NOVAS TÉCNICAS PARA ANALISADORES SINTÁTICOS TIPO MATRIZ DE TRANSIÇÃO

– *Estevam Gilberto* ***De Simone*** –

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 DOUTOR EM CIÊNCIAS (D. Sc.).

Aprovada por:

José Lucas Mourão Rangel Neto
(Presidente)

Jayme Szwarcfiter

Sergio Roberto Teixeira

Sueli Mendes dos Santos

Valdemar Setzer

RIO DE JANEIRO

DEZEMBRO DE 1980

SIMONE, *ESTEVAM GILBERTO DE*

Novas Técnicas para Analisadores Sintáticos Tipo Matriz de Transição [Rio de Janeiro] 1980

X, 157 p. 29,7 cm (COPPE-UFRJ, D.Sc., Engenharia de Sistemas e Computação, 1980)

Tese – Univ. Fed. Rio de Janeiro, Fac. Engenharia.

1. Compiladores I. COPPE/UFRJ

11 Título (série)

— Ã *Estevam e Elvira*

AGRADECIMENTOS

- Aos meus alunos da COPPE e Instituto de Matemática da UFRJ;
- Aos meus colegas professores da área de Computação e do Departamento de Ciência da Computação;
- Ao Lucas, não apenas pelos cursos, pela orientação e pela paciência mas, principalmente, por ter emprestado a este trabalho um pouco do brilho de sua inteligência;
- À Ligia, à Beti, ao Zé Carlos e à Cecília pela ajuda importante e desinteressada;
- Aos professores Celio Guimarães, Jayme Szwarcfiter, Sergio Teixeira, Sueli Mendes dos Santos e Valdemar Setzer por acederem em participar desta banca;
- à COPPE, a FINEP e a CAPES pelo apoio recebido.
- À Stavna, pelo fantástico trabalho de datilografia, desenho, composição gráfica (e até correção de erros de português e conteudo)!

RESUMO

O método de análise sintática por matrizes de transição proposto por |*Gries, 68*| é totalmente reelaborado. É proposta uma definição equivalente para a classe de gramáticas reconhecida pelo método. É fornecido um algorítmo construtor mais eficiente para as tabelas de controle do analisador. É alterada a definição das funções que controlam o analisador e fornecido seu novo algorítmo. Estas novas funções são representáveis em tabelas que permitem determinar seus pontos inacessiveis e são fornecidos algorítmos de compactação das tabelas que permitem reduzir grandemente o espaço ocupado, preservado o formato matricial.

Paralelamente é desenvolvido um algorítmo de recuperação de erros que permite uma recuperação sofisticada e precisa, sem utilizar informação adicional às tabelas do analisador, sem degradar o desempenho do analisador para programas corretos e corrigindo de alguma forma as imprecisões na detecção do erro pelo analisador.

Demonstra-se que as tabelas originais de um analisador de gramáticas por matrizes de transição são menores que suas correspondentes para gramáticas SLR (1) e que o analisador por matrizes de transição é mais rápido que o correspondente analisador SLR(1).

ABSTRACT

Transition matrices have been used in parsing, since the method was proposed by |Gries, 68|. We present here a completely new vèrsion of the method, with a new equivalent definition for the class of grammars accepted.

New, more efficient algorithms are presented both for parsing and for the construction of the parsing tables. The control functions are redefined, and allow the determination of inaccesible points and table compression.

In parallel, an algorithm for error recovery is presented which results in a sophisticate and accurate recovery, which need no information besides the one in the parsing tables, and which introduces no degradation of the performance in the parser.

It is shown that the original tables of the transition matrix parser are smaller than the corresponding SLR(1) parsers, and also faster.

ÍNDICE

ÍNDICE DOS PARÁGRAFOS

CAPÍTULO I

- INTRODUÇÃO -

De todas as fases de um compilador a análise sintática é, sem dúvida, o problema melhor resolvido. Entretanto, talvez pela própria variedade de soluções distintas, é um campo aberto a polêmicas já clássicas: analisadores sintáticos ascendentes contra descendentes, analisadores controlados por tabelas ou por programas, geração automática de analisadores contra programação manual, gramáticas SLR contra gramáticas LL contra gramáticas LALR contra gramáticas LR contra gramáticas de precedência contra gramáticas recursivo-descendentes, e muitas mais. É realmente um tanto arriscado aventurar-se a pesquisar neste terreno.

Este trabalho é uma recomposição do método de análise sintática por matrizes de transição, proposto por |*Gries, 68*|. De fato, restou daquele método apenas o essencial: a classe de gramáticas, ditas *"gramáticas de matrizes de transição"* (GMT).

Não se encontra na literatura, referências posteriores a 1972 para analisadores GMT. Entre nós, todavia, o método ainda é bastante popular, sendo utilizado atualmente para a feitura de compiladores comerciais ou didáticos. A origem deste trabalho está no desenvolvimento do primeiro gerador de analisadores sintáticos feito na COPPE/UFRJ. Inicialmente voltado apenas para GMT esse gerador – o NHÃO NHÃO – deu origem a geradores de analisadores para as principais classes de gramáticas. O NHÃO NHÃO, em uso desde 1978, já permitiu a geração de tabelas de controle para uma dezena de linguagens distintas, usando matrizes de transi

ção.

O trabalho está dividido nas seguintes secções principais:

a) no Capítulo II é apresentado o estritamente necessário de teoria de linguagens, relações binárias, grafos e gramáticas de precedência de operadores, visando fixar os conceitos e a notação;

b) no Capítulo III faz-se uma revisão do método original com suas principais propriedades;

c) o Capítulo IV é o centro do trabalho onde o método é reformulado, são demonstrados os teoremas principais, é definido o novo conjunto de relações, é redefinida a classe de gramáticas e são propostos o novo analisador sintático e o novo construtor de analisadores sintáticos; além disso é apresentado o método de compactação das tabelas;

d) no Capítulo V, é feita uma revisão bibliográfica de recuperação de erros e apresentado o algorítmo recuperador;

e) o capítulo VI é dedicado a avaliação do trabalho e às conclusões.

Visando criar um sistema de referência sobre o próprio texto este foi paragrafado sendo os parágrafos numerados em romano e claramente delimitados. A notação utilizada está definida no Capítulo II ou na sua primeira ocorrência. O texto faz uso intensivo dessa notação menos pela afinidade do autor ao rigorismo que por sua capacidade de síntese.

CAPÍTULO II

– FUNDAMENTOS –

2.1 – FORMALISMO E DEFINIÇÕES BÁSICAS

São introduzidos aqui os conceitos necessários ao desenvolvimento da teoria nos capítulos posteriores. Paralelamente às definições será estabelecida sua notação. Serão adotadas a simbologia e a estruturação teórica de |*Aho & Ullman, 72*| .

■ Notação CONJUNTOS (I)

Um conjunto X formado pelos elementos $x_1, x_2, \ldots, x_n$ será denotado

(i) $X = \{x_1, x_2, \ldots, x_n\}$

O número de elementos desse conjunto será

(ii) $\#(X)$.

_____ ■

■ Definição GRAMÁTICA LIVRE DE CONTEXTO (GLC) (II)

Uma gramática G é dita livre de contexto ou, equivalentemente $G \in \mathcal{G}_{GLC}$, quando G é uma quádrupla (N, Σ, P, S) com:

(i) N, um conjunto finito de símbolos denominados não-terminais;

(ii) C, um conjunto finito de símbolos denominados terminais;

(iii) $N \cap C = \Phi$;

(iv) P, um conjunto finito de pares denominados produções onde o primeiro elemento é um não-terminal e o segundo é uma sequência não-vazia de terminais e/ou não-terminais. Ou seja,

$$P = \{(A,\alpha) \mid A \in N \wedge a \in (N \cup \Sigma)^+\}$$

Uma produção $(A, a) \in P$ será denotada $(A \rightarrow a)$;

(v) $S \in N$, um não-terminal especial denominado símbolo inicial.

Omitiremos, doravante, a menção "*livre de contexto*".

Notação CONJUNTOS GRAMATICAIS (III)

(i) NÃO TERMINAIS

$N = \{A, B, C, \ldots, T\}$ ou $N = \{$PROGRAMA, BLOCO, ...$\}$

(ii) TERMINAIS

$\Sigma = \{a, b, c, \ldots, t\}$ ou $C = \{$if, then, ...$\}$

(iii) TERMINAIS OU NÃO-TERMINAIS

$(N \cup C) = \{U, V, X, \ldots, Z\}$

(iv) SEQUENCIAS DE TERMINAIS

$(C)^* = \{u,v,x, \ldots z\}$

(v) SEQUENCIAS DE TERMINAIS E/OU NÃO-TERMINAIS

$(N \cup \Sigma)^* = \{\alpha, \beta, \gamma \ldots, \tau\}$

(vi) SEQUÊNCIA VAZIA

ε

Definição DERIVAÇÃO E REDUÇÃO (IV)

Sejam $G = (N, C, P, S)$ e $\beta A \gamma \in (N \cup \Sigma)^*$.

Diremos que $\beta A \gamma$ *"deriva diretamente"* $\beta\alpha\gamma$ em G, se $(A \to a) \in P$. Indica-se por:

(i) $\beta A \gamma \underset{G}{\Rightarrow} \beta\alpha\gamma$

Diremos ainda nesse caso que $\beta\alpha\gamma$ *"reduz-se diretamente"* a $\beta A \gamma$ em G e indicamos por:

(ii) $\beta A \gamma \underset{G}{\Leftarrow} \beta\alpha\gamma$

Por extensão, diremos que a *"deriva em zero ou mais PUA-sos"* - ou simplesmente *"deriva"* - 6 se existirem sequencias $\beta, \gamma, \ldots$ tais que

$$\alpha \underset{G}{\Rightarrow} \beta \underset{G}{\Rightarrow} \gamma \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} \delta$$

e indicamos

(iii) $\alpha \overset{*}{\underset{G}{\Rightarrow}} \delta$

Analogamente, 6 *"reduz-se"* a a em G e

(iv) $a \overset{*}{\underset{G}{\Leftarrow}} 6$

Quando nenhuma produção for aplicada ainda teremos

e

Quando pelo menos uma produção for aplicada diremos que

(vii) $\alpha \underset{G}{\overset{+}{\Longrightarrow}} \delta$ e (viii) $a \underset{G}{\overset{+}{\Longleftarrow}} 6$

Omitiremos o nome da gramática quando não causar dúvida.

Definição FORMA SENTENCIAL E SENTENÇA (V)

(i) se $G = (N, C, P, S)$ e $S \Rightarrow a \Rightarrow \beta \Rightarrow \ldots \Rightarrow \gamma$ então $\alpha, \beta, \ldots, \gamma$ serão formas sentenciais de G.

(ii) se $S \underset{G}{\overset{+}{\Longrightarrow}} u$ então u será sentença produzida por G.

Definição LINGUAGEM (VI)

Se $G = (N, C, P, S)$ então a linguagem produzida por G será

$$L(G) = \{u \in \Sigma^* \mid S \underset{G}{\overset{*}{\Longrightarrow}} u\}$$

Definição GRAMÁTICA REDUZIDA (VII)

Uma gramática $G = (N, \Sigma, P, S)$ será dita reduzida

se, simultaneamente:

(i) $L(G) \neq \Phi$ (caso particular de (iii) abaixo)

(ii) $(A \to \alpha) \in P \Rightarrow A \neq \alpha$

(iii) $A \in N \Rightarrow L(G') \neq \Phi,\ G' = (N, \Sigma, P, A)$

(iv) $A \in N \Rightarrow S \overset{*}{\underset{G}{\Longrightarrow}} \alpha A \beta$

Definição GRAMÁTICA UNICAMENTE INVERSÍVEL (VIII)

Uma gramática reduzida $G = (N, \Sigma, P, S)$ será dita unicamente inversível se não possuir duas produções com mesmo lado direito.

Definição DERIVAÇÃO (REDUÇÃO) DIREITA (ESQUERDA) (IX)

Se $G = (N, \Sigma, P, S)$ e $S \Longrightarrow \alpha Au \Longrightarrow \beta Bv \Longrightarrow \ldots \Longrightarrow \gamma C x$ então S deriva γCx por derivação mais a direita ou, simplesmente, 'derivação direita e

(i) $S \overset{*}{\underset{DIR}{\Longrightarrow}} \gamma Cx$

Ainda neste caso,

(ii) $S \overset{*}{\underset{DIR}{\Longleftarrow}} \gamma Cx$ (redução direita)

Se $S \Longrightarrow uA\alpha \Longrightarrow vB\beta \Longrightarrow \ldots \Longrightarrow xC\gamma$ então S deriva $xC\gamma$ por derivação esquerda e

(iii) $S \overset{*}{\underset{ESQ}{\Longrightarrow}} xC\gamma$

e, ainda,

(iv) $S \overset{*}{\underset{ESQ}{<=}} xC\gamma$ (redução esquerda)

Omitiremos a indicação direita/esquerda quando for suficientemente claro.

Definição RELAÇÃO BINÁRIA (X)

Sejam A, B conjuntos. Uma relação R é qualquer subconjunto de (AxB).

(i) $R \subseteq \{(a,b) \mid a \in A \wedge b \in B\} = (AxB)$

(ii) $(a,b) \in R \Leftrightarrow aRb$

Definição RELAÇÕES BINÁRIAS SOBRE CONJUNTOS GRAMATICAIS (XI)

(i) FIRSTNT $\subseteq$ (NxN)

A FIRSTNT B $\Leftrightarrow$ (A $\rightarrow$ Ba) $\in$ P

(ii) LASTNT $\subseteq$ (NxN)

A LASTNT B $\Leftrightarrow$ (A $\rightarrow$ αB) $\in$ P

(iii) NT.TERM $\subseteq$ (N x Σ)

A NT.TERM a $\Leftrightarrow$ (B $\rightarrow$ αAaβ) $\in$ P

(iv) TERM.NT (ΣxN)

a TERM.NT A $\Leftrightarrow$ (B $\rightarrow$ αaAβ) $\in$ P

(v) SYMB $\subseteq$ (NxN)

A SYMB B $\Leftrightarrow$ (A $\rightarrow$ B) $\in$ P

(vi) FIRSTTERM $\subseteq$ (N x Σ)

A FIRSTTERM a $\Leftrightarrow$ (A $\rightarrow$ Baα) $\in$ P, B $\in$ (N $\cup$ {ε})

(vii) LASTTERM $\subseteq (N \times \Sigma)$

A LASTTERM a $\Leftrightarrow (A \rightarrow \alpha a B) \in P$, $B \in (N \cup \{\varepsilon\})$

Definição OPERAÇÕES SOBRE RELAÇÕES (XII)

Sejam $R \subseteq (A \times B)$, $S \subseteq (A \times B)$, $T \subseteq (B \times C)$, $V \subseteq (A \times A)$

(i) SOMA

$(R + S) = \{(a,b) \mid aRb \vee aSb\} \subseteq (A \times B)$

(ii) PRODUTO

$(RT) = \{(a,c) \mid aRb \wedge bTc\} \subseteq (A \times C)$

(iii) POTÊNCIA

$(V^0) = \{(a,a) \mid a \in A\} \subseteq (A \times A)$

$(V^n) = VV^{n-1}$, $\forall n > 0$

(iv) TRANSPOSIÇÃO

$(V^{-1}) = \{(a,b) \mid bVa\} \subseteq (A \times A)$

(v) FECHAMENTO TRANSITIVO

$(V^+) = V + V^2 + V^3 + \ldots \subseteq (A \times A)$

(vi) FECHAMENTO TRANSITIVO REFLEXIVO

$(V^*) = V^0 + V^+ \subseteq (A \times A)$

Definição REPRESENTAÇÃO DE RELAÇÕES POR GRAFOS (XIII)

(Baseado em |*Hunt*, 77|)

Seja $R \subseteq (A \times B)$ e $G = (V,X)$ um grafo direcionado onde V é o conjunto finito de vértices e X é o conjunto finito de arestas (u, v), com u, v $\in$ V.

Se (i) $V = V_1 \cup V_2$ e $V_1 \cap V_2 = \phi$

(ii) $(u, v) \in X$, $u \in V_1$, $v \in V_2$ $\Leftrightarrow$ uRv (*)

(iii) não houver outras arestas

então G representa R.

(*) *Note que chamamos pelo mesmo nome tanto um elemento do domínio (ou contradomínio) de uma relação quanto o nó a ele correspondente no grafo. A distinção pode ser feita pelo contexto.*

Definição OPERAÇÕES SOBRE RELAÇÕES REPRESENTADAS POR GRAFOS (XIV)

Sejam: $G_1 = (I_1 \cup O_1, X_1)$ representando $R \subseteq (AxB)$

$G_2 = (I_2 \cup O_2, X_2)$ representando $S \subseteq (AxB)$

$G_3 = (I_3 \cup O_3, X_3)$ representando $T \subseteq (BxC)$

$G_4 = (I_4 \cup O_4, X_4)$ representando $W \subseteq (AxA)$

Suporemos ainda que todos os conjuntos de nós I_1, I_2, I_3, I_4, O_1, O_2, O_3 e O_4 são disjuntos entre si.

(i) SOMA

$G = (I \cup O, X)$ representará $R + S$ se:

(a) $I = I_1 = I_2$

(b) $O = O_1 = O_2$

(c) $(u, v) \in X_1 \Rightarrow (u,v) \in X$

(d) $(u, v) \in X_2 \Rightarrow (u,v) \in X$

(e) não há outras arestas

(ii) PRODUTO

$G = (I \cup C \cup O, X)$ representará RT se:

(a) $I = I_1$

(b) $O = O_3$

(c) $C = O_1 = I_3$

(d) $(u,v) \in X_1 \Rightarrow (u, v) \in X, \quad u \in I, \quad v \in C$

(e) $(u, v) \in X_3 \Rightarrow (u, v) \in X, \quad u \in C, \quad v \in O$

(f) não há outras arestas

(iii) TRANSPOSIÇÃO

$G = (I \cup O, X)$ representará W^{-1} se:

(a) $I = O_4$

(b) $O = I_4$

(c) $(u, v) \in X_4 \Rightarrow (v, u) \in X, \quad v \in I, \quad u \in O$

(d) não há outras arestas

(iv) FECHAMENTO TRANSITIVO REFLEXIVO

$G = (I \cup O, X)$ representará W^* se:

(a) $I = O$

(b) $(u, v) \in X_4 \Rightarrow (u, v) \in X$

(v) FECHAMENTO TRANSITIVO

Obtido por $W^+ = WW^*$

_____ ■

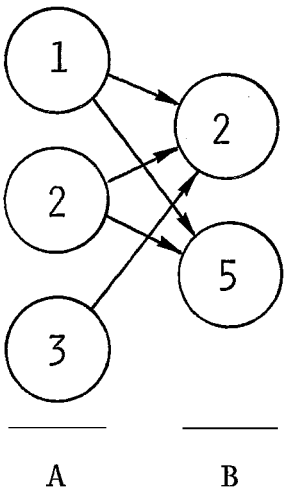
Exemplo

$A = \{1,2,3\}$ $R = \{(1,2), (3,2), (2,5)\}$ $W = \{(1,2),(2,3)\}$

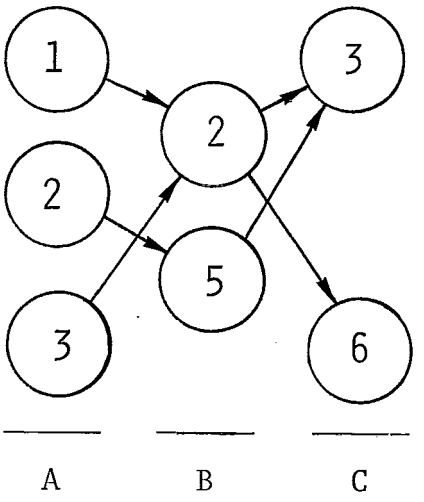
$B = \{2,5\}$ $S = \{(2,2), (1,5)\}$

$C = \{3,6\}$ $T = \{(2,3), (5,3), (2,6)\}$

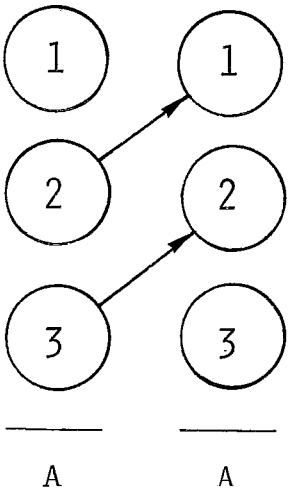
$R + S$



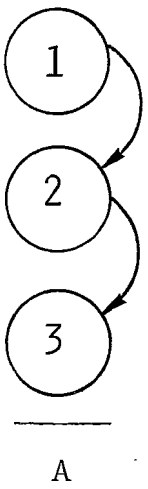
RT



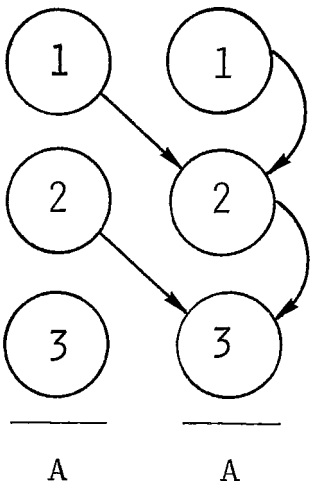
W^{-1}



W^*



W^+



■

Notação ALGORÍTMO (XVI)

Os algorítmos constantes deste trabalho serão escritos em português, envolvendo alguma notação matemática e lógica, além da notação definida no próprio texto. Seu funcionamento será demonstrado teoricamente quando necessário.

