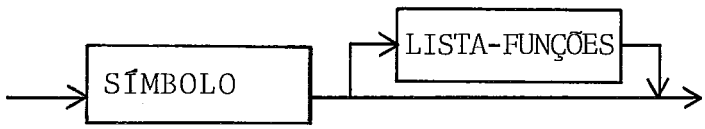
PRODUÇÃO



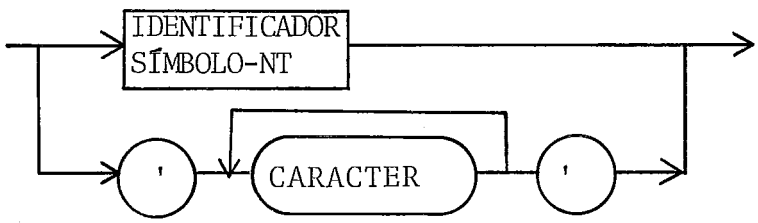
PART-DIR



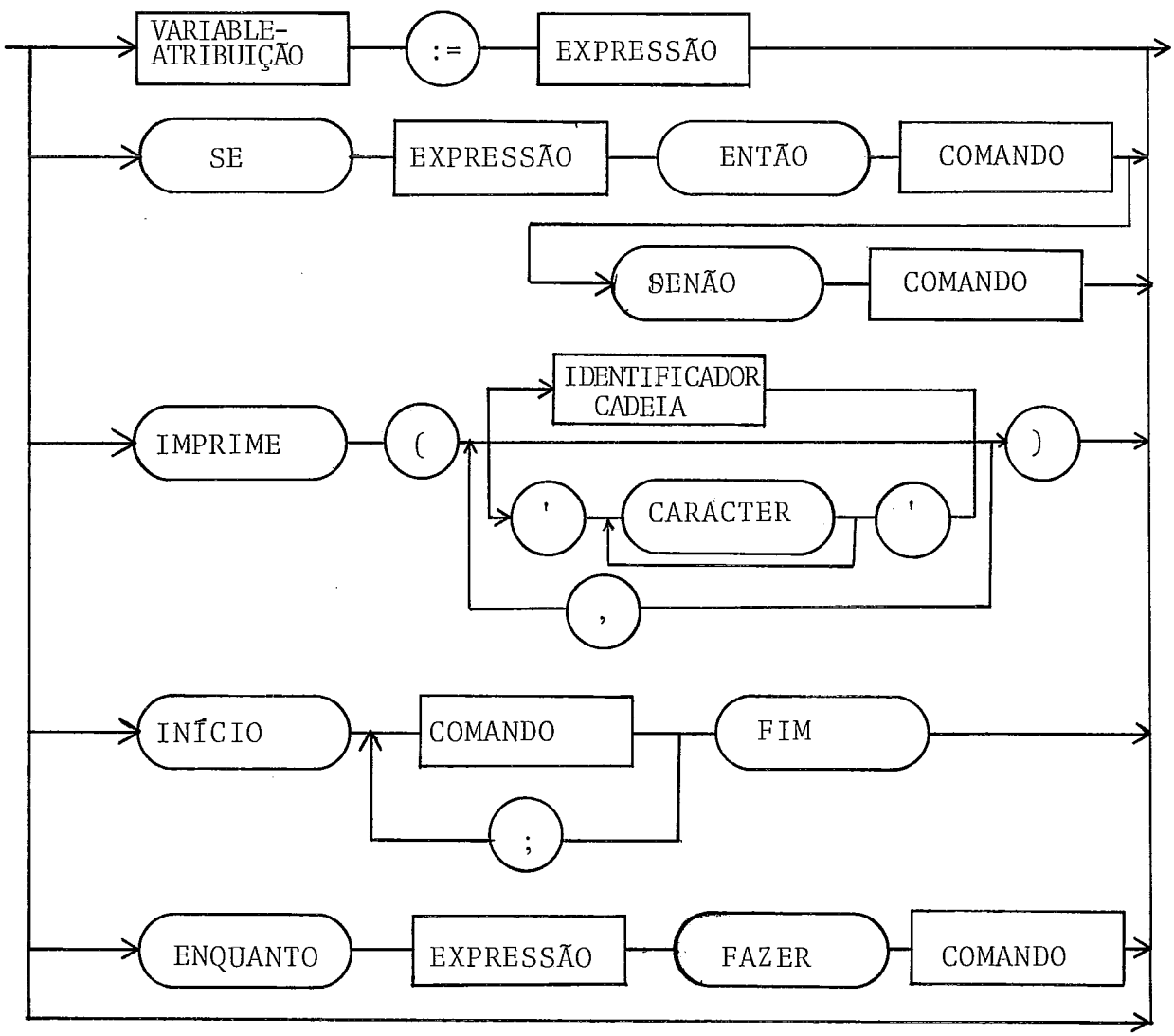
DESCR-SIMB



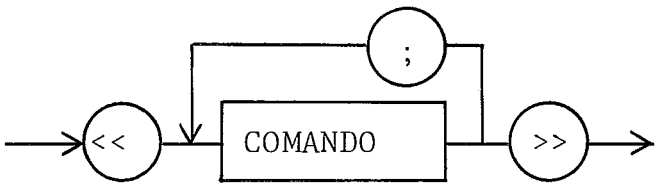