Definição ANALISADOR SINTÁTICO DA GRAMÁTICA G (XVII)

É um algorítmo que admite a sentença x como entrada e fornece como saída: . ,

(i) "ACEITO", se $x \in L(G)$, fornecendo ainda uma sequência ordenada das produções usadas em $S \overset{*}{\underset{G}{\Longrightarrow}} x$;

(ii) "REJEITADO", se $x \notin L(G)$.

2.2 – GRAMÁTICAS DE PRECEDÊNCIA DE OPERADORES

O presente trabalho guarda íntima relação com o estudo das gramáticas de precedência de operadores, que se justifica sua explicação detalhada aqui.

■ Definição: GRAMÁTICA DE OPERADORES (GO) (XVIII)

Uma gramática reduzida $G = (N, C, P, S)$ é dita gramática de operadores, ou equivalentemente da classe $\mathcal{C}_{GO}$, se

$$(A \rightarrow \alpha) \in P \quad \blacksquare\!\!\rangle \quad \alpha \neq \beta BC\gamma$$

Ou seja, não há dois não-terminais adjacentes no lado direito de suas produções.

———— ■

■ Teorema 1 (XIX)

$$G = (N, \Sigma, P, S) \in \mathcal{C}_{GO} \wedge S \overset{*}{\underset{G}{\Longrightarrow}} \alpha \quad \blacksquare\!\!\rangle \quad \alpha \neq \beta AB\gamma$$

Ou seja, não há forma sentencial em G com dois não-terminais adjacentes. ■

Demonstração: |*Floyd*, 63|

———— ■

A principal característica das GO é a possibilidade de se montar analisadores sintáticos baseados apenas em relações sobre os terminais da gramática. A subclasse importante das GO é a classe das gramáticas de precedência de operadores caracteriza

da pelas seguintes relações.

Definição RELAÇÕES DE PRECEDÊNCIA ENTRE OPERADORES |*Floyd*, 63| (XX)

(Formalização segundo |*Gries*, 71|)

Seja $G = (N, C, P, S) \in \mathcal{C}_{GO}$.

(i) relação $\gtrdot \subseteq (\Sigma \times \Sigma)$ (maior operadores)

$$a \underset{G}{\gtrdot} b \iff (A \to \alpha B b \beta) \in P \wedge (C \to \gamma a D) \in P \quad , D \in (N \cup \{\varepsilon\}) \wedge B \overset{*}{\underset{G}{\Longrightarrow}} \delta C$$

(ii) relação $\lessdot \subseteq (C \times C)$ (menor operadores)

$$a \underset{G}{\lessdot} b \iff (A \to \alpha a B \beta) \in P \wedge (C \to D b y) \in P \quad , D \in (N \cup \{\varepsilon\}) \wedge B \overset{*}{\underset{G}{\Longrightarrow}} C \delta$$

(iii) relação $\doteq \subseteq (\Sigma \times \Sigma)$ (igual operadores)

$$a \underset{G}{\doteq} b \iff (A \to \alpha a D b \beta) \in P \quad , D \in (N \cup \{\varepsilon\})$$

Pode-se deduzir com facilidade |*Gries*, 71| que, conforme as definições de (X):

(iv) $a \underset{G}{\ } b \iff a\ (\text{LASTTERM})^{-1}\ (\text{LASTNT}^*)^{-1}\ \text{NT.TERM}\ b$

(v) $a \underset{G}{\lessdot} b \iff a\ \text{TERM.NT}\ \text{FIRSTNT}^*\ \text{FIRSTTERM}\ b$

(vi) a $\stackrel{\circ}{=}_G$ b pode ser construída por inspeção na gramática.

A referência à gramática pode ser omitida na notação.

Definição GRAMÁTICAS DE PRECEDÊNCIA DE OPERADORES (GPO) (XXI)

Uma gramática unicamente inversível $G=(N, \Sigma, P, S)$ e $\mathcal{C}_{GO}$ será dita de precedência de operadores, ou equivalentemente, pertencente à classe $\mathcal{C}_{GPO}$, se:

(i) $\lessdot \quad n \quad \stackrel{\circ}{=} \quad = \phi$

(ii) $\lessdot \quad n \quad \gtrdot \quad = \phi$

(iii) $\stackrel{\circ}{=} \quad \cap \quad \gtrdot \quad = \phi$

Vamos nos aproveitar desta classe de gramáticas para introduzir os conceitos e exemplificar o problema da análise sintática, conforme o trataremos posteriormente.

Interessa-nos obter do analisador sintático a árvore sintática tal como definida abaixo.

Definição ÁRVORE SINTÁTICA (XXII)

Seja $G = (N, C, P, S)$ e x uma sentença em $L(G)$

A árvore sintática de x em G será uma árvore cons-

truída da forma abaixo, supondo-se conhecida uma derivação $S \overset{*}{\underset{G}{\Longrightarrow}} x$:

(i) a raiz terá rótulo S;

(ii) a cada passo da derivação, para o nó cujo rótulo é o lado esquerdo da produção usada crie tantos filhos rotulados quantos forem os símbolos do lado direito dessa produção.

As folhas da árvore sintática serão terminais e os nós internos serão não-terminais.

———— ■

■ Exemplo (XXIII)

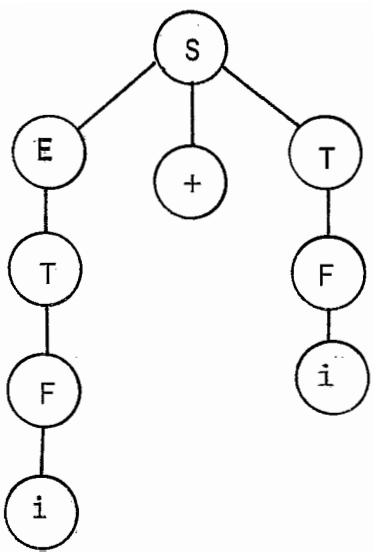
Seja $G_0 = (\{E, T, F\}, \{+, *, (,), i\}, P, E)$ com

$P = \{(E \to E + T), (E \to T), (T \to T * F), (T \to F), (F \to (E)), (F \to i)\}$.

Seja a derivação

$$S \underset{G_0}{\Longrightarrow} E+T \Longrightarrow T+T \Longrightarrow F+T \Longrightarrow i+T \Longrightarrow i+F \Longrightarrow i+i$$

A árvore sintática correspondente seria:



Se tomarmos outra derivação

$$S \underset{G_0}{\Longrightarrow} E + T \Longrightarrow E + F \Longrightarrow E + i \Longrightarrow T + i \Longrightarrow F + i \Longrightarrow i+i$$

ainda teremos, neste caso, a mesma árvore sintática.

_____ ■

■ Definição GRAMÁTICA AMBÍGUA (XXIV)

Uma gramática $G = (N, \Sigma, P, S)$ é dita ambígua, se:

(i) para $x \in L(G)$ existir mais de uma árvore sintática

Esta condição é equivalente a:

(ii) para $x \in L(G)$ existir mais de uma derivação direita (esquerda) de x em G.

_____ ■

Definição PARSE (XXV)

Seja $G = (N, \Sigma, P, S)$ com as produções de P numeradas 1,2, ..., p e uma derivação $S \overset{*}{\underset{G}{\Longrightarrow}} x$.

A sequência dos números das produções utilizadas na derivação é o parse de x em G.

Mais particularmente:

(i) o parse descendente de x em G é a sequência dos números das produções utilizados na derivação $S \overset{*}{\underset{ESQ}{\Longrightarrow}} x$;

(ii) o parse ascendente de x em G é a sequência dos números das produções utilizadas na redução $S \overset{*}{\underset{DIR}{\Longleftarrow}} x$;

Há correspondência biunívoca entre parse ascendente, parse descendente e árvore sintática.

Em todo o decorrer deste trabalho usaremos o parse como a sequência ordenada de produções prevista em XVII. (ii), a saída dos analisadores sintáticos. Como trataremos apenas analisadores ascendentes toda referência futura a parse significará parse ascendente.

Os métodos de análise que estudaremos não permitem a obtenção - ou, caso permitam, será ignorada - do parse da sentença. Veremos neste capítulo a definição de parse *"estrutural"* e no capítulo seguinte a definição de parse *"esparso"*.

Para as gramáticas de precedência de operadores não se

consegue obter analisadores sintáticos , tal como definidos em XVII, mas apenas analisadores sintáticos de sua gramática estrutural ("skeletal" em |*Aho & Ullman, 72*|). Esta gramática é obtida da gramática original pela substituição de todas as ocorrências de não-terminais pelo símbolo inicial. Isto é, em geral, suficiente para a construção de compiladores, sendo uma preocupação reduzir o tempo gasto com produções simples - do tipo $A \rightarrow B$ com $A, B \in N$ - que normalmente não carregam informação semântica.

Esta discussão sobre a necessidade de se eliminar produções simples tem gerado um sem número de "otimizadores" de analisadores sintáticos |*Anderson, 73*|, |*Aho & Ullman, 73*|, |*Pager, 74*|, |*Demers, 75*|, |*Backhouse, 76*|, |*LaLonde, 76*|, |*Soisalon - Soininen, 77*|, |*Rushby, 77*|, cada qual propondo métodos distintos de aperfeiçoar a análise sintática das classes de gramáticas, em particular a família LR, que produzem parses completos. Tanta efervecência no assunto deve-se ao alto lucro obtido pela análise sintática sem produções simples: |*Joliat, 73*| mostrou que sua eliminação eficiente dobra a velocidade da análise sintática e aumenta a velocidade do compilador, como um todo, em 15%.

Outra alternativa para o mesmo problema são os métodos recentes de análise sintática em gramáticas com lados direitos regulares |*De Rema, 70*|, |*LaLonde, 75*|, |*Simone, 80*| . que partem para a eliminação da causa primária das produções simples: a inadequação do mecanismo descritivo composto pelas gramáticas livres de contexto, para descrever as próprias linguagens livres de contexto.

De modo que estaremos tratando de analisadores sintáticos que, por natureza, gozam da propriedade de ignorar produções simples.

Vejamos, então, como opera o analisador sintático de gramáticas de precedência de operadores, descrito pelo algorítmo 1, a seguir.

Este algorítmo obtem o parse estrutural da sentença, sendo controlado por duas funções:

(i) $f : (C \times C) \longrightarrow \{\lessdot, \doteq, *, \text{"erro"}\}$

definido por:

$f(a, b) = \gtrdot \Leftrightarrow a \gtrdot b$

$f(a, b) = \doteq \quad a \doteq b$

$f(a,b) = \lessdot \quad a \lessdot b$

$f(a,b) =$ "erro", qualquer outro caso.

(ii) $g : (C \cup \{S\})^{+} \longrightarrow \{1, 2, \ldots, p, \text{"stop"}\}$

definida por:

$g(\alpha) = i \qquad (i: S \to \alpha) \in P$

$g(a) =$ "stop", qualquer outro caso.

O algorítmo é uma versão do clássico algorítmo *"avança-reduz"* para análise sintática ascendente. A função f determina as ações de avançar um símbolo da entrada ou reduzir, indicando também a frase a ser reduzida. A função g indica o número da produção usada.

Algorítmo_1 (XXVI)

ANALISADOR SINTÁTICO PARA GRAMÁTICA ESTRUTURAL DE GPO

Passo 1: (Inicialização)

Coloque na pilha o marcador "#" ;
Faça a variável MEIO = " " ;
Leia um símbolo da sentença para ENTRADA;
Repita o passo 2 até receber ordem de parada.

Passo 2: (Escolha da ação)

Caso f(topo da pilha, ENTRADA) igual a:

$\lessdot$ ou $\doteq$: execute o passo 3;
$\gtrdot$: execute o passo 4;
"erro" : execute o passo 5;

Passo 3: (Avanço)

Se MEIO = "S" então coloque S na pilha e faça MEIO=" ";
Coloque ENTRADA na pilha;
Leia novo símbolo para ENTRADA;

Passo 4: (Redução)

Retire da pilha todos os símbolos até encontrar dois símbolos cuja relação entre si seja $\lessdot$;
Concatene tais símbolos retirados, em ordem;
Se MEIO = "S", concatene também "S", à direita;
Seja SEQ o produto da concatenação;
Se G(SEQ) $\neq$ "stop" escreva G(SEQ) e faça MEIO = "S"

senão: verifique se o topo da pilha é "#", se MEIO = "S" e ENTRADA = "#" ;
Caso as três condições ocorram escreva "ACEITO" e pare;
Caso contrário escreva "REJEITADO" e pare;

Passo 5: (Erro)

Escreva "REJEITADO" e pare;

O algorítmo se baseia na seguinte propriedade das gramáticas de precedência e é válida também para as GPO, na forma abaixo:

"Seja $X_1 X_2 \ldots X_m$ uma forma sentencial e $a_1 a_2 \ldots a_n$ os terminais dessa forma sentencial. A sequência a ser reduzida é a sequência mais à esquerda tal que

$$a_g \lessdot a_{g+1} \doteq a_{g+2} \doteq \ldots \doteq a_k \gtrdot a_{k+1}$$ "

A demonstração desta propriedade pode ser vista em |*Gries, 71*| .

O sinal " # " é usado como inicialização da pilha e final da sentença de entrada. Faz parte do conjunto de terminais da gramática aumentada, obtida a partir de G = (N, C, P, S) com G_A = (N ∪ {S'}, Σ ∪ {#}, P ∪ {(S' → # S #)}, S').

O algorítmo nada mais faz que varrer a sequência de entrada da esquerda para a direita, efetuando uma redução sempre que a situação acima ocorrer. As funções f e g são calcula-

das sobre a gramática estrutural de G_A.

Exemplo (XXVII)

(i) $G_0 = (\{E, T, F\}, \{+, *, (,), i\}, P, E)$

$P = \{(E \to E + T), (T \to T * F), (F \to (E)), (E \to T), (T \to F), (F \to i)\}$

(ii) $G_{0A} = (\{S, E, T, F\}, \{\#, +, *, (,), i\}, P', S)$

$P' = \{(S \to \# E \#), (T \to T * F), (F \to (E)), (E \to T), (E \to E + T), (T \to F), (F \to i)\}$

(iii) $G_{ESTR} = (\{S\}, \{\#, +, *, (,), i\}, P'', S)$

$P'' = \{ 1 : S \to \#S\#$

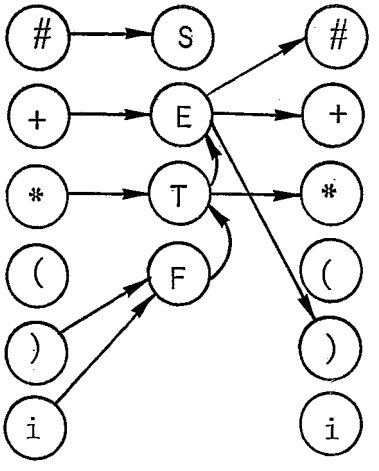
$2 : S \to S + S$

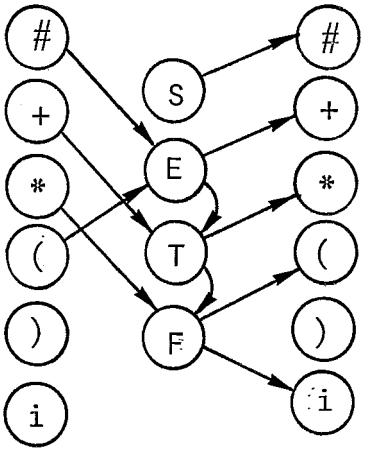
$3 : S \to S * S$

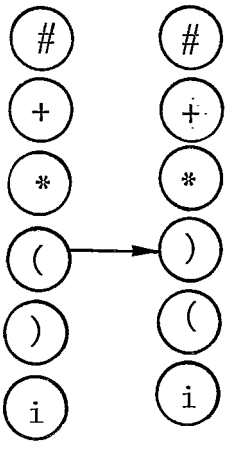
$4 : S \to (S)$

$5 : S \to i \}$

(iv) Cálculo das funções f e g







F	#	+	*	(	)	i
#	?	⋖	⋖	⋖	?	⋖
+	⋗	⋗	⋖	⋖	⋗	⋖
*	⋗	⋗	⋗	⋖	⋗	⋖
(	?	⋖	⋖	⋖	≐	⋖
)	⋗	⋗	⋗	?	⋗	?
i	⋗	⋗	⋗	?	⋗	?

? = "erro"

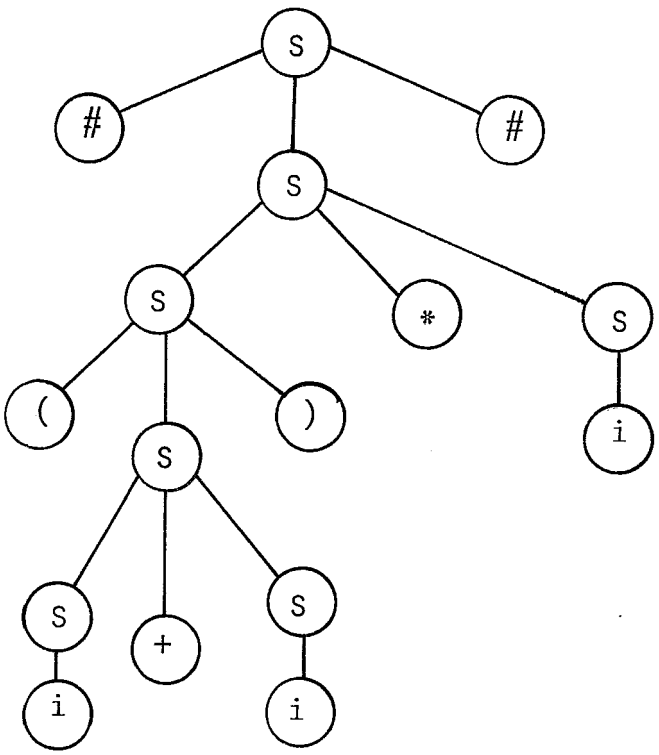
G	
1	#S#
2	S+S
3	S*S
4	(S)
5	i
"stop"	qq.outro

(v) Execução

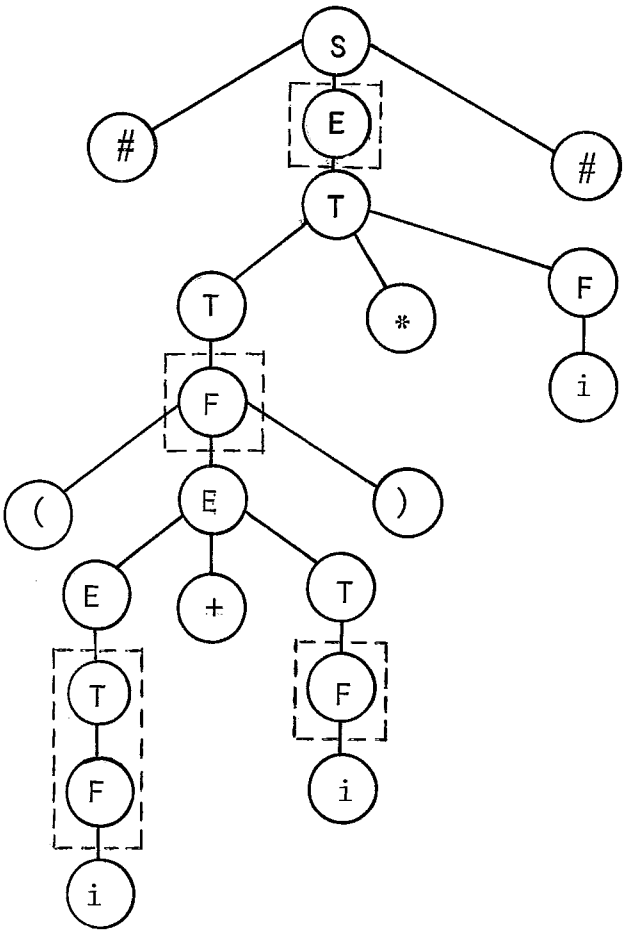
PILHA	MEIO	ENTRADA	PARSE ESTRUTURAL
#		(i+i)*i#	
# (		i+i)* i #	
# (i		+i) * i #	5
# (	S	+i) * i #	
# (S+		i) * i #	
# (S+i		) * i#	5
# (S+	S	) * i #	2
# (	S	) * i #	
# (S)		* i #	4
#	S	* i #	
# S*		i #	
# S*i		#	5
# S*	S	#	**3**
# S		#	ACEITO

(vi) Árvores Sintáticas

ESTRUTURAL



ORIGINAL



■

CAPÍTULO III

– MATRIZES DE TRANSIÇÃO –

Propriamente, começaremos aqui. Este capítulo será a revisão do conhecido sobre esta sub-classe das gramáticas livres de contexto. |*Gries, 68*| propôs a denominação *"transition matrix"* para a técnica de análise sintática que então formalizou. Essa *"matriz de transição"* guarda semelhanças com a *"tabela de transições"* de um automato finito. Tal semelhança é fatal à nomenclatura, Mais importante, porém, que o título da ferramenta é a definição dessa classe de gramáticas, mais ampla que a classe de precedência de operadores, baseada de certa forma em conceitos análogos. Como a classe de gramática de operadores, e sua nomenclatura, já estão também estabelecidas contentamo-nos em manter o título inadequado: matrizes de transição.

O fundamental deste capítulo foi extraído de |*Samelson & Bauer, 60*|, |*Conway, 63*| e, principalmente, |*Gries, 68*|. As demonstrações devidas são ali encontradas.

3.1 – GRAMÁTICA DE OPERADORES ESTENDIDA

O objetivo desta primeira modificação na gramática dada é reduzir o comprimento dos lados direitos das produções a, no máximo, três símbolos.

Note que a transformação descrita a seguir só é possí

vel se partirmos de uma gramática de operadores.

O processo se dará pela adição à gramática de novos não-terminais, denominados – outra vez com infelicidade – por |*Gries, 68*| de *"não-terminais estrelados"*, e de novas produções intermediárias, denominadas *"produções estreladas"*. A origem do nome vem do uso de um "*" para identificá-los. Preferimos, com maior clareza, sublinhá-los.

■ Definição (XXVIII)

GRAMÁTICA DE OPERADORES ESTENDIDA (GOE)

Seja $G = (N, C, P, S) \in \mathcal{G}_{GO}$. A gramática $G' = (N', C', P', S') = (N \cup \underline{N} \cup \{S'\}, \Sigma \cup \{\#\}, P', S')$ será sua gramática de operadores estendida se G' for construída conforme o algorítmo 2, a seguir.

———— ■

■ Algorítmo 2 (XXIX)

CONSTRUÇÃO DA GRAMÁTICA DE OPERADORES ESTENDIDA

Passo 1 : Construa $G' = (N', C', P', S')$ de modo que $G' = (N \cup \{S'\}, C \cup \{\#\}, P \cup \{S' \to \# S \#\}, S')$ com $S' \notin N$ e $\# \notin C$. Chamaremos esta gramática obtida no Passo 1 de gramática de operadores aumentada

Passo 2 : Se $(A \to aa) \in P'$, $\alpha \in (N' \cup \Sigma')^*$ então

(i) acrescente a produção $\underline{A} \to a$, se já não existir

alguma produção com lado direito a; caso exista, seja $\underline{V}$ o seu lado esquerdo;

(ii) substitua cada produção da forma

$(B \rightarrow a\beta) \in P'$ por $(B \rightarrow \underline{V}\beta)$;

Passo 3 : Se $(A \rightarrow Baa) \in P'$, $a \in (N' \cup C')$"

então

(i) acrescente a produção $\underline{V} \rightarrow Ba$, se já não existir alguma produção com lado direito Ba; caso exista, seja $\underline{V}$ o seu lado esquerdo;

(ii) substitua cada produção da forma

$(C \rightarrow Ba\beta) \in P'$ por $(C \rightarrow \underline{V}\beta)$;

Passo 4 : Se $(A \rightarrow \underline{U}a\alpha) \in P'$, $a \in (N' \cup \Sigma')$*

então

(i) acrescente a produção $\underline{V} \rightarrow \underline{U}a$, se já não existir alguma produção com lado direito $\underline{U}a$; caso exista seja $\underline{V}$ seu lado esquerdo;

(ii) substitua cada produção da forma

$(C \rightarrow Ua\beta) \in P'$ por $(C \rightarrow \underline{V}\beta)$;

Passo 5 : Se $(A \rightarrow UBa\alpha) \in P'$, $a \in (N' \cup \Sigma')$*

então

(i) acrescente a produção $\underline{V} \rightarrow \underline{U}Ba$, se já não existir alguma produção com lado direito UBa; caso exista seja $\underline{V}$ seu lado esquerdo;

(ii) substitua cada produção da forma

$(C \rightarrow \underline{U}Ba\beta) \in P'$ por $(C \rightarrow \underline{V}\beta)$;

■

Note que, a cada passo, $\underline{V}$ é um novo não-terminal criado quando uma nova produção for acrescentada e $\underline{N}$ é o conjunto desses não-terminais estrelados. Vamos convencionar que a numeração das produções originais seja (1, 2, ..., p), que o Passo 1 crie uma nova produção numerada 0 e que as demais produções acrescidas sejam numeradas (p + 1, p + 2, ...). Note que as substituições de produções não alteram sua numeração.

■ Exemplo (XXX)

$G = (N, \Sigma, P, S) \in \mathcal{G}_{GO}$

$N = \{S, A, B, C, D\}$ $\quad C = \{a, b, c, d\}$

$P = \{1 : S \to aAb\ ,\ 2 : A \to bcB\ ,\ 3 : B \to C,\ 4 : C \to Dd\ ,$
$5 : D \to d\}$

Após o Passo 1:

$N' = \{S', S, A, B, C, D\}$ $\quad C' = \{\#, a, b, c, d\}$

$P' = \{0 : S' \to \#S\#\ ,\ 1 : S \to aAb\ ,\ 2 : A \to bcB,\ 3 : B \to C\ ,$
$4 : C \to Dd\ ,\ 5 : D \to d\}$

Executando-se os passos seguintes:

0 : S' → #S#		0 : S' → $\underline{\#}$ S#		0 : S' → $\underline{\#}$ S#
1 : S → aAb		1 : S → $\underline{a}$ Ab		1 : S → $\underline{a}$ Ab
2 : A → bcB	P_2 ⇨	2 : A → $\underline{b}$ cB	P_3 ⇨	2 : A → $\underline{b}$ cB
3 : B → C		3 : B → C		3 : B → C
4 : C → Dd		4 : C → Dd		4 : C → $\underline{Dd}$
5 : D → d		5 : D → $\underline{d}$		5 : D → $\underline{d}$
		6 : $\underline{\#}$ → #		6 : $\underline{\#}$ → #

0 : S' → # S#
1 : S → a Ab
2 : A → bc B
3 : B → C
4 : C → Dd
5 : D → d
6 : # → #
7 : a → a
8 : b → b
9 : d → d
10 : Dd → Dd
11: bc → b c

7 : a → a
8 : b → b
9 : d → d

P_4

0 : S' → #S#
1 : S → aAb
2 : A → bc B
3 : B → C
4 : C → Dd
5 : D → d

P_5

6 : # → #
7 : a → a
8 : b → b
9 : d → d
10 : Dd → Dd
11 : bc → b c
12 : #S# → # S#
13 : aAb → a Ab

7 : a → a
8 : b → b
9 : d → d
10 : Dd → Dd

Propriedades das GOE (XXXI)

(i) após o passo 2 as produções são das formas

(A → B) , (A → Bca) , (A → Ua) , (U → a)

(ii) após o passo 3 as produções são das formas

(A → B) , (A → Uα) , (U → a) , (U → Ba)

(iii) após o passo 4 as produções são das formas

(A → B) , (A → UBaα), (U → a) , (U → Ba)

(U → U_1a) , (A → U)

(iv) em qualquer GOE temos apenas 7 formas de produção:

$(A \to B)$, $(A \to \underline{U})$, $(A \to \underline{U}B)$

$(\underline{U} \to a)$, $(\underline{U} \to Ba)$, $(\underline{U} \to \underline{U_1}a)$, $(\underline{U} \to \underline{U_1}Ba)$

(v) cada não-terminal estrelado aparece como lado esquerdo de uma Única produção; o lado direito correspondente aparece como lado direito apenas dessa produção.

(vi) se os novos não-terminais estrelados forem numerados conforme a ordem de aparecimento no algorítmo, então

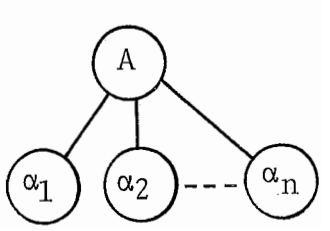
$(\underline{U_i} \to \underline{U_j}\alpha) \in P' \Rightarrow i > j$

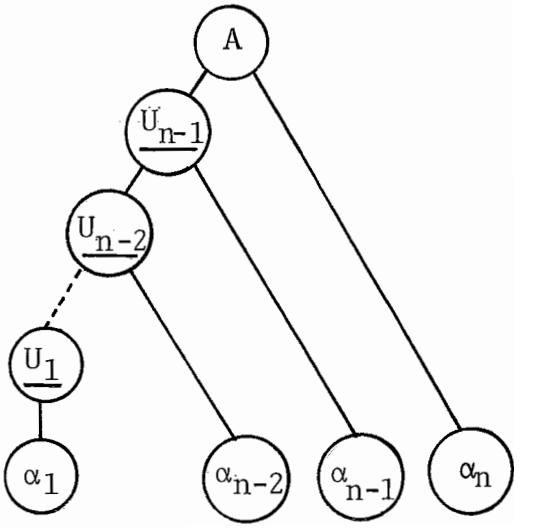
(vii) para cada produção $(A \to \alpha)$ na gramática original existe um Único conjunto de produções

$(\underline{U_1} \to \alpha_1)$, $(\underline{U_2} \to \underline{U_1}\alpha_2)$, ... , $(\underline{U_n} \to \underline{U_{n-1}}\ \alpha_n)$ tal que

$\alpha = \alpha_1\alpha_2 \cdots \alpha_n$

(viii) para cada árvore sintática de uma forma sentencial y na gramática de operadores pode-se construir uma Única árvore sintática da sentença $\#y\#$ na gramática de operadores estendida, substituindo-se o ramo $(A \to \alpha)$ pelas produções descritas no ítem anterior





Teorema 2 (XXXII)

Se uma GOE é não-ambígua então a GO da qual deriva é também não-ambígua.

Demonstração: decorrência das propriedades XXXI (iv), (v), (vi), (vii) e (viii) e devida a |*Gries, 68*|.

3.2 – ANÁLISE SINTÁTICA EM GOE

Iremos descrever a seguir o método de construção de analisadores sintáticos para GOE, baseado em relações de precedência, sendo portanto um reconhecedor ascendente que lê a sentença da esquerda para a direita. Pelo fato de propormos um analisador determinístico estaremos também definindo condições suficientes para que uma GOE seja não-ambígua.

Uma característica essencial deste método é que se baseará em relações de precedência entre não-terminais estrelados e terminais, não sendo capaz de detectar uma redução quando a produção usada for simples. Portanto, fornecerá um parse esparso.

As definições seguintes de "*frase*" e "*frase prima*" pretendem substituir a definição usual de "*handle*" em analisadores ascendentes e contornam o problema teórico da formalização do parse de modo mais sofisticado que a gramática estrutural vista no capítulo 2 para as gramáticas de precedência de operadores.

Definição FRASE (XXXIII)

Diremos que $\gamma \in (N' \cup \Sigma')^*$ é uma frase se

$S \overset{*}{\Longrightarrow} \beta A \delta \overset{*}{\Longrightarrow} \beta\gamma\delta$ e $A \overset{*}{\Longrightarrow} y$

Definição FRASE PRIMA (XXXIV)

Diremos que $y \in (N' \cup \Sigma')^*$ é uma frase prima se

(i) y é frase;

(ii) $y = \alpha X \beta$, $X \in (N \cup C)$;

(iii) $\gamma = \alpha_1 \alpha_2 \ldots \alpha_n \Rightarrow \alpha_i$ não é frase, qq. i

---

A definição é feita de tal forma que as frases primas serão o objeto das reduções, restando solucionar as reduções advindas das produções simples.

Teorema 3 (XXXV)

As frases primas em GOE são de 6 formas:

U , Ua, a, UA, Aa, UAa.

Demonstração : consequência da propriedade XXXI.(iv) e da definição de frase prima.

Vamos agora definir uma classe de gramáticas – subcon-

junto das GOE – para as quais será sempre possível obter um analisador sintático ascendente determinístico, do tipo que nos interessa.

■ ~~Definição~~ RELAÇÃO PRECEDE (XXXVI)

$$\text{PRECEDE} \subseteq (N' \cup C') \times (N' \cup C')$$

$$X \text{ PRECEDE } Y \Leftrightarrow S' \overset{*}{\Longrightarrow} \alpha XY\beta$$

_____ ■

■ ~~Definição~~ (XXXVII)

GRAMÁTICAS DE MATRIZES DE TRANSIÇÃO (GMT)

Uma gramática $G = (N', \Sigma', P', s') \in \mathcal{C}_{GOE}$ será dita gramática de matriz de transição (GMT) ou, equivalentemente, $G \in \mathcal{C}_{GMT}$, se:

(i) para qualquer par $(A,B) \in (N \cup \{S'\})^2$ se $A \overset{*}{\Longrightarrow} B$, então essa derivação é Única;

(ii) para cada par (U, a) e $(N \times C')$ no máximo uma das três seguintes condições é verdadeira:

(a)' $(A \to U) \in P' \wedge (A \text{ PRECEDE } a) \wedge (A \text{ Único})$;

(b) $(\underline{V} \to Ua) \in P'$;

(c) $(V \to a) \in P' \wedge (U \text{ PRECEDE } \underline{V})$;

(iii) para cada tripla $(U, A, a) \in (\underline{N} \times N \times C')$ no máximo uma das três seguintes condições é verdadeira:

(a) $(B \to Uc) \in P' \wedge (B \text{ PRECEDE } a) \wedge (C \overset{*}{\Longrightarrow} A) \wedge (B, C \text{ Únicos})$;

(b) $(V \rightarrow \underline{U}\ Ca) \in P' \quad \wedge (C \overset{*}{\Longrightarrow} A) \quad \wedge$ (C Único) ;

(c) $(\underline{V} \rightarrow Ca)$ E P' $\quad \wedge$ (U PRECEDE V)

$\wedge (C \overset{*}{\Longrightarrow} A) \quad \wedge$ (C único) ;

■

São consequências da definição:

(i) cada condição imposta corresponde a uma das seis possíveis formas de frase prima em GOE;

(ii) a GOE (e, em consequência de XXXI. (v) também a gramática de operadores original) não necessita ser unicamente inversível admitindo a existência de duas produções $(A \rightarrow B)$ e $(C \rightarrow B)$, desde que respeitada a condição de derivação Única. Para as produções não simples é necessário que a gramática seja UI, em consequência das condições (i).a e (ii.a) da definição de GMT.

■ Teorema 4 (XXXVIII)

$G \in \mathcal{G}_{GMT} \Rightarrow$ G é não-ambígua.

Demonstração: |Gries, 68|.

■

■ Algorítmo 3 (XXXIX)

CONSTRUTOR DE ANALISADOR SINTÁTICO PARA GMT

Passo 1 : Verifique se $A \overset{*}{\Rightarrow} B$ então a derivação é Única;

Passo 2 : Construa a GOE através do Alg.2;

Passo 3 : Construa a relação PRECEDE $\subset (N \cup \Sigma) \times (\Sigma \cup C')$;

Passo 4 : Para cada par (U, a) verifique se apenas uma das condições (i).a, (i).b e (i).c prevalece.
Assim sendo construa a quádrupla
(U, a, X, j)
onde X é o lado esquerdo da produção usada
j é o número dessa produção;

Passo 5 : Para cada tripla (U, A, a) verifique se apenas uma das condições (ii).a, (ii).b e (ii).c ocorre.
Assim sendo construa a 5-tupla
(U, A, a, X, j)

Passo 6 : Se mais de uma condição ocorrer nos passos 4 ou 5 ou se não for verificada a condição do passo 1 indique que $G \notin \mathcal{C}_{GMT}$.

■

É importante que se faça uma verificação sumária da complexidade do Alg. 3, visando sua comparação com o algorítmo a ser proposto no capitulo seguinte:

(i) o Passo 1, conforme veremos no próximo capítulo, é redutível ao problema de determinação da existência de dois caminhos distintos entre um par de nós de um digrafo esparso. Esse algorítmo é de $O(n)$, onde n é o número de não-terminais envolvidos em produções simples;

(ii) o Passo 2 é o Alg.2 para a construção da GOE. Esse algorítmo é de complexidade $O(pn)$, onde p é o número de produções e n é a soma dos comprimentos dos lados direitos. Como

$P \leqslant n$, temos complexidade $0(n^2)$.

(iii) o Passo 3 constroi a relação PRECEDE. Esta relação pode ser expressa como PRECEDE = $\lessdot$ + $\gtrdot$ + $\doteq$, onde estas são as relações de |*Wirth & Weber, 66*|. Conforme |*Hunt, 77*|,que detalharemos no próximo capítulo, é calculável por algorítmo de complexidade $0(n^2)$, onde n é a soma dos números de terminais e não-terminais;

(iv) o Passo 4 percorre todas as produções para cada par $(\underline{U}, a) \in (\underline{N} \times C')$, tornando o algorítmo de complexidade $0(pmn)$, com $p = \#(P')$, $m = \#(\underline{N})$, $n = \#(\Sigma')$. Como $p \leq C_1 n$ e $m \leq C_2 n$, para constantes C_1 e C_2, teremos complexidade $0(n^3)$;

(v) o Passo 5, por analogia ao Passo 4, terá complexidade $0(pmnr)$, com $p = \#(P')$, $m = \#(\underline{N})$, $n = \#(C')$, $r = \#(N)$. Por conseguinte $0(n^4)$;

(vi) em conclusão, temos um algorítmo $0(n^4)$.

Resta-nos apresentar o analisador sintático para GMT. Alguns problemas ainda não estão resolvidos, tais como as reduções da forma $A \overset{*}{\Longleftarrow} B$ e o armazenamento das triplas obtidas pelo construtor. Seguiremos precisamente a solução de |*Gries, 68*| que advoga a tradução das triplas em matriz de controle (a "*matriz de transição*") acoplada a um conjunto de subprogramas.

A geração da matriz e dos subprogramas é algorítmica mas não será necessário descrevê-la formalmente aqui bastando lembrar:

(i) como a matriz será bidimensional $(\underline{N} \times C)$, todas as tuplas que contiverem o par $(\underline{U}, a)$ serão tratadas pelo mesmo

subprograma;

(ii) a verificação do elemento da tupla correspondente ao não-terminal será feita sempre sobre o conjunto $SYMB^*(A) = \{B \mid A \overset{*}{\Longrightarrow} B\}$, quando todas as reduções diretas de $A \overset{*}{\Longleftarrow} B$ serão feitas de uma só vez;

(iii) as tuplas advindas das condições (i).a e (ii).a (ou seja, $X \rightarrow \amalg$ e $X \rightarrow \amalg A$) irão compor um subprograma que após verificar o não-terminal original, será:

" pop; MEIO:=X;
write (j); "

(iv) as tuplas advindas das condições (i).b e (ii).b (ou seja, $X \rightarrow \amalg a$ e $X \rightarrow \amalg Aa$) irão compor um subprograma que, após verificar o não-terminal, será:

" pop; push (X); MEIO:=ε;
write (j); SCAN; "

(v) as tuplas advindas das condições (i).c e (ii).c (ou seja, $X \rightarrow a$ e $X \rightarrow Aa$) irão compor um subprograma que, após verificar o não-terminal, será:

" push (X); MEIO:=ε;
write (j); SCAN; "

O analisador sintático da GMT é o programa composto pelos subprogramas gerados e inicialização.

Exemplo (XL)

$G = (N, L, P, S) \in \mathcal{G}_{GO}$

$N = \{PROG, STATE, VAR\}$ $C = \{'if', 'then', 'else', ':=', 'or', 'var'\}$

P = {1 PROG → STATE
2 PROG → if EXPR then STATE
3 STATE → if EXPR then STATE else STATE
4 STATE → var := EXPR
5 EXPR → EXPR or var
6 EXPR → var }

S = PROG

Construindo a GOE

0 S' ⟶ #PROG#
1 PROG ⟶ STATE
2 PROG ⟶ ifEXPRthen STATE
3 STATE ⟶ ifEXPRthenSTATEelse STATE
4 STATE ⟶ var:= EXPR
5 EXPR ⟶ EXPRorvar
6 EXPR ⟶ var
7 # ⟶ #
8 if ⟶ if
9 var ⟶ var
10 EXPRor ⟶ EXPR or
11 var:= ⟶ var :=
12 EXPRorvar ⟶ EXPRor var
13 #PROG# ⟶ # PROG #
14 ifEXPRthen ⟶ if EXPR then
15 ifEXPRthenSTATEelse ⟶ ifEXPRthen STATE else.