SÍMBOLO



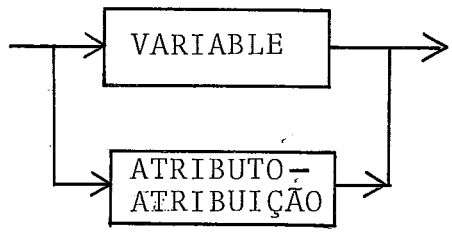
COMANDO



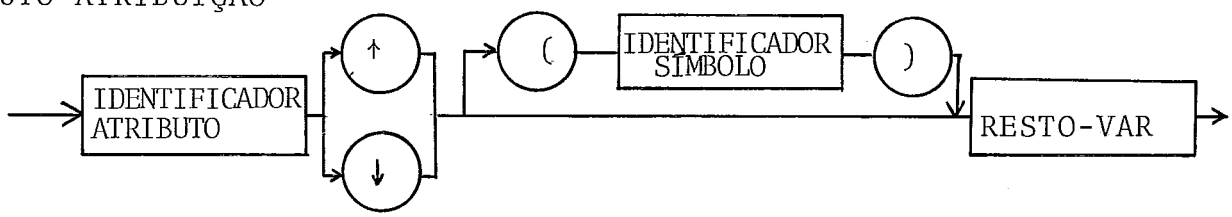
LISTA-FUNÇÕES



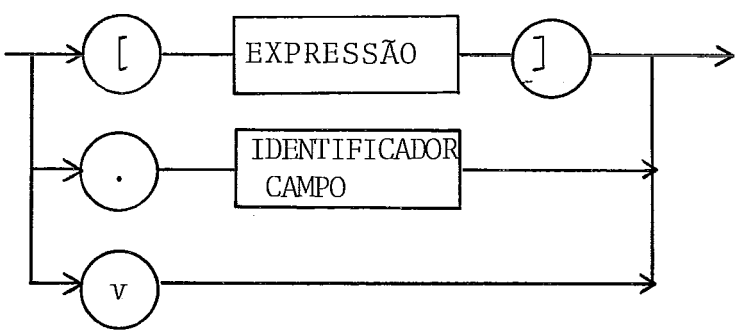
VARIABLE-ATRIBUIÇÃO