TUPLA	U	A	a	X	j	FORMA
1	#	PROG	#	S'	0	ESPECIAL
2	#	ε	if	if	8	U → a
3	#	ε	var	var	9	U → a
4	if	EXPR	then	ifEXPRthen	14	V → UAa
5	if	ε	var	var	9	U → a
6	if	EXPR	or	EXPRor	10	U → Aa
7	ifEXPRthen	STATE	#	PROG	2	A → Ua
8	ifEXPRthen	ε	if	if	8	U → a
9	ifEXPRthen	STATE	else	ifEXPRthenSTATEelse	15	V → UAa
10	ifEXPRthen	ε	var	var	9	U → a
11	ifEXPRthanSTATEelse	STATE	#	STATE	3	A → UA
12	ifEXPRthenSTATEelse	ε	if	if	8	U → a
13	ifEXPRthenSTATEelse	STATE	else	STATE	3	A → UA
14	ifEXPRthenSTATEelse	ε	var	var	9	U → a
15	var	ε	#	EXPR	6	A + U
16	var	ε	then	EXPR	6	A → U
17	var	ε	else	EXPR	6	A → U
18	var	ε	:=	var:=	11	V → Ua
19	var	ε	or	EXPR	6	A + U
20	var:=	EXPR	#	STATE	4	A + UA
21	var:=	EXPR	else	STATE	4	A + UA
22	var:=	ε	var	var	9	U → a
23	var:=	EXPR	or	EXPRor	10	U → Aa
24	EXPRor	ε	var	EXPRorvar	12	V → Ua
25	EXPRorvar	ε	#	EXPR	5	A → U
26	EXPRorvar	ε	then	EXPR	5	A → U
27	EXPRorvar	ε	else	EXPR	5	A → U
28	EXPRorvar	ε	or	EXPR	5	A + U

Analisador Sintático: (Suprimida saída do número de produções)

```
begin
push (#); MEIO:= ε ; SCAN; DONE:= false;
while not DONE do
  case MAT-TRANSICAO [top, SCANNED] of
  begin
 0: ERROR; DONE:= true;
 1: if MEIO=PROG or MEIO=STATE then
    begin write ("ACEITO"); DONE:=true; end
    else ERROR;
 2: if MEIO = E then begin push (if); SCAN; end else ERROR;
 3: if MEIO = E then begin push (~ar); SCAN; end else ERROR;
 4: if MEIO =  EXPR then begin pop; push (ifEXPRthen);
                                MEIO:= ε ; SCAN;          end
                    else ERROR;
 5: if MEIO =  EXPR then begin push (EXPRor);
                                MEIO:= E ; SCAN; end
                    else ERROR;
 6: if MEIO = STATE then begin pop; MEIO:=PROG end else ERROR;
 7: if MEIO = STATE then begin pop; push (ifEXPRthenSTATEelse);
                                MEIO:= ε ; SCAN;               end
                    else ERROR;
 8: if MEIO = STATE then begin pop; MEIO:=STATE end else ERROR;
 9: if MEIO = E then begin pop; MEIO:=EXPR end else ERROR;
10: if MEIO = E then begin pop; push (~ar:=)
                           SCAN;              end
                else ERROR;
11: if MEIO = E then begin pop; push (EXPRorvar);
                           SCAN;                end
    end;
end.
```

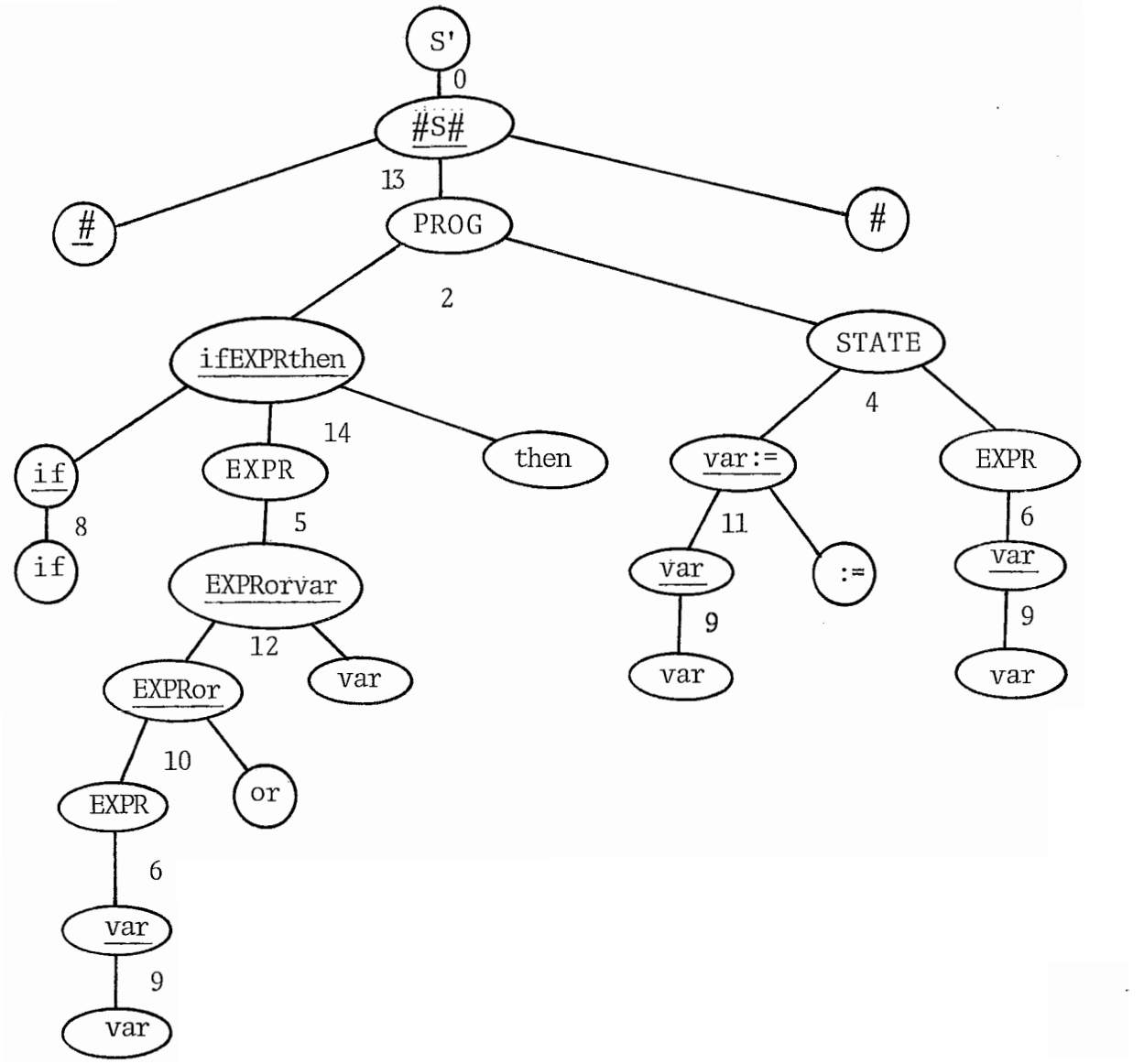
A matriz de transição para este reconhecedor será:

	#	if	then	else	var	:=	or
#	1	2	0	0	3	0	0
if	0	0	4	0	3	0	5
ifEXPRthen	6	2	0	7	3	0	0
ifEXPRthenSTATEelse	8	2	0	8	3	0	0
var	9	0	9	9	0	10	9
var:=	11	0	0	11	3	0	5
EXPRor	0	0	0	0	12	0	0
EXPRorvar	9	0	9	9	0	0	9

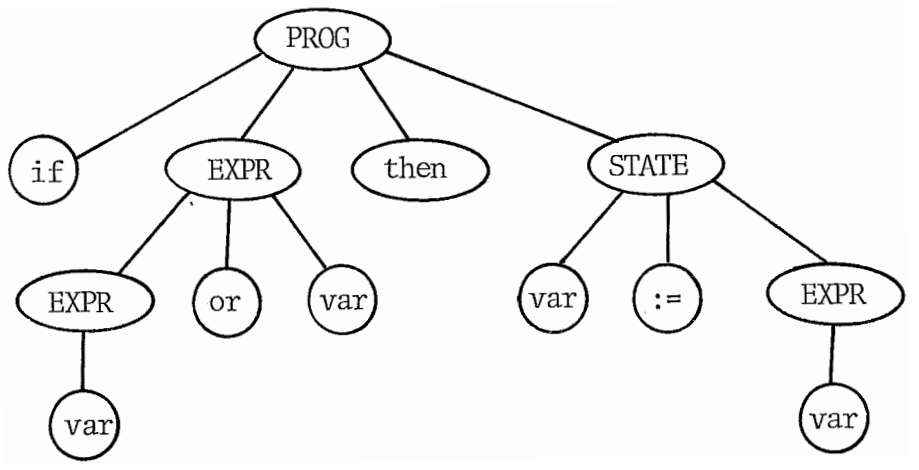
Análise da sentença "if var or var then var:=var"

PILHA	MEIO	ENTRADA	PARSE
#	E	if var or var then var := var #	
# if		var or var then var := var #	8
# if var		or var then var := var #	9
# if	EXPR		6
# if EXPRor	E	var then var := var #	10
# if EXPRorvar		then var := var #	12
# if	EXPR		5
# ifEXPRthen	E	var := var #	14
# ifEXPRthen var		:= var #	9
# ifEXPRthen var:=		var #	11
# ifEXPRthen var:= var		#	9
# ifEXPRthen var:=	EXPR	#	6
# ifEXPRthen	STATE	#	4
#	PROG	#	2
			13,0
			aceito

. O parse fornecido pelo analisador corresponde a seguinte árvore sintática:



Conforme XXI.(viii), na gramática original:



Note que, apesar do analisador sintático fornecer um parse esparso (não há qualquer tupla que ordene redução pela produção 1), o parse completo pode ser obtido facilmente, pois uma das condições das GMT é que $A \overset{*}{\Longrightarrow} B$ seja Única e, portanto, conhecida *"a priori"*. Temos agora um método que dispõe da velocidade de um analisador sintático esparso que pode, porém, fornecer a análise completa.

CAPÍTULO IV

— MATRIZES DE TRANSIÇÃO, OUTRA VEZ —

"Here 1'm bach again where 1 started. Richer by many experiences but poorer by many exploded convictions and many perished certainties".

A. Huxley

Nesta seção e nas seguintes vamos recomeçar a estabelecer novamente a teoria. Evidentemente o demonstrado no capítulo anterior é considerado válido. O fundamental agora não está no descobrimento de falhas ou incorreções no método ali apresentado. Mas trata-se de descobrir novas relações, a princípio desapercebidas, de conseguir caminhos mais curtos para os mesmos objetivos, de montar novamente o quebra-cabeças teórico de modo a reviver um processo de análise sintática virtualmente superado. A medida final da utilidade do que será construido se fará quando da comparação com os métodos mais modernos de 'análise sintática.

■ Teorema 5 (XLI)

Seja $G = (N \cup \underline{N} \cup \{S'\}, \Sigma \cup \{\#\}, P', S') \in \mathfrak{C}_{GMT}$.

$S \xrightarrow[dir]{*} \alpha \quad \blacksquare\!\triangleright \quad \alpha = \underline{X_1}\ \underline{X_2} \ldots \underline{X_n}\ A\ a_1\ a_2 \ldots a_m$,

$\underline{X_i} \in \underline{N}$, $\quad 1 \leqslant i \leqslant n$,

$A \in (N \cup \{\varepsilon\})$,

$$a_i \in (\Sigma \cup \{\varepsilon\}), \quad 1 \leqslant i \leqslant n \quad .$$

Demonstração: (por indução no comprimento da derivação)

Seja $S' \Longrightarrow \mathbf{a_0} \Longrightarrow \mathbf{a_1} \Longrightarrow \ldots \Longrightarrow \alpha_k \Longrightarrow \alpha$, a derivação direita de *a* em G. Esta derivação é Única devido à não-ambiguidade de $G \in \mathcal{C}_{GMT}$, conforme Teo. 4 (XXXVIII).

i) Base da indução

$\alpha_0 = \underline{\#S\#}$, por construção da GOE e, portanto, do formato

$\underline{X_1}\ \underline{X_2} \ldots \underline{X_n}\ A\ a_1\ a_2 \ldots a_m$ com $n = 1$,
$X_1 = \underline{\#S\#}$,
$A = \varepsilon$
$m = 1$
$a_1 = \varepsilon$

ii) Passo da indução

Seja $\alpha_i = \underline{X_1}\ \underline{X_2} \ldots \underline{X_n}\ A\ a_1\ a_2 \ldots a_m$, com $\mathbf{i \geqslant 1}$.

Conforme os 7 tipos de produção das GOE (XXXI), a forma sentencial α_{i+1} será:

a) se $A \neq \varepsilon$

1. $A \rightarrow B$ ▶ $\alpha_i \Rightarrow \alpha_{i+1} = \underline{X_1}\ \underline{X_2} \ldots \underline{X_n}\ B\ a_1\ a_2 \ldots a_m$
2. $A \rightarrow \underline{U}$ ▶ $\alpha_i \Rightarrow \alpha_{i+1} = \underline{X_1}\ \underline{X_2} \ldots \underline{X_n}\ \underline{U}\ a_1\ a_2 \ldots a_m$
3. $A \rightarrow \underline{U}B$ ▶ $\alpha_i \Rightarrow \alpha_{i+1} = \underline{X_1}\ \underline{X_2} \ldots \underline{X_n}\ \underline{U}\ B\ a_1\ a_2 \ldots a_m$

b) se $A = \varepsilon$

4. $\underline{X_n} \rightarrow a$ ▶ $\alpha_i \Rightarrow \alpha_{i+1} = \underline{X_1}\ \underline{X_2} \ldots a\ a_1\ a_2 \ldots a_m$
5. $\underline{X_n} \rightarrow Ba$ ▶ $\alpha_i \Rightarrow \alpha_{i+1} = \underline{X_1}\ \underline{X_2} \ldots Ba\ a_1 a_2 \ldots a_m$

6. $\underline{X_n} \rightarrow \underline{U}a$ ▶ $\alpha_i \Rightarrow \alpha_{i+1} = \underline{X_1}\ \underline{X_2} \ldots \underline{U}\ a\ a_1\ a_2 \ldots a_m$

7. $\underline{X_n} \rightarrow \underline{U}Ba$ ▶ $\alpha_i \Rightarrow \alpha_{i+1} = \underline{X_1}\ \underline{X_2} \ldots \underline{U}\ B\ a\ a_1 a_2 \ldots a_m$

Em todos os casos α_{i+1} é do formato especificado, c.q.d.

_____ ■

■ Teorema 6 (XLII)

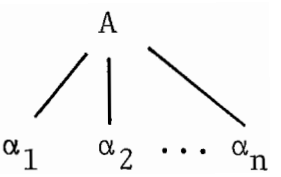
Seja $G \in \mathcal{C}_{GO}$ e $G' \in \mathcal{C}_{GMT}$ sua correspondente GOE, obtida pelo algorítmo 2 (XXIX), obedecendo a convenção de numeração das produções de modo que se as produções originais de G são numeradas (1,2, ..., p) as produções acrescidas à GOE são numeradas (0, p+1, p+2, ...).

Nessas condições G' cobre G à direita ("*right-covers*", |*Aho & Ullman, 72*|), ou seja: o parse ascendente $k_1 k_2 \ldots k_n$ da sentença y em G pode ser obtido do parse ascendente $\ell_1 \ell_2 \ldots \ell_n$ da sentença #y# em G', no caso pela eliminação de todo $\ell_i > p$.

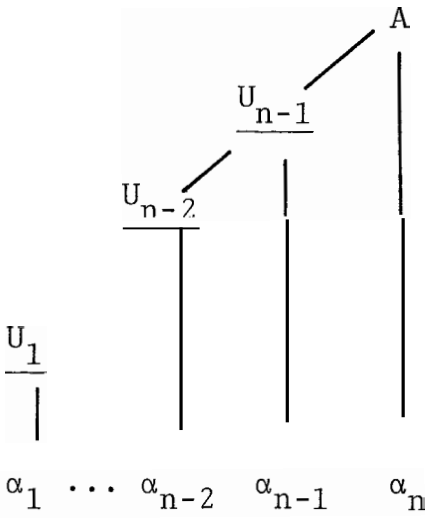
A cobertura à direita significa que as ações semânticas associadas às produções de G podem ser comandadas pelo analisador sintático de G'.

Demonstração: (por indução no número de nós internos da árvore sintática)

Pela propriedade XXXI.(viii) à cada árvore sintática de y em G corresponde uma Única árvore sintática de #y# em G' substituindo-se cada ramo



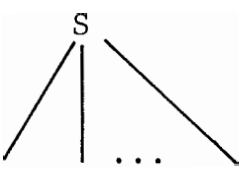
por



Um parse ascendente é obtido visitando-se a raiz de certa subarvore, visitando-se seus filhos da esquerda para a direita e emitindo-se a produção usada.

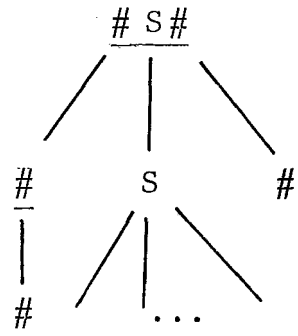
i) Base da indução: na raiz das árvores sintáticas

a) em G:



o parse será: (subarvore S) (S → a)

b) em G':



o parse será: (# → #) (subarvore S) (S → β) (#S# → #S#)

Como (# → #) e (#S# → # S #) são produções acrescidas logo tem numeração maior que p e (S → β) ∈ P' tem mesma numeração que (S → a) E P, a proposição é válida.

ii) Passo da indução

Para um nó genérico A na árvore sintática:

a) em G: $A \rightarrow \alpha_1\alpha_2 \ldots \alpha_n$, com parse

(subarvore α_1) (subarvore α_2)...(subarvore α_n)($A \rightarrow \alpha_1\alpha_2\ldots\alpha_n$)

b) em G': para o correspondente nó A, conforme a propriedade de XXXI.(viii), teremos o parse

(subarvore α_1) ($U_1 \rightarrow \alpha_1$)(subarvore α_2)($U_2 \rightarrow U_1\ \alpha_2$) ...

(subarvore a) ($U_{n-1} \rightarrow U_{n-2}$ an-1) (subarvore α_n)($A \rightarrow U_{n-1}\alpha_n$).

Como ($U_i \rightarrow U_{i-1}\ \alpha_i$) ∈ P' é acrescida tem numeração maior que p e ($A \rightarrow U_{n-1}\ \alpha_n$) ∈ P' tem mesma numeração que ($A \rightarrow \alpha_1\ \alpha_2 \ldots \alpha_n$) E P, a proposição é demonstrada.

Note que se α_n = Ba, então teremos uma produção ($U_n \rightarrow U_{n-1}\ \alpha_n$) E P' e será também acrescentada a produção

$(A \rightarrow \underline{U_n}) \in P'$, o que valida a demonstração, c.q.d..

Exemplo (XLIII)

$G = (N, \Sigma, P, S) \in \mathcal{C}_{GO}$

$P = \{1 : S \rightarrow aBc$

$2 : B \rightarrow b \}$

$p = 2$

$G' = (N \cup \underline{N} \cup \{S'\}, \Sigma \cup \{\#\}, P', S')$

$P' = \{0 : S' \rightarrow \underline{\#S\#}$

$1 : S \rightarrow \underline{aBc}$

$2 : B \rightarrow \underline{b}$

$3 : \underline{\#} \rightarrow \#$

$4 : \underline{a} \rightarrow a$

$5 : \underline{b} \rightarrow b$

$6 : \underline{\#S\#} \rightarrow \underline{\#}\ S\ \#$

$7 : \underline{aBc} \rightarrow \underline{a}\ B\ c\ \}$

$$S \underset{G}{\overset{1}{\Longrightarrow}} aBc \overset{2}{\Longrightarrow} abc$$

$$\underline{\#S\#} \underset{G'}{\overset{6}{\Rightarrow}} \underline{\#}\ S\ \# \overset{1}{\Rightarrow} \underline{\#}\ \underline{aBc}\ \# \overset{7}{\Rightarrow} \underline{\#}\ \underline{a}\ B\ c\ \# \overset{2}{\Rightarrow} \underline{\#}\ \underline{a}\ \underline{b}\ c\ \# \overset{5}{\Rightarrow} \underline{\#}\ \underline{a}\ b\ c\ \# \overset{4}{\Rightarrow}$$

$$\underline{\#}\ abc\ \# \overset{3}{\Rightarrow} \#\ abc\ \#$$

Parses ascendentes:

em G : 2 1

em G': 3 4 5 2 7 1 6

Estes dois teoremas iniciais visam esclarecer as principais consequências da execução da análise sintática sobre a gramática de operadores estendida:

i) os não-terminais estrelados, que serão o alfabeto da pilha do reconhecedor, funcionam como *abreviaturas* dos pedaços de lados direitos das produções, concentrando contexto a esquerda no topo da pilha. Note que o contexto descrito por um estrelado está limitado à produção, de forma que um analisador sintático que tome por base apenas o estrelado do topo da pilha pode levar a um conteúdo da pilha que não corresponde a qualquer prefixo correto de sentença da linguagem. Este fato terá implicações na recuperação de erros.

ii) as reduções (ou derivações) nas GOE podem ser divididas em duas classes: as reduções que correspondem a efetivas reduções na gramática original e aquelàs que apenas alteram conteúdo da pilha, aguardando a entrada da frase primária completa. Isto não é novidade nos analisadores ascendentes, todos derivados do algorítmo *"AVANÇA-REDUZ"* geral |*Aho & Ullman, 72*|. O que caracteriza, porém, as GOE é que mesmo a ação *"AVANÇA"*, significa uma redução (ou derivação) pela necessidade de concentrar a produção em uso no topo da pilha.

iii) o método necessita sempre três elementos para determinar a ação do reconhecedor: um não-terminal estrelado situado no topo da pilha, um não-terminal armazenado em variável especial e um terminal situado no inicio da sentença ainda por ser lida. Esta organização é particularmente desinteressante, qualquer que seja a maneira de armazenaras configurações admissíveis:

a) se armazenadas em formato tabela, obrigam o uso de matriz tridimensional altamente esparsa, com dimensões

(#(N) x #(N) x #(C)). Normalmente isto é inviável, mesmo para pequenas gramáticas;

b) se armazenadas em formato lista, obrigam à busca em pelo menos dois elementos distintos e prejudicam sensivelmente a eficiência do analisador;

c) se armazenadas em formato matriz de transição mais subprogramas – conforme a proposta de |*Gries, 68*| e sem dúvida a pior opção – além do espaço ocupado para armazenamento de milhares de instruções repetidas, acarretam enorme dificuldade de alteração do analisador;

Esta argumentação preliminar visa esclarecer os motivos da reforma geral no método e a base de sua reconstrução. Neste ponto começaremos a teoria outra vez.

4.1 – GRAMÁTICAS DE MATRIZES DE TRANSIÇÃO

Vamos começar com um exemplo que nos acompanhe no decorrer do capítulo. Este foi retirado propositalmente de |*Anderson, 73*| para comparação com outros métodos, mais adiante.

Exemplo (XLIV)

G_E=(N = {S, A, C, E, T, P, B}, #(N) = 7

Σ = {id, :=, if, then, else, +, *, (,), or} #(Σ) = 10

P, #(P) = 13

S)

P = { 1 : S → C
2 : S → A
3 : A → id := E
4 : C → if B then A
5 : C → if B then A else S
6 : E → T
7 : E → E + T
8 : T → P
9 : T → T * P
10 : P → (E)
11 : P → id
12 : B → B or id
13 : B → id }

$GO'_E = (N \cup \{S'\}, \Sigma \cup \{\#\}, P \cup \{0: S' \rightarrow \# S \#\}, S')$

(Obtida após o Passo 1 do Alg. 2)

$G'_E = (N', C', P', S')$ $\quad \#(N') = 20 \quad \#(\Sigma') = 11$

$\#(\underline{N}) = 13 \quad \#(P') = 27$

P' = {
0 : S' → #S#
1 : S → C
2 : S → A
3 : A → id:= E
4 : C → ifBthen A
5 : C → ifBthenAelse S
6 : E → T
7 : E → E+ T
8 : T → P
9 : T → T* P

10 :	P	→	(E)		
11 :	P	→	id		
12 :	B	→	Borid		
13 :	B	→	id		
14 :	#	→			#
15 :	id	→			id
16 :	if	→			if
17 :	(	→			(
18 :	E+	→		E	+
19 :	T*	→		T	*
20 :	Bor	→		B	or
21 :	#S#	→	#	S	#
22 :	id:=	→	id		:=
23 :	ifBthen	→	if	B	then
24 :	ifBthenAelse	→	ifBthen	A	else
25 :	(E)	→	(	E	)
26 :	Borid	→	Bor		id

O teorema seguinte pode ser visualmente verificado no exemplo.

Teorema 7 (XLV)

Seja $G^1=(N' C', P', S')$ uma gramática de operadores estendida construída pelo Alg. 2(XXIX), seguindo a convenção da numeração de produções adotada, e sejam $p'=\#(P')-1$ e $p=\#(P)$.

i) todas as produções de P' numeradas de 0 a p são das formas

(A → U) ou (A → U B) ou (A → B);

ii) existe k tal que

(a) todas as produções de P' numeradas de p+1 até k são das formas (U → a) ou (U → Ba);

(b) todas as produções de P' numeradas de k+1 até p' são das formas (U → Ua) ou (U → UBa);

Demonstração:

Imediata, por simples verificação do funcionamento do Alg. 2:

i) as produções numeradas de 0 a p são as próprias produções da gramática original (acrescidas de 0:S' → #S#), devidamente alteradas. Necessariamente são da forma A → *a*, e os formatos possíveis dos lados direitos são os citados, conforme a propriedade XXXI.(iv);

ii) no final do Passo 3 do Alg. 2, todas as produções são das formas (A → B), (A → Uα), (U → a) e (U → Ba), pela propriedade XXXI.(ii). Admitimos que a produção de maior número nesse instante seja a k-ésima produção. Os Passos 4 e 5 do algorítmo operam apenas sobre produções da forma (A → Ua). Logo qualquer nova produção criada será das formas (U_1 → Ua) ou (U_1 → UBa), com numeração maior que k. Isto demonstra a tese.

■

Sabemos, agora, que as produções de uma GOE podem ser divididas em 3 faixas, conforme seu tipo e numeração. 0 até p,

p+1 até k e k+1 até p'. No nosso exemplo, p=13, k=20 e p'=26. Relacionando este ponto com a análise sintática:

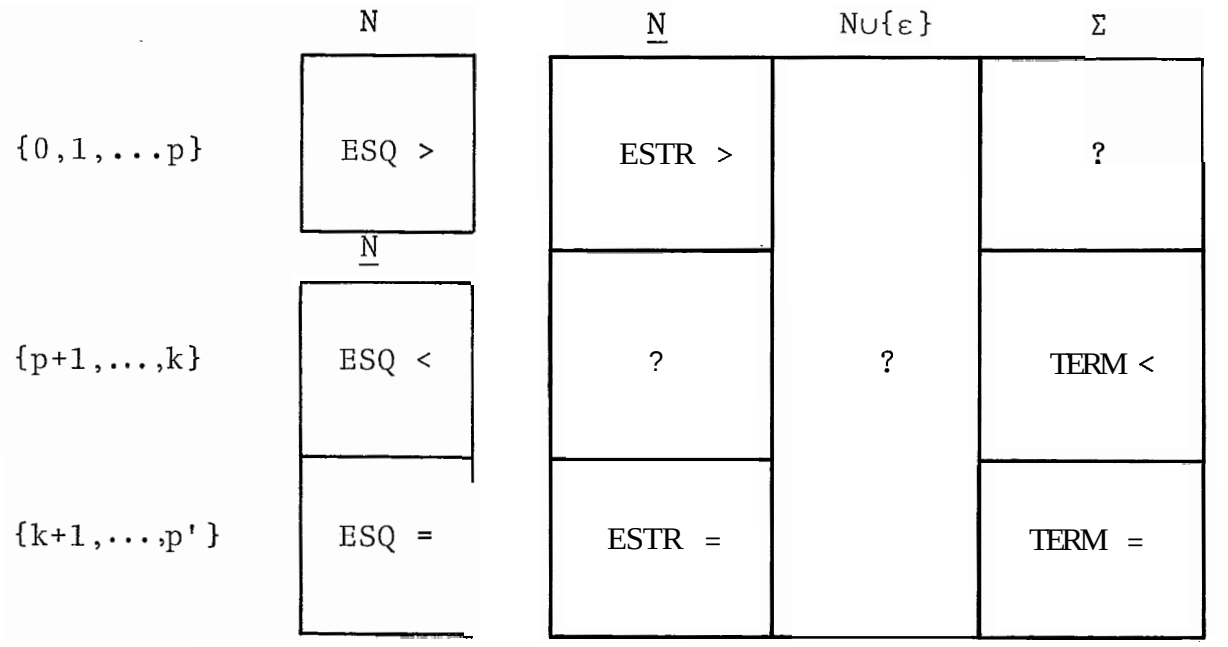
(a) ao se usar uma produção da primeira faixa teremos, também, uma redução na gramática original, conforme o Teorema 6;

(b) ao se usar uma produção da segunda faixa teremos uma ação "AVANÇA" do analisador;

(c) ao se usar uma produção da terceira faixa concentramos contexto esquerdo no topo da pilha.

Podemos agora partir para o cálculo da matriz de transição. Para tanto, é necessário que montemos um esquema de representação da gramática através de matrizes booleanas, uma vez que os elementos de cada configuração do analisador – o estrelado no topo da pilha, o não terminal na variável MEIO e o terminal no início da sentença a ser lida – em geral serão elementos de cada produção.

O pequeno esboço, a seguir, nos esclarecerá a situação de nosso quebra-cabeças e permitirá ver com maior clareza os passos a serem tomados. Essa representação da gramática em forma de relações visa obter as configurações (U, A, a) possíveis do analisador sintático.



Definição (XLVI)

MAIS ALGUMAS RELAÇÕES SOBRE CONJUNTOS GRAMATICAIS

i) RELAÇÃO ESQ > $\subseteq$ $\{0,1,\ldots,p\} \times N$

j ESQ> A $\Leftrightarrow$ $(j: A \to \alpha) \in P'$

ii) RELAÇÁO ESQ< $\subseteq$ $\{p+1,\ldots,k\} \times \underline{N}$

j ESQ< $\underline{U}$ $\Leftrightarrow$ $(j: \underline{U} \to \alpha) \in P'$

iii) RELAÇÃO ESQ= $\subseteq$ $\{k+1,\ldots,p'\} \times \underline{N}$

j ESQ= $\underline{U}$ $\Leftrightarrow$ $(j: \underline{U} \to \alpha) \in P'$

iv) RELAÇÃO ESTR> $\subseteq$ $\{0,1,\ldots p\} \times \underline{N}$

j ESTR> $\underline{U}$ $\Leftrightarrow$ $(j: A \to \underline{U}B) \in P'$, $B \in (N \cup \{\varepsilon\})$

v) RELAÇÃO ESTR= $\subseteq$ $\{k+1,\ldots,p'\} \times \underline{N}$

j ESTR= $\underline{U}$ $\Leftrightarrow$ $(j: \underline{U}_1 \to \underline{U}Ba) \in P'$, $B \in (N \cup \{\varepsilon\})$, $a \in \Sigma$

vi) RELAÇÃO TERM< $\subseteq$ $\{p+1,\ldots,k\} \times C$

j TERM< a $\Leftrightarrow$ $(j: \underline{U}_1 \to Ba) \in P'$, $B \in (N \cup \{\varepsilon\})$

vii) RELAÇÃO TERM= $\{k+1,\ldots,p'\} \times C$

j TERM= a $\Leftrightarrow$ $(j: \underline{U}_1 \rightarrow \underline{U}Ba) \in P'$, $B \in (Nu\{\varepsilon\})$

viii) RELAÇÃO NTERM $\subseteq$ $\{0,1,\ldots,p'\} \times (N \cup \{\varepsilon\})$

j NTERM A $\Leftrightarrow$ $(j: X \rightarrow \underline{U}\ A\ a) \in P'$, $X \in (N \cup \underline{N})$

$\underline{U} \in (\underline{N} \cup \{\varepsilon\})$

$a \in (\Sigma \cup \{\varepsilon\})$

ix) RELAÇÃO ESTR.NT $\subseteq$ $\underline{N} \times N$

$\underline{U}$ ESTR.NT A $\Leftrightarrow$ $(X \rightarrow \underline{U}\ A\ a) \in P'$, $X \in (\underline{N} \cup N)$

$a \in (\Sigma \cup \{\varepsilon\})$

As sete primeiras relações compõem a representação da gramática.

Propriedade (XLVII)

As relações ESQ>, ESQ<, ESQ=, ESTR>, ESTR=, TERM<, TERM= e NTERM contém apenas um elemento "*verdadeiro*" por linha.

Demonstração: Cada linha dessas produções corresponde a uma produção da gramática estendida e cada produção possui apenas um dos elementos do contradomínio.

Algumas vezes, com objetivos de armazenamento, chamaremos essas relações com apenas um elemento por linha de "*funções*".

Temos três frentes a atacar:

a) determinar as relações que fornecem o elemento terminal nas triplas correspondentes a faixa de produção de redução;

b) determinar as relações que fornecem o elemento estrelado nas triplas correspondentes a faixa de produções de avanço;

c) determinar as relações que permitam complementar a relação NTERM, de modo a contornar a "*transparência*" das produções simples;

Definição (XLVIII)

RELAÇÕES DE PRECEDÊNCIA EM GOE

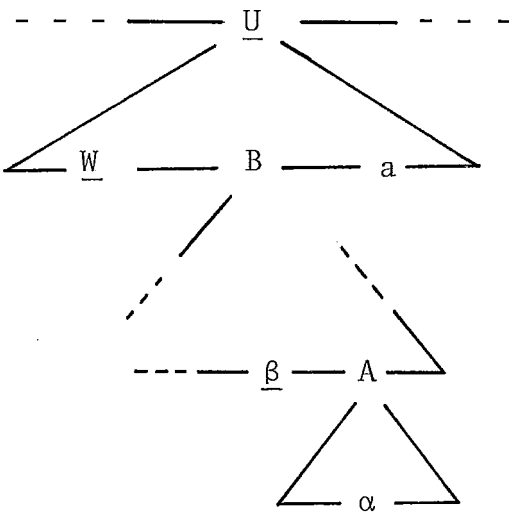
i) RELAÇÃO $\overset{*}{>} \subseteq \{1,2,\ldots,p\} \times C$

$j \overset{*}{>} a \Leftrightarrow (j: A \to \alpha) \in P' \quad , \quad A \in N$

$\wedge\ B \overset{*}{\Longrightarrow} \underline{\beta}A$

$\wedge\ (\underline{U} \to \underline{W}\ B\ a) \in P' \quad , \quad \underline{W} \in (\underline{N} \cup \{\varepsilon\})$

Veja-se a representação esquemática dessas condições:



A redução $A \Leftarrow a$ será efetuada ao depararmos com o símbolo a na entrada. Isto corresponde a definir o conjunto de seguidores ("*FOLLOW*") 'de A.

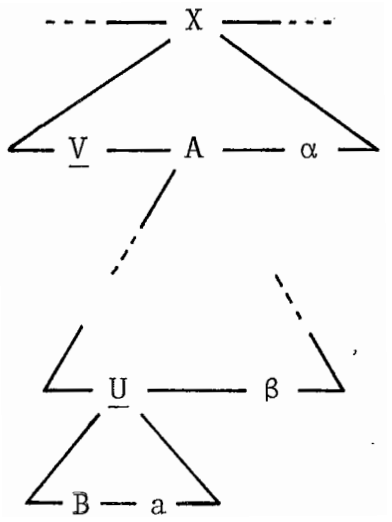
ii) RELAÇÃO $\lessdot \subseteq \{p+1, p+2, \ldots, k\} \times \underline{N}$

$j \lessdot \underline{V} \Leftrightarrow (j: \underline{U} \to Ba) \in P'$, $B \in (N \cup \{\varepsilon\})$

$\wedge (X \to \underline{V}A\alpha) \in P'$

$\wedge A \overset{+}{\Longrightarrow} Ba\beta$

Representando esquematicamente a situação



A admissão dos símbolos derivados de $\underline{U}$ se fará apenas quando o topo da pilha for $\underline{V}$. Em outras palavras determinamos o conjunto de estrelados predecessores de $\underline{U}$.

■

■ Teorema 8 CÁLCULO DE $\lessdot$ e $\gtrdot$ (XL IX)

i) $j \gtrdot a \Leftrightarrow j$ ESQ> $(\text{LASTNT}^*)^{-1}$ NT.TERM a

ii) $j \lessdot \underline{V} \Leftrightarrow j$ TERM< $(\text{ESTR.NT FIRSTNT}^* \text{ FIRSTTERM})^{-1} \underline{V}$

Demonstração: Aplicação das definições das relações, lembrando que $A \underset{G'}{\Longrightarrow} a \Leftrightarrow A \underset{G}{\Longrightarrow} \alpha$.

■

Vejamos, agora, o problema das produções simples.

■ Definição RELAÇÃO MEIO (L)

RELAÇÃO MEIO ⊂ {0,1,...,p'} x N

j MEIO A ◁■▷ (j: X → ⊔ B a) ∈ P' , X ∈ N'

⊔ ∈ (N ∪ {ε}),

a ∈ (C u {ε})

∧ (B $\overset{*}{\Longrightarrow}$ A)

_____ ■

■ Teorema 9 CÁLCULO DE MEIO (LI)

j MEIO A ◁■▷ j NTERM SYMB* A

Demonstração: Aplicação da definição das relações.

_____ ■

Com o ferramenta1 que definimos até aqui estamos aptos a conseguir o conjunto de triplas (⊔, A, a) que representam configurações admissíveis do analisador sintático. É muito importante notar que a metodologia adotada baseou-se totalmente no uso de relações, o que nos permite garantir que o processo geral terá complexidade máxima $0(n^2)$, onde n é o número total de símbolos envolvidos nas relações. A verificação desta complexidade será feita por ocasião da definição do construtor mais adiante.

Vejamos, então, a equivalência desta definição com a proposta por |*Gries, 68*| .

Redefinição

GRAMÁTICAS DE MATRIZES DE TRANSIÇÃO (LII)

Uma gramática $G = (N', C', P', S')$ de operadores estendida, reduzida, será dita gramática de matrizes de transição ou, equivalentemente, $G \in \mathcal{G}_{GMT}$ se:

i) para cada tripla $(\underline{U}, A, a) \in (N \times (N \cup \{\varepsilon\}) \times C)$ no máximo uma das três seguintes condições é verdadeira:

(a) existe uma Única produção – numerada j – tal que

$(j: B \to \underline{U}\ C) \in P'$, $C \in (N \cup \{\varepsilon\})$ ⇒

$(j\ \mathrm{ESTR}>\ \underline{U}) \wedge (C \overset{*}{\Longrightarrow} A)$

$\wedge\ (j\ \mathrm{MEIO}\ A)$

$\wedge\ (j \gg a)$

(b) existe uma Única produção – numerada j – tal que

$(j: \underline{V} \to C_a) \in P'$, $C \in (N \cup \{\varepsilon\})$ ⇒

$(j \ll U)$

$\wedge\ (j\ \mathrm{MEIO}\ A)$

$\wedge\ (j\ \mathrm{TERM}<\ a) \wedge (C \overset{*}{\Longrightarrow} A)$

(c) existe uma Única produção – numerada j – tal que

$(j: \underline{V} \to \underline{U}\ C\ a) \in P'$, $C \in (N \cup \{\varepsilon\})$ ⇒

$(j\ \mathrm{ESTR}=\ \underline{U}) \wedge (C \overset{*}{\Longrightarrow} A)$

$\wedge\ (j\ \mathrm{MEIO}\ A)$

$\wedge\ (j\ \mathrm{TERM}=\ a)$

ii) para qualquer $A,B \in N$ se $A \overset{*}{\Longrightarrow} B$ então essa derivação é Única. ■

Teorema 10 (LIII)

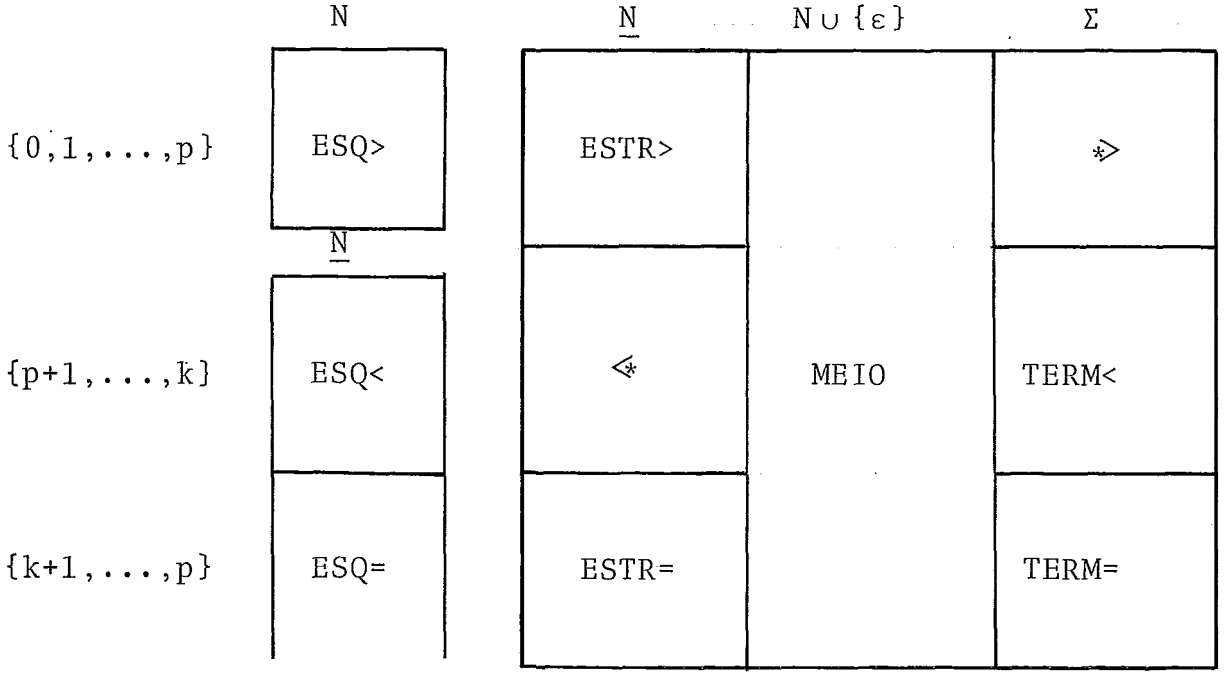
A definição original das GMT (XXXVII) e a redefinição das GMT (LII) corresponde à mesma classe de gramáticas.