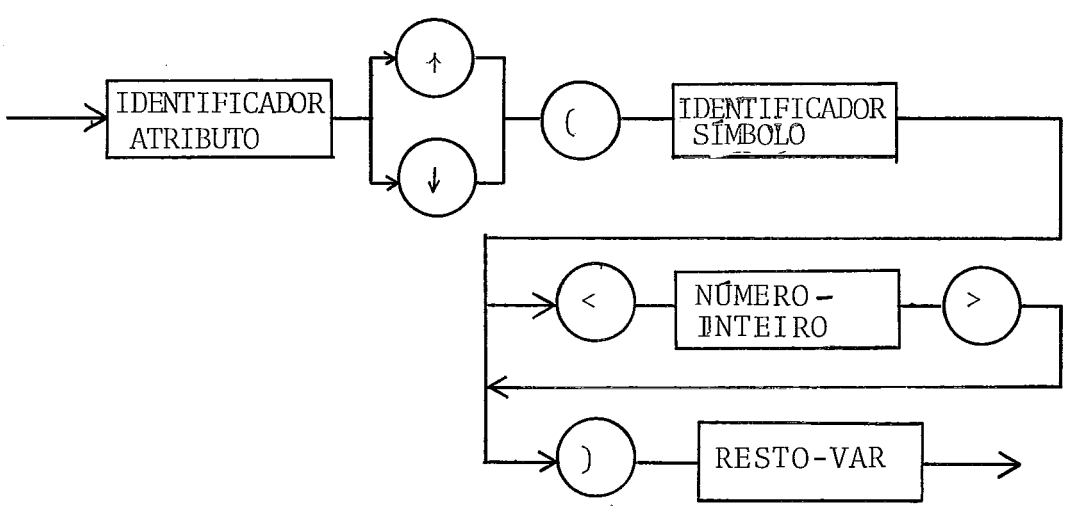
ATRIBUTO-ATRIBUIÇÃO



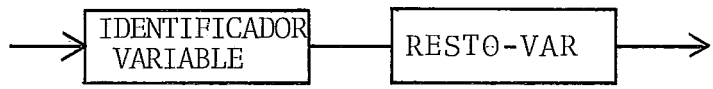
RESTO-VAR



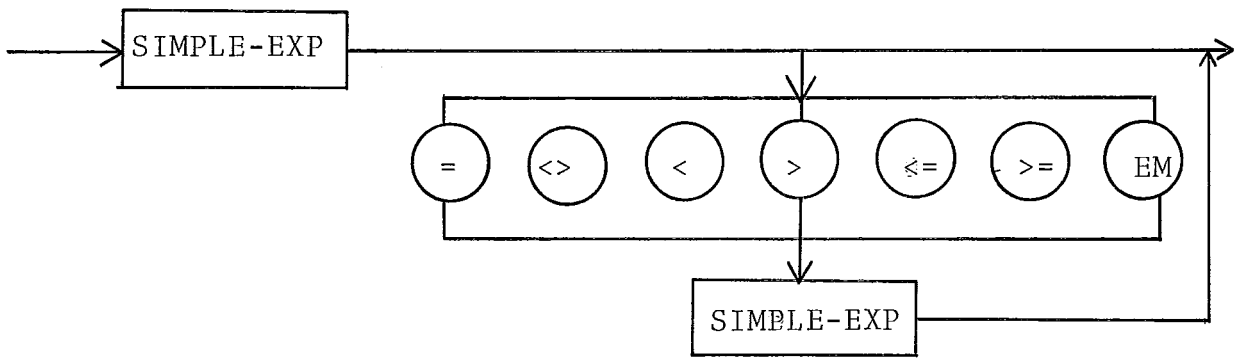
ATRIBUTO