Demonstração: Basta verificar que as condições impostas às GMT na redefinição (LII.(i), a, b, c e LII.(ii)) repetem as condições impostas na definição original (XXXVII.(i),.(ii), a, b, c e .(iii), a, b, c), usando a nova notação de relações (definidas em XLVI, XLVIII e L).

Vale a pena exemplificar o conjunto de relações que foram definidas para a gramática do exemplo XLIV.

Exemplo (continuação do exemplo XLIV) (LIV)

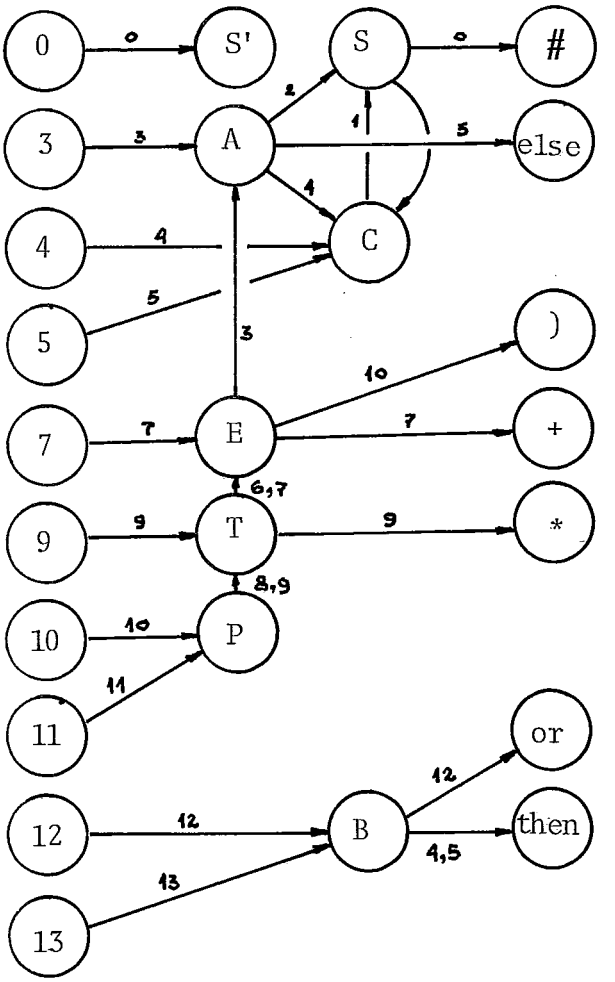
Esboço



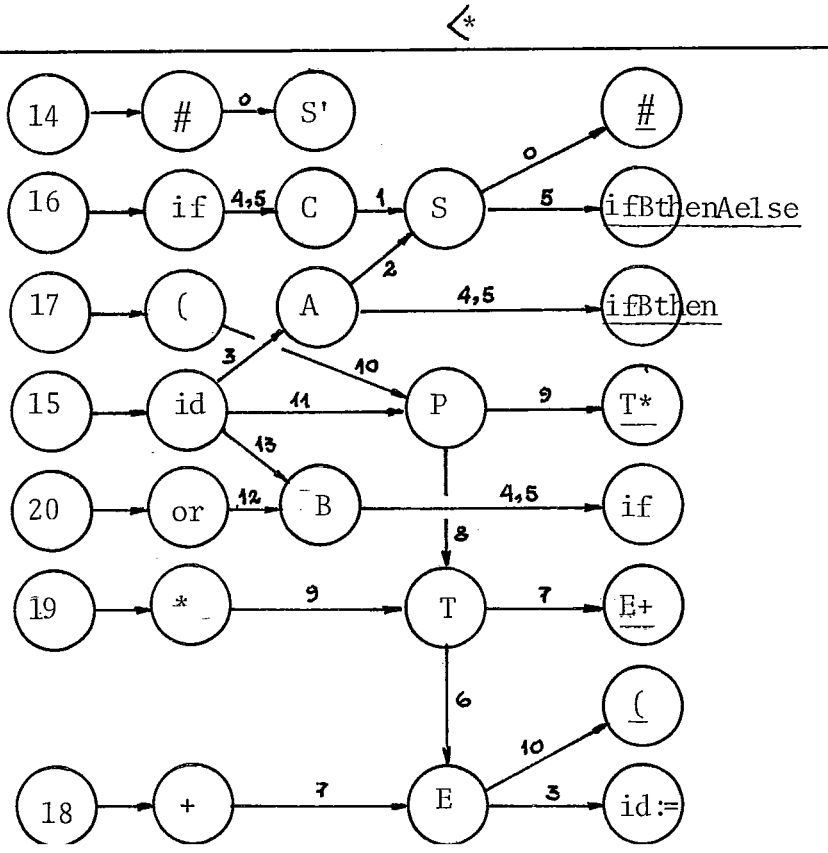
Aproveitamos a ocasião para calcular as relações, através de grafos. Os números nas arestas indicam o número da produção na gramática original que provocou a relação

ENTR>	MEIO
S	ε
id:=	E,T,P
ifBthen	A
ifBthenAelse	S,A,C
E+	T,P
T*	P
(E)	ε
id	ε
Borid	ε
id	ε

*>



TERM<	MEIO
#	ε
if	ε
(	ε
id	ε
or	B
*	T,P
+	E,T,P



ESTR=	MEIO		TERM=
#	S,A,C	21 —0→	#
id	ε	22 —3→	:=
if	B	23 —4,5→	then
ifBthen		24 —5→	else
(	E,T,P	25 —10→	)
Bor	ε	26 —12→	id

PRODUÇÃO	N													N ∪ {ε}								Σ										
	1	2	3	4	5	6	7	8	9	10	11	12	13	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	9	10	11
0								X						X																		
1																	X															
2																X																
3									X									X	X	X		X					X					
4										X						X						X										
5											X				X	X	X					X										
6																			X	X												
7					X														X	X		X					X	X			X	
8																				X												
9						X														X		X					X	X	X		X	
10												X		X								X					X	X	X		X	
11		X												X								X					X	X	X		X	
12													X	X											X							X
13		X												X											X							X
14														X								X										
15	X		X	X	X	X			X	X	X			X									X									
16	X									X				X										X								
17			X	X	X				X					X											X							
18			X						X									X	X	X						X						
19			X	X					X										X	X							X					
20		X																			X											X
21	X														X	X	X					X										
22		X												X									X									
23			X																		X			X								
24										X						X									X							
25				X														X	X	X						X						
26								X						X									X									

Códigos:

N : 1 #
2 id
3 if
4 (
5 E+
6 T*
7 Bor
8 #S#
9 id:=
10 ifBthen
11 ifBthenAelse
12 (⋿)
13 Borid

(N ∪ {ε}): 1 ε
2 S
3 A
4 C
5 E
6 T
7 P
8 B

Σ : 1 #
2 id
3 :=
4 if
5 then
6 else
7 +
8 *
9 (
10)
11 or

Note que as informações necessárias ao analisador sintático ocupam $\#(P') \times \#(N \cup N \cup \{\varepsilon\} \cup C)$ bits. Neste exemplo 864 bits (108 bytes).

Outra informação interessante que podemos extrair das relações é a frequência com que cada símbolo ocorre em configurações do analisador. Note que podemos ignorar as produções 0,1, 2,6,8 (produções simples) e 14 (produção inicial). Neste caso teremos 37 triplas na zona "REDUZ", 27 triplas na zona *"AVANÇA"* e 10 triplas na zona *"CONCENTRA"*, em um total de 74 configurações admissíveis (pois como veremos mais adiante esta gramática pertence à classe das GMT). A ocorrência dos símbolos é dada pela tabela a seguir.

ESTRELADOS			NÃO-TERMINAIS			TERMINAIS		
CÓDIGO	SÍMBOLO	FREQ.	CÓDIGO	SÍMBOLO	FREQ.	CÓDIGO	SÍMBOLO	FREQ.
1	#	5	1	ε	30	1	#	15
2	id	8	2	S	2	2	id	9
3	if	3	3	A	4	3	:=	1
4	(	10	4	C	2	4	if	2
5	E+	12	5	E	5	5	then	3
6	T*	7	6	T	12	6	else	9
7	Bor	1	7	P	17	7	c	11
8	#S#	0	8	B	2	8	*	9
9	id:=	13				9	(	4
10	ifBthen	3				10	)	8
11	ifBthenAelse	5				11	or	3
12	(E)	5						
13	Borid	2						

PRODUÇÃO	FREQ.	PRODUÇÃO	FREQ.	PRODUÇÃO	FREQ.
3	6	15	8	21	3
4	1	16	2	22	1
5	3	17	4	23	1
7	8	18	6	24	1
9	5	19	6	25	3
10	5	20	1	26	1
11	5				
12	2				
13	2				

PRODUÇÃO DA GRAMÁTICA ORIGINAL X FREQUÊNCIA

P	FREQ.	P	FREQ.	P	FREQ.
0	3	5	7	10	12
1	0	6	0	11	13
2	0	7	14	12	4
3	15	8	0	13	10
4	4	9	11		

Total maior que 74 pois algumas triplas podem advi'r de várias produções

Estas estatísticas são interessantes para que se possa esboçar uma medida dos pontos fracos da gramática com vistas a recuperação de erros. Outro ponto, este muito importante, que se evidencia é alta ocorrência de configurações sem não-terminal (indicado como "ε"), cujo valor típico empiricamente obtido é da ordem de 50%. Veremos, também, mais tarde como isto pode ser utilizado para fins de armazenamento.

4.2 – ANÁLISE SINTÁTICA EM GMT, OUTRA VEZ

Na seção anterior redefinimos a classe das GMT, através do uso de diversas novas relações, obtendo uma matriz que contém as informações necessárias para a análise sintática. Nesta seção iremos operar sobre essa matriz, torná-la adequada à consulta durante a análise sintática, descrever o construtor da tabela de controle e fornecer o algorítmo do analisador.

O problema agora é que, durante a análise sintática dispomos da configuração do analisador – a tripla (LL, A, a) – e queremos obter a ação a ser feita. Nossa matriz atual não é adequada para tal fim.

Definição CONJUNTO DE ESTADOS (LV)

Seja $G' \in \mathcal{C}_{GMT}$. O conjunto de estados de G' será

$$E=\{(LL,A) \mid (A=\varepsilon) \vee (S' \stackrel{+}{\Rightarrow} \alpha\, LL\, A\, \gamma)\} \subseteq (N \times (N \cup \{\varepsilon\}))$$

Tal como foi feito com as produções vamos admitir que os estados sejamnumerados (1, 2, ..., e).

Teorema 11 (LVI)

Seja E o conjunto de estados de $G' \in \mathcal{C}_{GMT}$.
Temos que e = #(E) é igual ao número de pares (LL, A) tais que (A = ε) OU (LL ESTR.NT FIRSTNT* A).

Demonstração:

Por definição (LV) o número de estados é o número de elementos do conjunto

$$E = \{(\underline{U}, A) \mid (A=\varepsilon) \vee (S' \overset{+}{\Longrightarrow} \alpha\ \underline{U}\ A\ \gamma)\}$$

Resta-nos demonstrar que

$$S' \overset{*}{\Longrightarrow} \alpha\ \underline{U}\ A\ \gamma \quad \blacklozenge \quad \underline{U}\ \text{ESTR.NT}\quad \text{FIRSTNT*}\quad A$$

Como, por L.(i) e XI.(i):

$$\underline{U}\ \text{ESTR.NT}\quad B \quad \blacklozenge \quad (X \to \underline{U}\ B\ a) \in P'$$
$$X \in (N \cup \underline{N}) \ , \quad a \in (\Sigma \cup \{\varepsilon\})$$

$$B\ \text{FIRSTNT}\quad A \quad \blacklozenge \quad (B \to Aa) \in P'$$

a, ainda, por XXXI.(vi),

$$(B \to An) \in P \quad \blacktriangleright \quad B \overset{*}{\underset{G'}{\Longrightarrow}} A\alpha$$

segue, que

$$X \underset{G'}{\Longrightarrow} \underline{U}\ B\ a \overset{*}{\Longrightarrow} \underline{U}A\alpha$$

Como a gramática é reduzida

$$S' \overset{*}{\Longrightarrow} \alpha\ X\ \gamma \overset{*}{\Longrightarrow} \alpha\ \underline{U}\ A\ \alpha\ \gamma$$

c.q.d.

———————— ■

Note que o cálculo de (ESTR.NT FIRSTNT*) é um dos cálculos parciais de $\mathcal{G}$. Dispomos, portanto, de uma forma de obter o número de estados e podemos também estabelecer sua função de numeração.

Definição (LVII)

FUNÇÃO GOTO

GOTO : $(\mathbb{N} \times (N \cup \{E\})) \longrightarrow \{1,2,...,e)$, com domínio E.

i) Seja $\underline{n} = \#(\mathbb{N})$

GOTO $(U, E) \in \{1,2,...,\underline{n}\}$

Ou seja, iremos numerar os estados $(\mathbb{U}, E)$ com os primeiros $\underline{n}$ inteiros positivos;

ii) para cada par $(\mathbb{U}, A)$ tal que

$\underline{U}$ ESTR.NT FIRSTNT" A

GOTO $(\mathbb{U}, A) \in \{\underline{n}+1,...,e\}$

O nome foi escolhido por analogia ao nome da função com domínio semelhante e mesmo nome encontrada nos analisadores tipo LR. Como veremos mais adiante esta analogia serve bastante bem para fins de compreensão e avaliação. Entretanto, ressaltemos claramente que a função GOTO das GMT *não corresponde* a uma *"ação"* do analisador sendo propriamente uma função de numeração para permitir um nível de indireção na tabela (eliminando correspondentemente a necessidade de uma tabela tridimensional).

Exemplo (continuação do exemplo LIV) (LVIII)

Temos inicialmente a relação

U ESTR.NT FIRSTNT* A

ESTR. NT	FIRSTNT*						
	S	A	C	E	T	P	B
#	X	X	X				
id							
if							X
(				X	X	X	
E+					X	X	
T*						X	
Bor							
#S#							
id:=				X	X	X	
ifBthen		X					
ifBthenAelse	X	X	X				
(E)							
Bor id							

Numerando conforme o definido

GO TO							
	ε	S	A	C	E	T	P	B
#	1	14	15	16				
id	2							
if	3							17
(	4				18	19	20	
E+	5					21	22	
T*	6						23	
Bor	7							
#S#	8							
id:=	9				24	25	26	
ifBthen	10		27					
ifBthenAelse	11	28	29	30				
(E)	12							
Borid	13							

Esta primeira função efetuará a transformação '.do par estrelado/não-terminal para o número do estado após uma redução. Definamos agora as demais funções que controlarão o analisador sintático que estamos a propor.

■ Definição (LIX)

FUNÇÕES AÇÃO E AVANÇAREDUZ

AÇAO: $(E \times \Sigma) \longrightarrow \{ \langle^*, \stackrel{*}{=} \quad ^*\rangle, \text{'pare'} \}$

AVANÇAREDUZ: $(E \times \Sigma) \longrightarrow \{0,1,\ldots,\max(\#(E), \#(P))\}$

Seja o par $(i,a) \in (E \times C)$, onde $i = (L, A)$.

i) para $0 \leq j \leq p$:

$(j \quad ESTR> L)$

$\wedge(j \quad MEIO \quad A)$

$\wedge(j \quad *> \quad a)$ ■▷ AÇÃO $(i,a) = *>$

■▷ AVANÇAREDUZ $(i,a) = j$

ii) para $p+1 \leqslant j \leqslant k$:

$(j \quad <* \quad U)$

$\wedge(j \quad MEIO \quad A)$

$\wedge(j \quad TERM< a)$

$\wedge(j \quad ESQ< V)$ ■▷ AÇÃO $(i, a) = <*$

■▷ AVANÇAREDUZ $(i,a) = GOTO(V, E)$.

iii) para $k+1 \leqslant j \leqslant p'$:

$(j \quad ESTR= L)$

$\wedge(j \quad MEIO \quad A)$

$\wedge(j \quad TERM= a)$

$\wedge(j \quad ESQ= V)$ ■▷ AÇÃO (i,a)= se $j=k+1$ então 'pare'

senão $\stackrel{*}{=}$

$\blacksquare\!\!>$ AVANÇAREDUZ (i,a) = GOTO($\underline{V}$,E)

Note que em (ii) e (iii) $\underline{V}$ é Único, conforme a propriedade (XLVII).

(iv) em qualquer outro caso:

AÇÃO (i,a) = não definido

AVANÇAREDUZ (i,a) = não definido

———— ■

Note que a função AVANÇAREDUZ para as ações $\lessdot$ e $\overset{*}{=}$ indica diretamente qual o estado seguinte do analisador. Quando a ação é $\gtrdot$ – indicando redução – nos fornece o número da produção usada *na gramática original*. Evidentemente, uma quarta função se faz necessária para conseguirmos reencetar a análise.

■ Definição (LX)

FUNÇÃO ESQUERDO

ESQUERDO : {0,1,...,p} ⟶ N

j ESQ> A $\blacksquare\!\!>$ ESQUERDO (j) = A

Pela propriedade (XLVII), A é Único.

———— ■

Vamos agora fornecer o algorítmo analisador sintático, de modo que a utilização dessas funções possa ser melhor entendida no exemplo que lhe segue.

Algorítmo 4 (LXI)

ANALISADOR SINTÁTICO PARA GMT, MODIFICADO.

Comentário:

- Seja ω a sentença a ser analisada e seja *"SCAN"* a operação de leitura do primeiro símbolo de ω para a variável ENTRADA.
- Sejam *"POP"* e *"PUSH(i)"* as operações de retirar e colocar os símbolos na pilha.
- Seja G' e $\mathcal{G}_{GMT}$ a gramática de operadores estendida e seja # o delimitador de sentença.

Passo 1: (INICIALIZAÇÃO)

PUSH (#) ; Faça MEIO = e; SCAN ;

Passo 2: (CICLO DE ANÁLISE)

Faça X = topo da pilha;

Se MEIO $\neq$ E então

se GOTO (X, MEIO) ="não definido" execute o passo 3

senão faça X = GOTO (X, MEIO);

Caso AÇÃO (X, ENTRADA) igual a:

$\lessdot$: PUSH (AVANÇAREDUZ (X, ENTRADA));

Faça MEIO = E; SCAN;

Repita o passo 2;

$\overset{*}{=}$:POP;

PUSH (AVANÇAREDUZ (X, ENTRADA));

Faça MEIO = E; SCAN;

Repita o passo 2;

*> : Faça J = AVANÇAREDUZ (X, ENTRADA); escreva J;

POP;

Faça MEIO = ESQUERDO (J);

Repita o passo 2;

'pare' : escreva "ACEITO";

PARE;

mão definido: excute o passo 3;

Passo 3: (SENTENÇA INCORRETA)

Escreva "REJEITADO" e PARE;

■

■ Exemplo (Continuação do exemplo LIV) (LXII)

As tabelas a seguir apresentadas foram construídas seguindo:

a) para GOTO, a definição LVII e o exemplo LVIII.

Note que a coluna "ε" da função GOTO é desnecessária uma vez que os primeiros n = # (N) estados tem o mesmo número que os correspondentes não-terminais estrelados.

b) para AÇÃO e AVANÇAREDUZ, a definição LIX, usando os grafos do exemplo LIV.

c) para ESQUERDO, a definição LX, usando o grafo do exemplo LIV.

d) o algorítmo construtor dessas funções será objeto da próxima seção.

	AÇÃO / AVANÇAREDUZ											GOTO							ESQ
	#	id	:=	if	then	else	+	*	(	)	or	S	A	C	E	T	P	B	
1		⋖2		⋖3								14	15	16					S
2	⋗11		=9		⋗13	⋗11	⋗11	⋗11		⋗11	⋗13								S
3		⋖2																17	A
4		⋖2							⋖4						18	19	20		C
5		⋖2							⋖4							21	22		C
6		⋖2							⋖4								23		E
7		=13																	E
8																			T
9		⋖2							⋖4						24	25	26		T
10		⋖2											27						P
11		⋖2		⋖3								28	29	30					P
12	⋗10					⋗10	⋗10	⋗10		⋗10									B
13					⋗12						⋗12								B
14	PARE																		
15	PARE																		
16	PARE																		
17					=10						⋖7								
18							5			=12									
19							5	⋖6		=12									
20							⋖5	⋖6		=12									
21	⋗7					⋗7	⋗7	⋖6		⋗7									
22	⋗7					⋗7	⋗7	⋖6		⋗7									
23	⋗9					⋗9	⋗9	⋗9		⋗9									
24	⋗3					⋗3	⋖5												
25	⋗3					⋗3	⋖5	⋖6											
26	⋗3					⋗3	⋖5	⋖6											
27	⋗4					=11													
28	⋗5																		
29	⋗5																		
30	⋗5																		

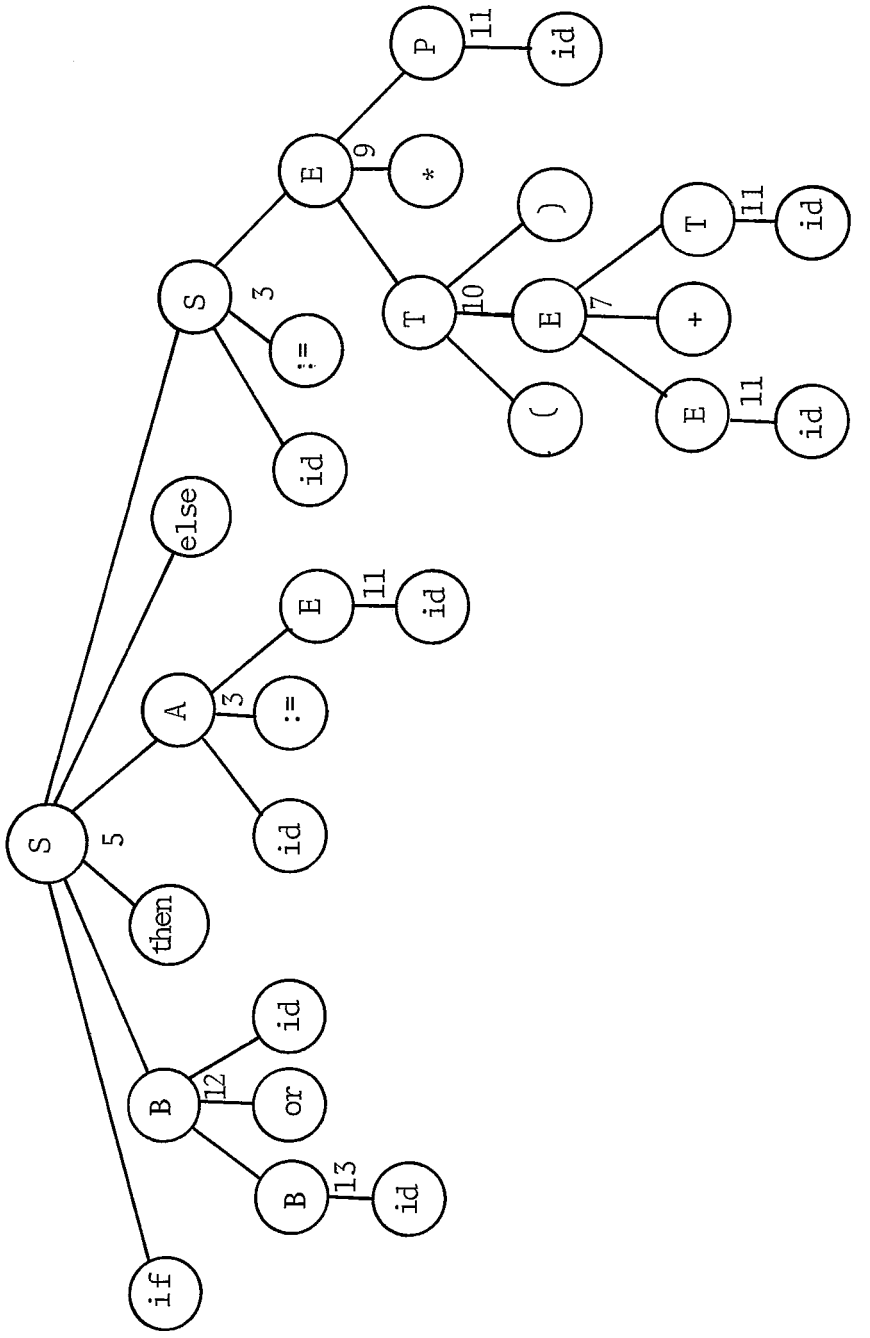
CÓDIGO DOS ESTRELADOS	
1	#
2	id
3	if
4	(
5	E+
6	T*
7	Bor
8	#S#
9	id:=
10	ifBthen
11	ifBthenAelse
12	(E)
13	Borid

Análise sintática de:

if id or id
then id := id
else id := (id + id) * id #

P I L H A	MEIO	ENTRADA	AÇÃO	PARSE
1	ε	if	⋖ 3	
1 3	ε	id	⋖ 2	
1 3 2	ε	or	⋗ 13	13
1 3	B	or	⋖ 7	
1 3 7	ε	id	≐ 13	
1 3 13	ε	then	⋗ 12	12
1 3	B	then	≐ 10	
1 10	ε	id	⋖ 2	
1 10 2	ε	:=	≐ 9,	
1 10 9	ε	id	⋖ 2	
1 1 0 9 2	ε	else	⋗ 11	11
1 10 9	P	else	⋗ 3	3
1 10	A	else	≐ 11	
1 11	ε	id	⋖ 2	
1 11 2	ε	:=	≐ 9	
1 11 9	ε	(	⋖ 4	
1 1 1 9 4	ε	id	⋖ 2	
1 1 1 9 4 2	ε	+	⋗ 11	11
1 1 1 9 4	P	+	⋖ 5	
1 1 1 9 4 5	ε	id	⋖ 2	
1 1 1 9 4 5 2	ε	)	⋗ 11	11
1 1 1 9 4 5	P	)	⋗ 7	7
1 1 1 9 4	E	)	≐ 12	
1 11 9 12	ε	*	⋗ 10	10
1 11 9	P	*	⋖ 6	
1 1 1 9 6	ε	id	⋖ 2	
1 1 1 9 6 2	ε	#	⋗ 11	11
1 1 1 9 6	P	#	4 9	9
1 11 9	T	#	⋗ 3	3
1 11	A	#	⋗ 5	5
1	C	#	PARE	ACEITO

Árvore sintática "*espatsa*" construída pelo analisador:



Note que apesar de termos obtido um parse esparso 'da sentença, não há qualquer dificuldade em obtermos o parse 'completo, pois uma das condições para gramáticas de matrizes de transição é que a derivação em produções simples seja sempre Única.

No nosso exemplo, a seguinte tabela efetua a conversão entre os parses:

Não-Terminal esperado	S	A	C	E	T	P	B
S							
A	2						
C	1						
E							
T				6			
P					8		
B							

Não-Terminal Usado

Produção Omitida

Com isso a obtenção do parse completo é trivial:

ESPARSO : 13,12,11, ,3,11, ,11, ,7,10, ,11,9, ,3, ,5
COMPLETO: 13,12,11,8,6,3,11,8,6,11,8,7,10,8,6,11,9,6,3,2,5,1

A alteração correspondente poderia ser feita no próprio analisador sintático mas, como já observamos, geralmente não haverá interesse.

4.3 – CONSTRUTOR DE ANALISADORES SINTÁTICOS PARA GMT

Nesta seção iremos fornecer os algorítmos que verificam a pertinência de uma gramática à classe das gramáticas de matrizes de transição e constroem as funções GOTO, AÇÃO, AVANÇA

REDUZ e ESQUERDO.

Inicialmente vamos tratar do algorítmo que verifica a unicidade das derivações em produções simples. O algorítmo a seguir é uma variação do algorítmo que calcula o fechamento transitivo reflexivo de uma relação binária. Inicialmente, o algorítmo cria um grafo representando as produções simples e, a seguir, efetua o seu fechamento. Note que o algorítmo já fornece como saída a relação SYMB* que será necessária para o construtor.

Algorítmo 5 (LXIII)

VERIFICAÇÃO DA UNICIDADE DE $A \stackrel{*}{\Longrightarrow} B$
E CONSTRUÇÃO DE SYMB*

Comentário:

SYMB* $\subset (N \times N)$ será tratada como um vetor de conjuntos SYMB*(A), com $A \in N$.

Passo 1: (constroi grafo GSIMPLES)

Para toda produção $(A \rightarrow B) \in P$ da gramática original crie, se já não existirem, em GSIMPLES: o nó A, o nó B e a aresta (A,B);

Passo 2: Crie, o conjunto MARCADO e os conjuntos SYMB*(A), para todo $A \in N$, inicialmente vazios;

Crie o nó Ω em GSIMPLES;

Passo 3: Crie uma pilha de arestas (A,B) onde os nós da aresta serão indicados por TOPO1 e $TOPO_2$;

Para todo nó X em GSIMPLES, tal que $X \notin$ MARCADO empilhe (X, Ω) e execute os passos 4 e 5; após varrer o conjunto de nós, pare;

Passo 4: Se TOPO1 $\notin$ MARCADO então:

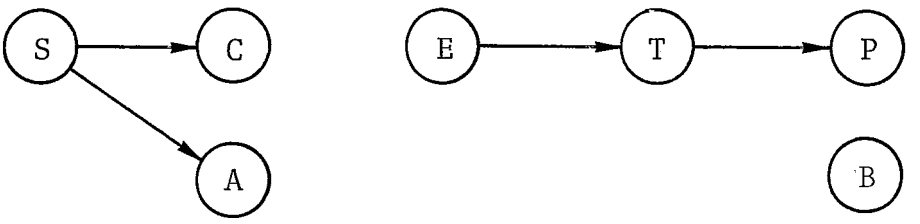
- faça SYMB* (TOPO1) = SYMB*(TOPO1) u {TOPO1};
- faça MARCADO = MARCADO u {TOPO1};
- para todo Y tal que (TOPO1, Y) em GSIMPLES, empilhe (Y, TOPO1) e execute os passos 4 e 5;

Passo 5: Se SYMB*(TOPO2) $\cap$ SYMB*(TOPO1) # Φ e TOPO2 $\neq$ Ω então escreva "AMBIGUIDADE EM PRODUÇÕES SIMPLES";
Faça SYMB*(TOPO2) = SYMB*(TOPO2) $\cup$ SYMB*(TOPO1);
Desempilhe;

———— ■

■ Exemplo (continuação do exemplo LXII) (LXIV)

As produções simples da gramática estão representadas no seguinte grafo construido pelo Passo 1.



PICHA	MARCADO	SYMB*							
		Ω	S	A	C	E	T	P	B
(S,Ω)	S		S						
(S,Ω) (C,S)	C		C		C				
(S,Ω) (A,S)	A		A	A					
(S,Ω)		S							
		C							
		A							
(E,Ω)	E					E			
(E,Ω) (T,E)	T						T		
(E,Ω) (T,E) (P,T)	P							P	
(E,Ω) (T,E)						T	P		
						P			
(E,Ω)		E							
		T							
		P							
(B,Ω)	B	B							B

SYMB*	S	A	C	E	T	P	B
S	1	1	1				
A		1					
C			1				
E				1	1	1	
T					1	1	
P						1	
B							1

O funcionamento do algorítmo pode ser verificado se atentarmos para os passos 4 e 5, que efetuam o fechamento transitivo reflexivo da relação SYMB. Nesse instante o Passo 5 verifica se o nó corrente já não foi alcançado por um de seus antecessares empilhados. Como pelo menos uma aresta sera distinta haverá dois caminhos distintos entre o par de nós.

O algorítmo percorre cada nó tantas vezes quantas forem as arestas incidentes ao nó – o conjunto MARCADO assegura que o conjunto de sucessores será calculado uma Única vez. A complexidade do algorítmo será, portanto, proporcional ao número de arestas. Como temos menos arestas em GSIMPLES que produções teremos um algorítmo O(p).

Apresentamos a seguir o algorítmo construtor do analisador sintático que, receberá como entrada uma gramática de operadores reduzida e fornecerá como saída as funções ESQUERDO, GOTO, AÇÃO e AVANÇAREDUZ ou a indicação de que a gramática não é GMT. Evidentemente usaremos os algorítmos já demonstrados como passos do construtor.

O algorítmo a seguir é o ponto central deste trabalho,

Algorítmo 6 (LXV)

CONSTRUTOR DE ANALISADORES SINTÁTICOS GMT, OUTRA VEZ

ENTRADA : $G = (N, \Sigma, P, S) \in \mathcal{C}_{GO}$

SAfDA : se $G \in \mathcal{C}_{GMT}$: tabelas ESQUERDO, GOTO, AÇÃO, AVANÇAREDUZ

senão: mensagem "$G \notin \mathcal{C}_{GMT}$".

Passo 1: Construa G' = (N', C', P', S'), usando o Alg. 2.
Sejam p = #(P), p' = #(P')-1 e k conforme o teorema 7 (XLV).

Passo 2: Execute o Alg. 5 e prossiga se houver unicidade nas produções simples; caso contrário escreva a mensagem "$G \notin \mathcal{C}_{GMT}$" e pare;

Passo 3: Varrendo as produções da gramática original construa, por inspeção, as relações LASTNT, NT.TERM, FIRSTNT e FIRSTTERM e a função ESQUERDO;

Passo 4: Varrendo as produções da gramática estendida:
- para $0 \leqslant j \leqslant p$ construa por inspeção a relação ESTR.NT e as funções NTERM e ESTR>;
- para $p+1 \leqslant j \leqslant k$ construa por inspeção as funções TERM< e NTERM;
- para $k+1 \leqslant j \leqslant p'$ construa por inspeção a relação ESTR.NT e as funções ESTR= , TERM= e NTERM;

(Note que chamamos *"função"* as relações que satisfazem a propriedade (XLVII) de possuir um Único elemento por linha).

Passo 5: (Constroi GOTO)
Chamemos $n = \#(N)$. Calcule ESTR.NT FIRSTNT* $\subset$ (NxN)
Construa GOTO numerando os pontos onde essa relação é verdadeira, a partir de $n + 1$.
Assuma que todo $U \in N$ está numerado entre 1 e n e faça GOTO(U,ε) igual ao número de U;

Passo 6: (Calcula relações básicas)

Calcule:

$$MEIO = NTERM\ SYMB^*$$

$$\Rightarrow = ESQ> (LASTNT^*)^{-1}\ NT.TERM$$

$$\dot{\lessdot} = TERM< (ESTR.NT\ \ FIRSTNT^*\ \ FIRSTTERM)^{-1}$$

Passo 7: (Constroi AÇÃO E AVANÇAREDUZ)

Neste passo, à cada atribuição de valor à função AÇÃO se esta já tiver valor atribuído indique $G \notin \mathcal{G}_{GMT}$ e pare.

Varrendo as produções da gramática estendida:

– para $0 \leqslant j \leqslant p$:

Sejam: ESTR = {U | j ESTR> U} = {U} (Único)

NT = {A | j MEIO A)

T = {a | j $\Rightarrow$ a)

Para cada par (A,a) E (NT x T) faça:

AÇÃO (GOTO (U,A),a) = $\Rightarrow$

AVANÇAREDUZ (GOTO (U,A),a) = j

– para $p+1 \leqslant j \leqslant k$:

Sejam: ESTR = {U | j $\dot{\lessdot}$ U}

NT = {A | j MEIO A)

T = {a | j TERM< a) = {a} (único)

REDUZ = {V | j ESQ< V} = {V} (único)

Para cada par (U, A) $\in$ (ESTR x NT) faça:

AÇÃO (GOTO (U,A), a) = $\lessdot$

AVANÇAREDUZ (GOTO (U,A), a) = GOTO (V, E)

– para j = k+1 :

Sejam: ESTR = {U | j ESTR= U} = {#} (Único)

NT = {A | j MEIO A}

T = {a | j TERM= a) = {#} (Único)

Para cada A ∈ NT faça:

AÇÃO (GOTO (#, A), #) = 'pare'

– para k+1 < j p':

Sejam: ESTR = {U | j ESTR= U} = {U} (Único)

NT = {A | j MEIO A}

T = {a | j TERM= a) = {a} (Único)

REDUZ = {V | j ESQ= V} = {V} (Único)

Para cada A ∈ NT faça:

AÇÃO (GOTO (U, A) , a) = $\stackrel{*}{=}$

AVANÇAREDUZ (GOTO (U, A) , a) = GOTO (V, E) .

A distinção fundamental entre este construtor e o proposto por |*Gries, 68*| é que naquele testamos as condições para cada tripla (U, A, a) enquanto que neste conhecemos o caminho para chegar apenas às triplas onde as condições serão verdadeiras.

Isto nos leva a uma conclusão importante:

a) se o algorítmo visita todos os pontos da tabela que represen

tam configurações válidas do analisador e cada um destes pontos uma Única vez;

b) se a complexidade de tempo dos demais passos do algorítmo é inferior a complexidade de tempo do passo 7, que efetua tal visita;

c) se qualquer outro algorítmo construtor necessita visitar pelo menos uma vez cada ponto da tabela que represente uma configuração válida do analisador;

d) então o algorítmo aqui proposto é tão eficiente, em complexidade de tempo, quanto o mais eficiente que se possa construir;

Verificando a condição b, acima:

a) o Passo 1 é $O(n)$, onde n é a doma dos comprimentos dos lados direitos das produções de P;

b) o Passo 2 é $O(p)$, onde p é o número de produções de P;

c) o Passo 3 é, também, $O(p)$;

d) o Passo 4 é $O(p')$, onde $p' = \#(P')$;

e) o Passo 5 é $O(n^2)$, pois as relações ESTR.NT e FIRSTNT são "*esparsas*", no sentido que o número de arestas dos grafos que as representam são menores, respectivamente, que a soma dos comprimentos dos lados direitos e o número de produções. |*Hunt*, 77| demonstra que o produto e o fechamento de relações esparsas é $O(n^2)$.

f) o Passo 6 é $O(n^2)$ pelos mesmos motivos, pois todas as relações envolvidas são esparsas;

Logo, o restante do algorítmo construtor, com exceção

do crucial passo 7, é $O(n^2)$.

No pior caso, que ocorrerá para gramáticas que preencham todos os pontos da tabela, o passo 7 será $O(n^3)$. Tais gramáticas são suficientemente patológicas para terem apenas interesse teórico. As gramáticas de linguagem de programação usuais, em particular gramáticas de operadores que são *"pontuadas"* por terminais, levam a tabelas cuja densidade de configurações válidas – e, portanto, a complexidade do passo 7 – é $O(n^2)$.

Esta discussão demonstra, informalmente, que:

a) o construtor proposto é, no pior caso, mais eficiente em complexidade de tempo em uma ordem de grandeza que o proposto por |*Gries, 68*|; e, para gramáticas usuais, mais eficiente em duas ordens de grandeza;

b) o construtor proposto é tão eficiente quanto o melhor que se possa criar;

A tabela gerada pelo construtor é, precisamente, e exemplificada em (LXII).

Resta-nos demonstrar que o Alg. 6 reconhece precisamente a classe das GMT, tal como redefinida em (LII).

■ Teorema 12 (LXVI)

O Alg. 6 reconhece a classe das GMT.