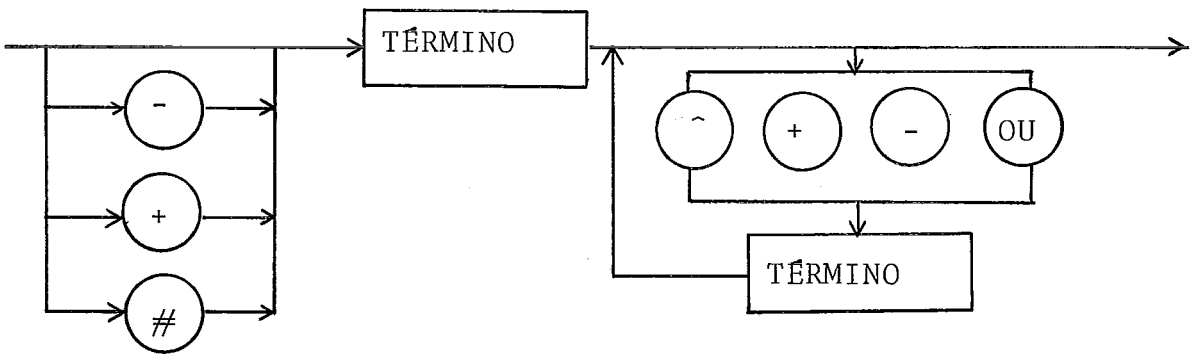
VARIABLE



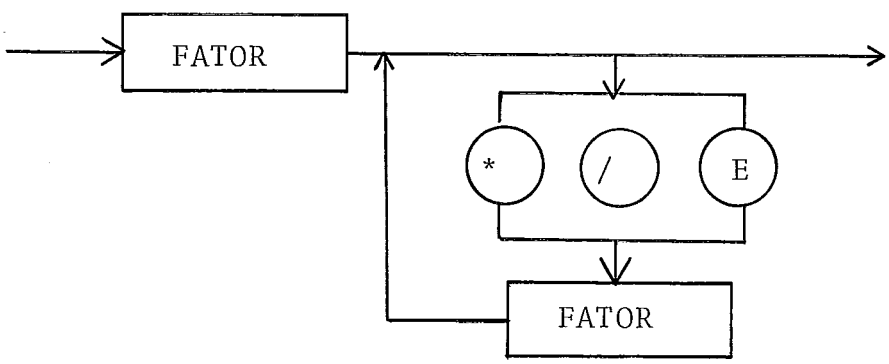
EXPRESSÃO



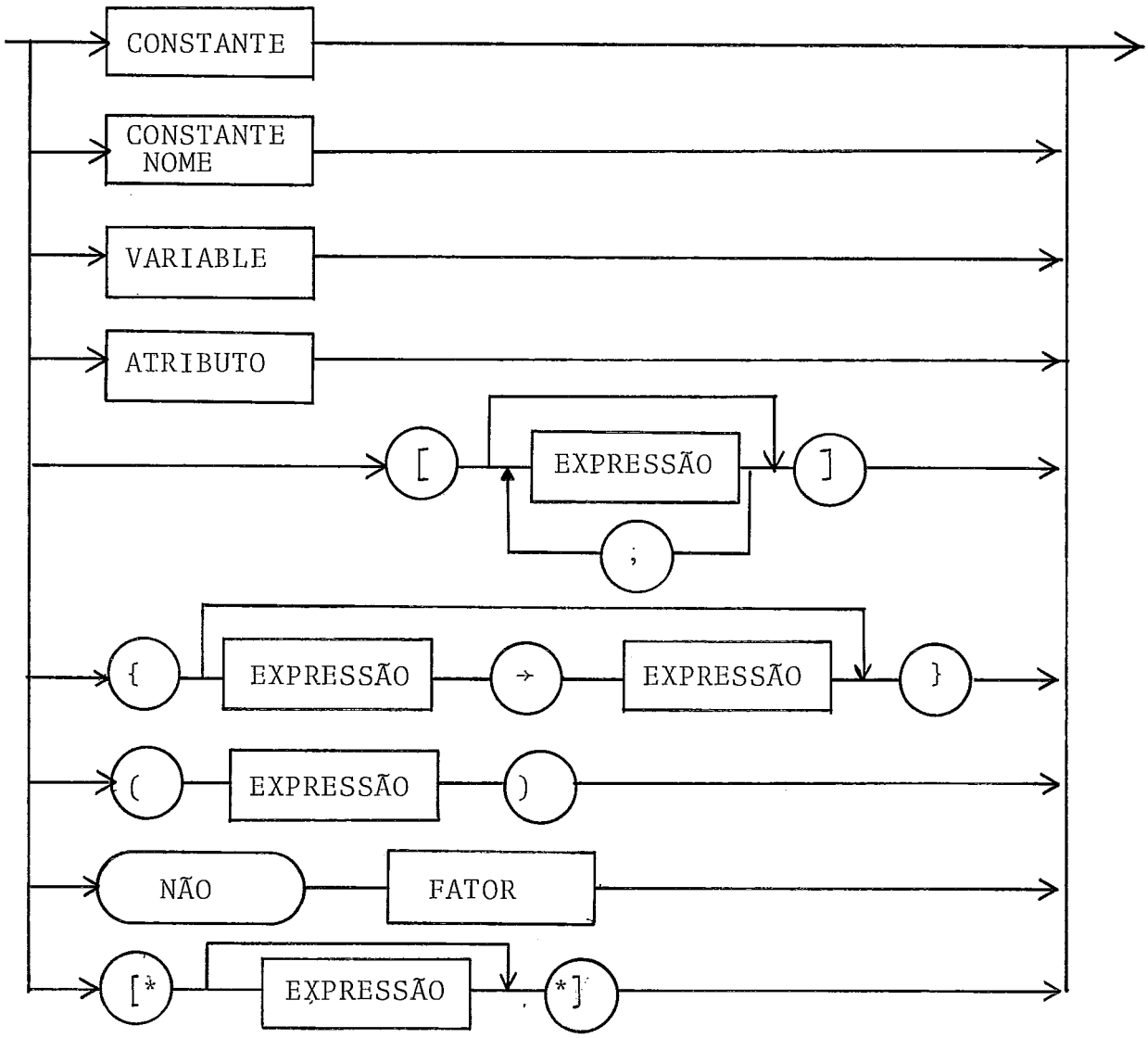
SIMPLE-EXP



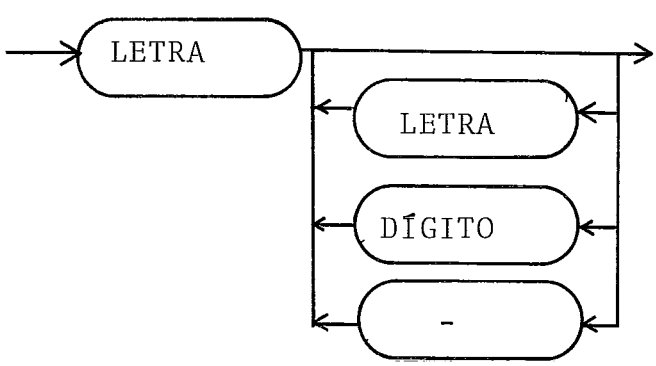
TÉRMINO



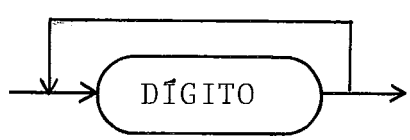
FATOR



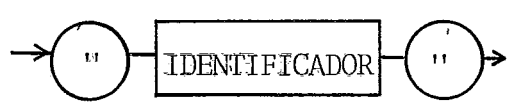
IDENTIFICADOR



NÚMERO-INTEIRO



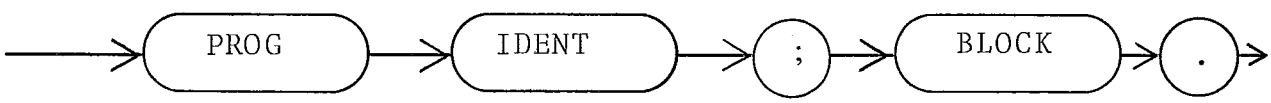
CONSTANTE-NOME



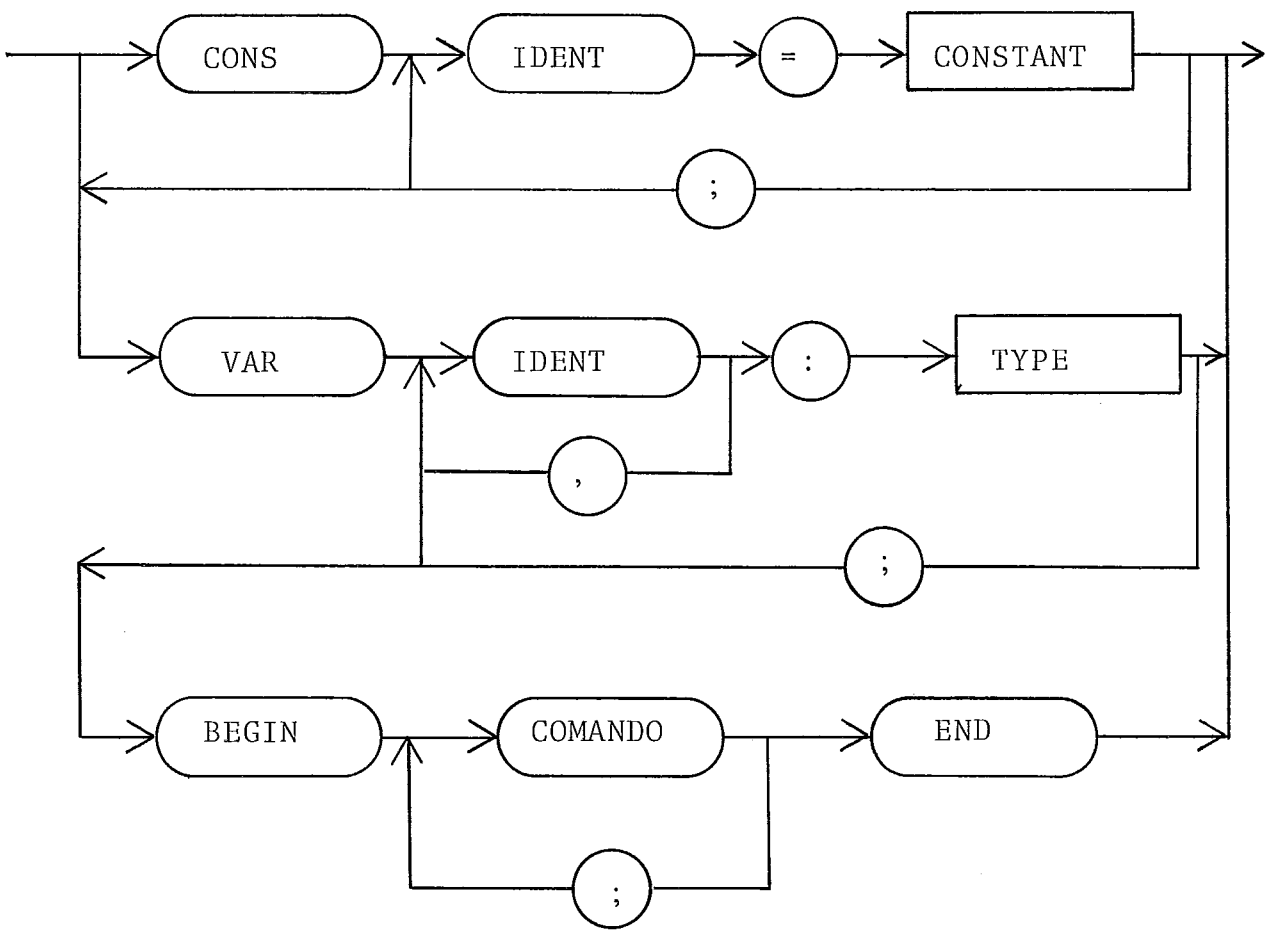
APÊNDICE B

DIAGRAMA DE SINTAXE DA LINGUAGEM S

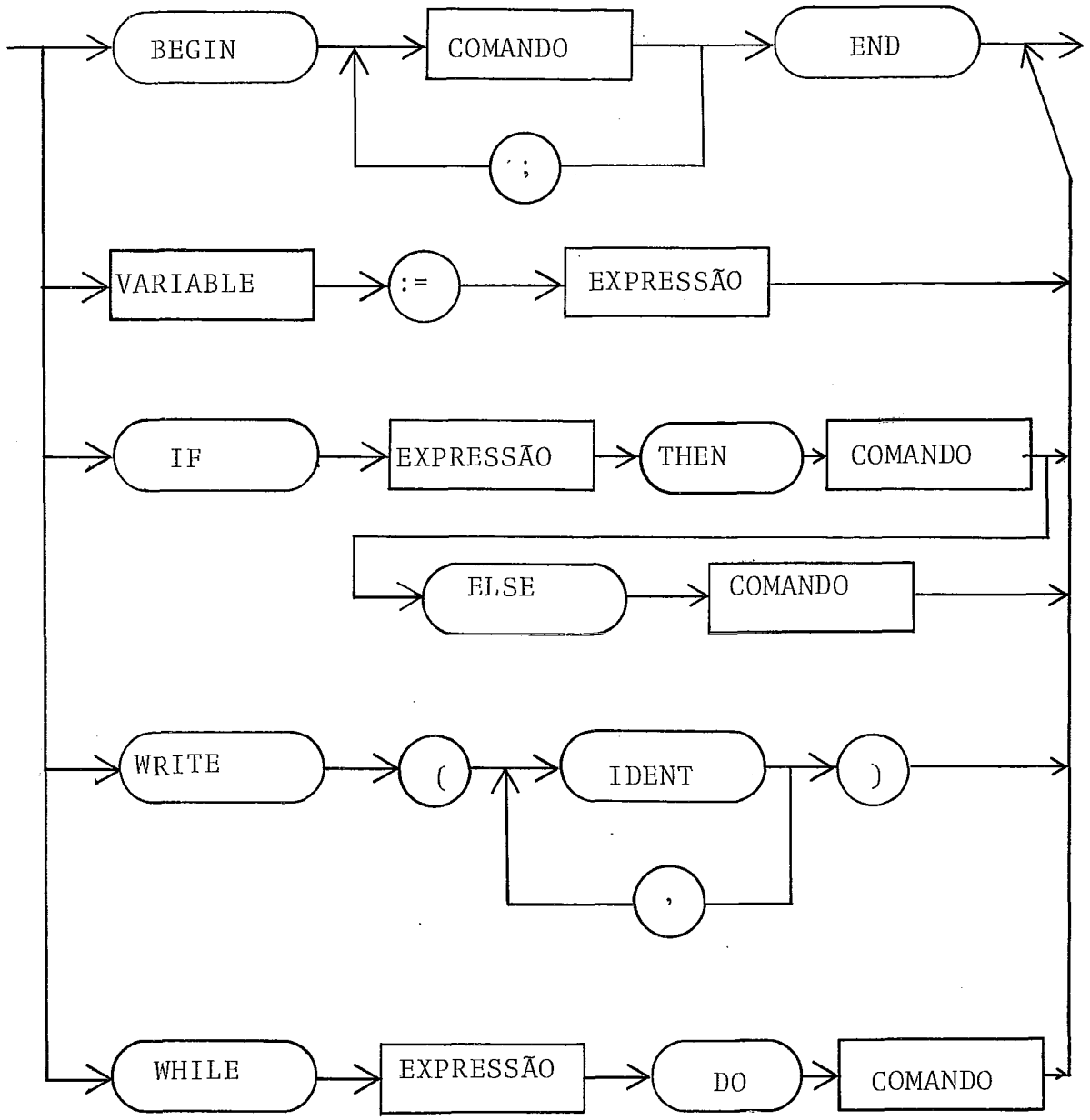
PROGRAM



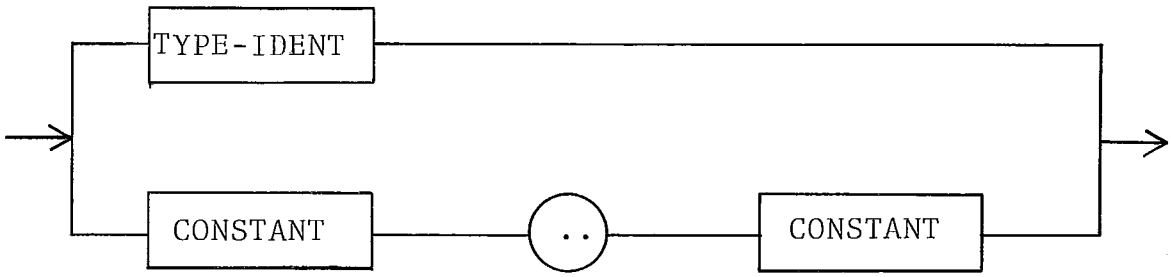
BLOCK



COMANDO

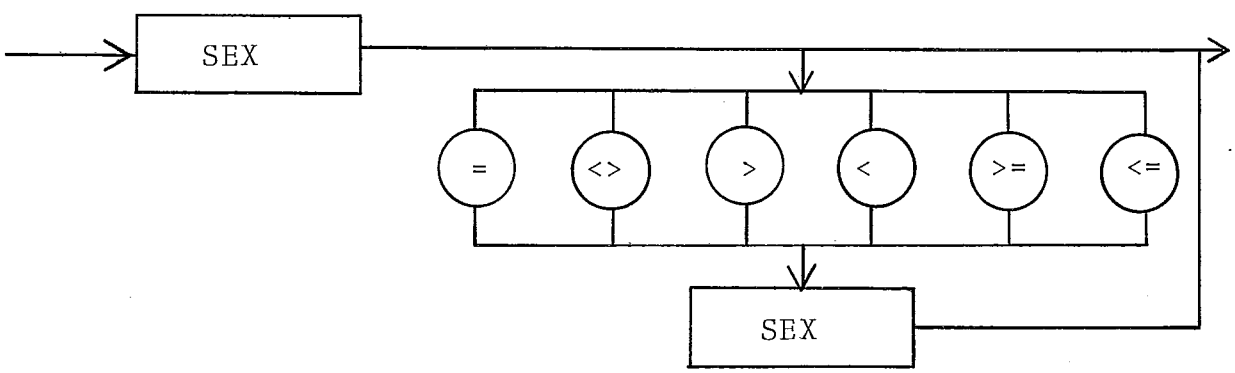

BEGIN
COMANDO
END
;
VARIABLE
:=
EXPRESSÃO
IF
EXPRESSÃO
THEN
COMANDO
ELSE
COMANDO
WRITE
(
IDENT
)
,
WHILE
EXPRESSÃO
DO
COMANDO

TYPE

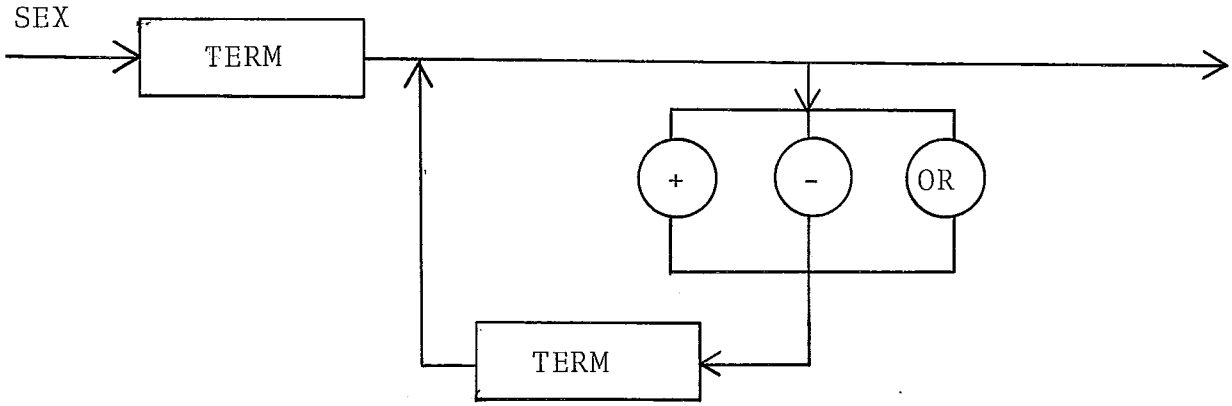


TYPE-IDENT

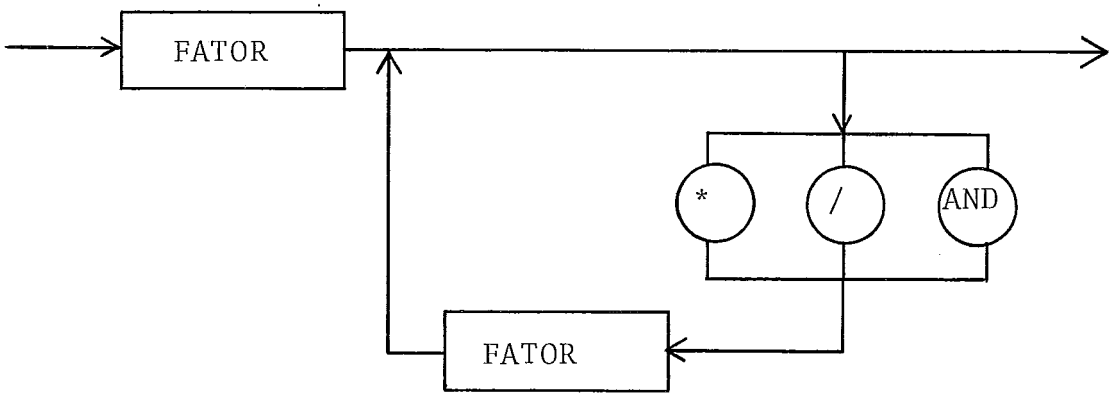
EXPRESSÃO



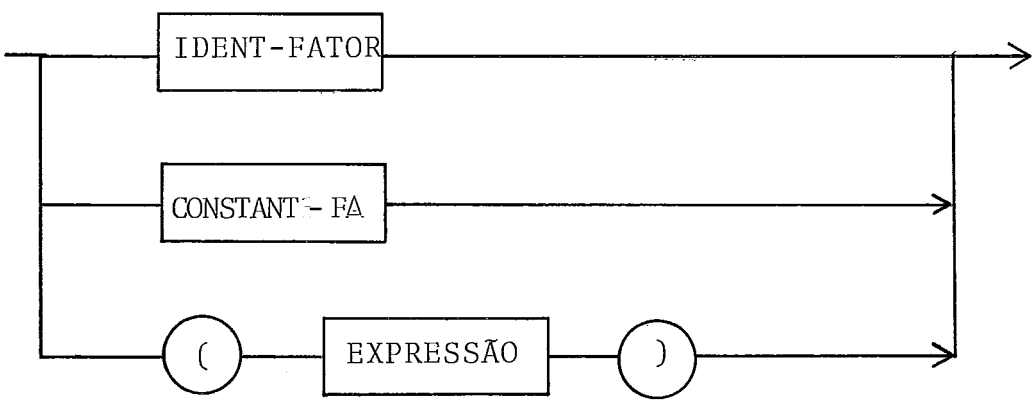
SEX



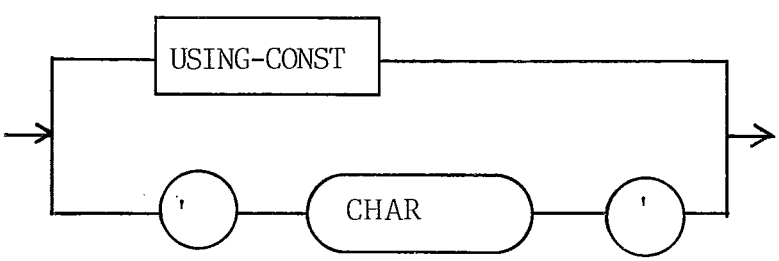
TERM



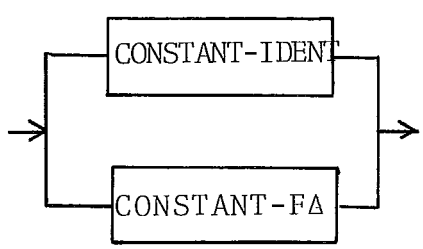
FATOR



CONSTANT-FA

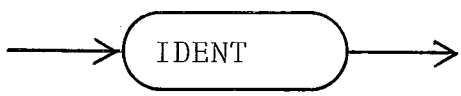


CONSTANT

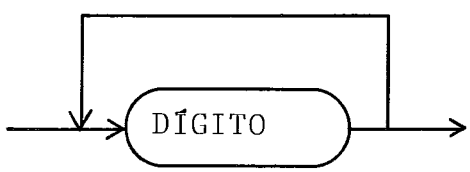


VARIABLE

IDENT-FATOR



USING-CONST



CONSTANT-IDENT