Demonstração: Imediata. As condições necessárias e suficientes

para que $G \in \mathcal{C}_{GMT}$ são verificadas da seguinte forma:

a) a unicidade das derivações em produções simples pelo Passo 2;

b) as condições (i)a , (i).b e (i).c, pelo Passo 7;

c) a condição de que no máximo uma das proposições seja verdadeira é garantida pelo algorítmo quando exige que AJÃO seja uma função;

d) o mesmo para a unicidade do não-terminal da configuração.

———— ■

4.4 – ARMAZENAMENTO DA TABELA DE CONTRÔLE

Nesta secção iremos desenvolver técnicas de armazenamento das funções AÇÃO, AVANÇAREDUZ, ESQUERDO e GOTO, de modo a tornar a análise sintática em GMT um método atualizado e eficiente em economia de espaço. Pretendemos fazer uso de sua principal vantagem, no caso: na medida em que a quantidade de informações para decisão da ação do analisador sintático é menor, é natural que o método deva possuir formas de armazenar as tabelas de controle em menor espaço. Esta secção pretende obtê-las. Note que não nos interessaremos por soluções gerais de armazenamento – como armazenar matrizes esparsas em forma de listas, por exemplo – mas, apenas, por desenvolver soluções que são particulares e características deste método.

Definição

MÁQUINA CARACTERÍSTICA (LXVII)

Um transdutor finito determinístico (TFD) |*Aho & Ullman, 72*| M = (Q, IT, A, 6, q,, F) onde

a) Q é um conjunto finito de estados

b) π é um alfabeto finito de entrada

c) A é um alfabeto finito de saída

d) δ é uma função de transição entre

(Qx(π ∪ {E})) e os subconjuntos finitos de Q x A*

e) $q_0 \in Q$ é o estado inicial

f) $F \subseteq Q$ é o conjunto de estados finais

será dito *máquina característica* de um analisador sintático para GMT com estado inicial E_0 e funções AÇÃO, GOTO, AVANÇAREDUZ e ESQUERDO se satisfizer as seguintes condições:

(i) $Q = E \cup \{q_f\}$, com $q_f \notin E$

(ii) $\pi = N \cup C$

(iii) $A = \{0,1,\ldots,p\}$

(iv) $q_0 = E_0$

(v) $F = \{q_f\}$

(vi) 6 satisfaz a:

- $\delta(q,\varepsilon) = \phi$
- se $q, q_1 \neq q_f$ e $a \in \Sigma$

$$\delta(q,a) = q_1/\varepsilon \iff \text{AÇÃO } (q,a) = \triangleleft \text{ ou } \triangleq$$

$$\text{e AVANÇAREDUZ } (q,a) = q_1$$

- se $q, q_1 \neq q_f$ e $A \in N$

$$\delta(q,A) = q_1/\varepsilon \iff \text{GOTO } (q,A) = q_1$$

– se $q \neq q_f$

$\delta(q,a) = q_f/j$ ⟺ AÇÃO $(q,a) = \triangleright$

e AVANÇAREDUZ $(q,a) = j$

$\delta(q,a) = q_f/0$ ⟺ AÇÃO (q,a) = 'pare'

■

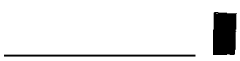
Notemos, mais informalmente, que acaba de ser definido um transdutor finito – determinístico – que engloba as funções AÇÃO, AVANÇAREDUZ e GOTO. A função GOTO é simplesmente transposta. É criado um estado adicional q_f que é o Único estado final. A função AVANÇAREDUZ quando indica uma transição de estado – AÇÃO = $\lessdot$ ou $\overset{*}{=}$ – é simplesmente transposta e quando indica uma redução – AÇÃO = $\triangleright$ – é criada uma transição para o estado final adicionado e a máquina emite o número da produção. Esta máquina nos será Útil para demonstrar propriedades das tabelas de controle pois eliminamos a pilha do "*pushdown transducer*".

■ Exemplo (continuação do exemplo LXIV) (LXVIII)

MAQUINA CARACTERÍSTICA

EST	#	id	:=	if	then	else	+	*	(	)	or	S	A	C	E	T	P	B
1		2		3								14	15	16				
2	R/11		9		R/13	R/11	R/11	R/11		R/11	R/13							
3		2																17
4		2							4						18	19	20	
5		2							4							21	22	
6		2							4								23	
7		13																
8																		
9		2							4						24	25	26	
10		2											27					
11		2		3								28	29	30				
12	R/10					R/10	R/10	R/10		R/10								
13					R/12						R/12							
14	R/0																	
15	R/0																	
16	R/0																	
17					10						7							
18							5			12								
19							5	6		12								
20							5	6		12								
21	R/7					R/7	R/7	6		R/7								
22	R/7					R/7	R/7	6		R/7								
23	R/9					R/9	R/9	R/9		R/9								
24	R/3					R/3	5											
25	R/3					R/3	5	6										
26	R/3					R/3	5	6										
27	R/4					11												
28	R/5																	
29	R/5																	
30	R/5																	
R																		

R é o estado final e não tem transições por construção.



Antes de iniciarmos a minimização vamos determinar quais os pontos inacessíveis da máquina característica. Seguimos, aqui, o caminho proposto por |*Anderson, 73*|.

Teorema 13 (LXIX)

$\delta(q,A)$ = não definido, A E N

$\wedge(\forall a \in \Sigma \mid (A \rightarrow a\alpha) \in P)$ ⇒ AÇÃO $[q,a] \neq \ll$)

⇔ $\delta(q,A)$ é inacessível.

Demonstração

A declaração de um ponto como inacessível significa que o analisador não atingirá uma certa configuração tanto para as sentenças corretas como *para uma sentença qualquer do conjunto* C*.

Evidentemente, como $\delta(q,A)$ = não definido, não existe qualquer sentença em L(G) que leve o analisador a uma configuração tal que o estado no topo da pilha seja q e a variável MEIO seja A. Concentremo-nos nas sentenças $\#y\# \notin L(G)$.

– parte se

Seja a redução de $\#y\# \notin L(G)$

$$\beta A x \Longleftarrow \beta a \alpha x \overset{*}{\Longleftarrow} \#y\#$$

Seja $q = (\beta,\varepsilon)$. Jamais poderemos ter uma configuração (q,A), pois aa não pode ser admitido sem alterar o estado q pois AÇÃO $[q,a] \neq \ll$

– parte somente se

Seja $\delta(q,A)$ inacessível, $q = (\beta,\varepsilon)$ e $\#y\# \notin L(G)$. Supo

nhamos por absurdo que AÇÃO [q,a] = <* e (A → aa) ∈ P.

Logo:

$\underline{\beta}\ \underline{a}\ x \Longrightarrow \underline{\beta}\ a\ x$

Sem perda de generalidade, podemos supor que x seja uma continuação tal que implique na redução $A \Longleftarrow aa$

Logo:

$\underline{\beta}\ A\ z \Longleftarrow \underline{\beta}\ \underline{a\alpha}\ z \overset{*}{\Longleftarrow} \underline{\beta}\ \underline{a}\ x \overset{*}{\Longleftarrow} \#y\#$

c.q.d.

_____ ■

O método simples de declarar inacessíveis os pontos do quadrante (q,A) – que corresponde à função GOTO – é marcar todos os pontos correspondentes a um certo estado q, *exceto* aqueles onde $\delta(q,a) = <^*$ e $(A \to aa) \in P$.

■ Teorema 14 (LXX)

$\delta(\delta(q,A),a)$ = não definido, $A \in N$, $a \in C$

$\wedge\ \neg$ (A FOLLOW a)

⟺ $\delta(\delta(q,A),a)$ é inacessível.

Demonstração

Temos que A FOLLOW a ⟺ $S \overset{*}{\underset{G}{\Longrightarrow}} a\,A\,a\beta$.

Seja $q_1 = \delta(q,A)$. A demonstração é muito simples e suponhamos que $\#y\# \notin L(G)$.

– parte se: admitamos a hipótese e por absurdo seja

$\delta(q_1,a)$ acessível. Logo:

$$\underline{\beta}\, A\, a\, x \Longleftarrow \#y\# \quad , \quad \text{com } q = (\underline{\beta},\varepsilon)$$

Necessariamente para se obter A tivemos de reduzí-lo com

$$(j : A \rightarrow \alpha) \in P$$

Portanto, existe alguma sentença $z \in L(G)$ tal que

$$S' \overset{*}{\Longrightarrow} \underline{\gamma}\, A\, a\, \omega \overset{*}{\Longrightarrow} \#z\#$$

Logo: A FOLLOW a, o que contraria a hipótese, c.q.d.

– parte somente se: se $\delta(q_1,a)$ é inacessível então não existe $\#y\#$ tal que

$$\underline{\beta}\, A\, a\, x \Longleftarrow \#y\#$$

Logo $S' \overset{*}{\not\Longrightarrow}_{G} \alpha\, A\, a\beta$ e, portanto $\neg$ (A FOLLOW a)

———————— ■

O Teorema 14 permite criar um método simples de obter os pontos inacessíveis no 'quadrante 6 (6 (q, A), a), pois A FOLLOW a $\Leftrightarrow$ A $(\text{LASTNT}^*)^{-1}$ NT.TERM a, sendo subproduto do cálculo da relação $\Rightarrow$

No quadrante 6(6(q,E),a) claramente todos os pontos são acessíveis, assim como em δ(6(q,A),A) todos os pontos são inacessíveis.

■ Exemplo (continuação do exemplo LXVIII) (LXXI)

Neste exemplo os pontos inacessíveis já estão transpostos para a máquina característica GMT e são indicados por a.

$A \rightarrow a\alpha$

	#	id	:=	if	then	else	+	*	(	)	or
S											
A		X									
C				X							
E											
T											
P		X							X		
B		X									

A FOLLOW a

	#	id	:=	if	then	else	+	*	(	)	or
S	X										
A	X					X					
C	X										
E	X					X	X			X	
T	X					X	X	X		X	
P	X					X	X	X		X	
-					X						X

MAQUINA CARACTERÍSTICA

EST	#	id	:=	if	then	else	+	*	(	)	or	S	A	C	E	T	P	B
1		2		3								14	15	16	ϕ	ϕ	•	•
2	R/11		9		R/13	R/11	R/11	R/11		R/11	R/13	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
3		2										ϕ	•	ϕ	ϕ	ϕ	•	17
4		2							4			ϕ	•	ϕ	18	19	20	•
5		2							4			ϕ	•	ϕ	ϕ	21	22	•
6		2							4			ϕ	•	ϕ	ϕ	ϕ	23	•
7		13										ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
8	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
9		2							4			ϕ	•	ϕ	24	25	26	•
10		2										ϕ	27	ϕ	ϕ	ϕ	•	•
11		2		3								28	29	30	ϕ	ϕ	•	•
12	R/10					R/10	R/10	R/10		R/10		ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
13					R/12						R/12	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
14	R/0	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ							
15	R/0	ϕ	ϕ	ϕ	ϕ	•	ϕ	ϕ	ϕ	ϕ	ϕ							
16	R/0	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ							
17	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	7							
18	•	ϕ	ϕ	ϕ	ϕ	•	5	ϕ	ϕ	12	ϕ							
19	•	ϕ	ϕ	ϕ	ϕ	•	5	6	ϕ	12	ϕ							
20	•	ϕ	ϕ	ϕ	ϕ	•	5	6	ϕ	12	ϕ							
21	R/7	ϕ	ϕ	ϕ	ϕ	R/7	R/7	6	ϕ	R/7	ϕ							
22	R/7	ϕ	ϕ	ϕ	ϕ	R/7	R/7	6	ϕ	R/7	ϕ							
23	R/9	ϕ	ϕ	ϕ	ϕ	R/9	R/9	R/9	ϕ	R/9	ϕ							
24	R/3	ϕ	ϕ	ϕ	ϕ	R/3	5	ϕ	ϕ	•	ϕ							
25	R/3	ϕ	ϕ	ϕ	ϕ	R/3	5	6	ϕ	•	ϕ							
26	R/3	ϕ	ϕ	ϕ	ϕ	R/3	5	6	ϕ	•	ϕ							
27	R/4	ϕ	ϕ	ϕ	ϕ	11	ϕ	ϕ	ϕ	ϕ	ϕ							
28	R/5	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ							
29	R/5	ϕ	ϕ	ϕ	ϕ	•	ϕ	ϕ	ϕ	ϕ	ϕ							
30	R/5	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ							
R																		

O estado 8 é inacessível. O estado **R** não faz parte das tabelas.

Note a importância da determinação dos pontos inacessíveis. Esta máquina pode ser submetida aos algorítmos usuais de obtenção de TFD mínimo através de classes de equivalência |*Hopcroft & Ullman, 79*|. A máquina resultante chamaremos máquina característica mínima.

Para efeito de compactação das tabelas, porém, vamos preferir, como |*Anderson, 73*|, usar um algorítmo menos complexo. De início faremos a fusão dos estados $q = \delta(q,A)$ com $A \neq E$. As alterações correspondentes serão feitas na numeração dos estados. A regra para fusão de estados é simples:

a) dois estados q_1 e q_2 serão unificados em um só desde que:

(i) $\delta(q_1,a) = \delta(q_2,a) = q/\varepsilon$ ou $\delta(q_1,a) = \Phi$ ou $\delta(q_2,a) = \Phi$, para todo $a \in \pi$

(ii) $\delta(q_1,a) = \delta(q_2,a) = R/n$

Em seguida faremos a fusão de colunas no quadrante $\delta(q,A)$ – função GOTO – anotando as alterações para a função ESQUERDO. À máquina característica resultante chamaremos pseudo-mínima.

■ Exemplo (continuação do exemplo LXXI) (LXXII)

MÁQUINA CARACTERÍSTICA PSEUDO-MÍNIMA														
ESTADO	#	id	:=	if	then	else	+	*	(	)	or	1	2	3
1		2		3								13	•	•
2	R/11		8		R/13	R/11	R/11	R/11		R/11	R/13	ϕ	ϕ	ϕ
3		2										•	•	13
4		2							4			•	14	•
5		2							4			•	15	•
6		2							4			•	16	•
7		12										ϕ	ϕ	ϕ
8		2							4			•	17	•
9		2										18	•	•
10		2		3		R/10	R/10	R/10				19	•	•
11	R/10											ϕ	ϕ	ϕ
12					R/12						R/12	ϕ	ϕ	ϕ
13	R/0	ϕ	ϕ	ϕ	9	•	ϕ	ϕ	ϕ	ϕ	7			
14	•	ϕ	ϕ	ϕ	ϕ	•	5	6	ϕ	11	ϕ			
15	R/7	ϕ	ϕ	ϕ	ϕ	R/7	R/7	6	ϕ	R/7	ϕ			
16	R/9	ϕ	ϕ	ϕ	ϕ	R/9	R/9	R/9	ϕ	R/9	ϕ		Φ	
17	R/3	ϕ	ϕ	ϕ	ϕ	R/3	5	6	ϕ	•	ϕ			
18	R/4	ϕ	ϕ	ϕ	ϕ	10	ϕ	ϕ	ϕ	ϕ	ϕ			
19	R/5	ϕ	ϕ	ϕ	ϕ	•	ϕ	ϕ	ϕ	ϕ	ϕ			

ESQUERDO
1
1
1
1
1
2
2
2
2
2
2
3
3

O Único quadrante ainda não minimizado é o correspondente a $\delta(\delta(q,\varepsilon),a)$. Mas para isto é necessário que exploremos algumas propriedades das tabelas e não mais da máquina característica.

Teorema 15 (LXXIII)

i) o valor de AVANÇAREDUZ quando AÇÃO = ⇒ é constante no estado, independentemente do terminal, exceto nos estados que advém de duas produções com lados direitos iguais;

ii) o valor de AVANÇAREDUZ quando AÇÃO = ⇐ é constante para um ter

minal, em todos os estados com mesmo não-terminal (em particular o primeiro quadrante (LU, E)).

Demonstração: Imediata, verificando-se o funcionamento do construtor.

■

Este teorema nos permite montar uma estratégia de armazenamento especial para a função AVANÇAREDUZ, *permitindo que apenas a função AÇÃO fique armazenada como matriz* (note que um valor da função AÇÃO ocupa apenas 2 bits).

Iremos subdividir a função AVANÇAREDUZ em três funções: AVANÇA, REDUZ e CONCENTRA, conforme o correspondente valor de AÇÃO seja $\lessdot$, $\doteq$ ou $\gtrdot$. Ou seja, usaremos a função AÇÃO para particionar a função AVANÇAREDUZ, da seguinte forma:

i) AVANÇA : (E x C) $\longrightarrow$ {0,1,...,n}

AVANÇA [Ê,a] = j $\Leftrightarrow$ AÇÃO [Ê,a] $\lessdot$

AVANÇAREDUZ [Ê,a] = j

ii) REDUZ : (E x C) $\longrightarrow$ {0,1,...,p}

REDUZ [Ê,a] = j $\Leftrightarrow$ AÇÃO a $\gtrdot$

AVANÇAREDUZ [Ê,a] = j

iii) CONCENTRA : (E x C) $\longrightarrow$ {0,1,...,n}

CONCENTRA [Ê,a] = j AÇÃO [Ê,a] $\doteq$

AVANÇAREDUZ [Ê,a] = j

A estratégia de armazenamento dessas funções será:

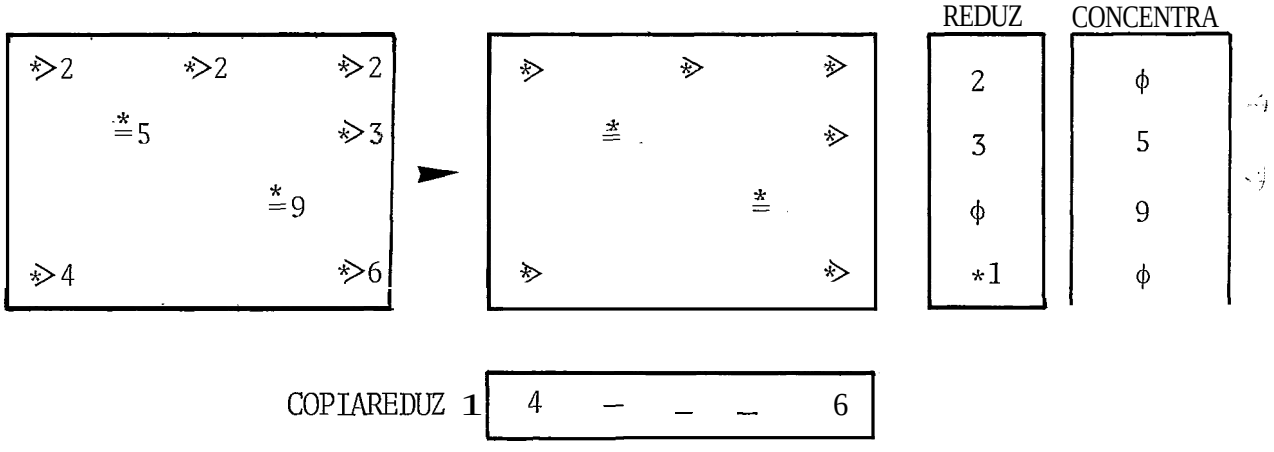
i) para REDUZ e CONCENTRA tentaremos acomodá-las em uma coluna, isto é, partiremos da hipótese que são constantes em certo estado. Pelo teorema 15 isto é bastante provável para REDUZ e apenas não será verdadeiro para CONCENTRA em gramáticas com produções

$$A \to aab...$$

$$B \to aac...$$

Porém, quando tal não for possível marcaremos esse estado e o valor da função indicará uma cópia da linha.

Por exemplo:



ii) para a função AVANÇA tentaremos acomodá-la em uma linha, o que é provável dado o teorema 15. Se isso não for possível usaremos um esquema semelhante, repetindo porém a coluna.

Por exemplo

1 COPIAAVANÇA

◁*3	◁*5	◁*4	4
◁*3			ϕ
◁*3	◁*5		ϕ
◁*3		◁*6	6

AVANÇA

3	5	*1

Garantidamente pelo teorema 15 o quadrante (L, E) não provoca colisões.

Feito isso nada nos impede colapsar a tabela AÇÃO, criando um nível de indireção através da coluna T-AÇÃO. Com isto chegamos ao nível final da compactação.

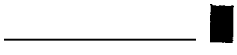
Exemplo (continuação do exemplo LXXII) (LXXIV)

	AÇÃO											GOTO		
	#	id	:=	if	then	else	+	*	(	)	or	1	2	3
1	•	◁	•	◁	•	•	•	•	•	•	•	13	•	•
2	▷	•	═	•	▷	▷	▷	▷	•	▷	▷	ϕ	ϕ	ϕ
3	•	◁	•	•	•	•	•	•	•	•	•	•	•	13
4	•	◁	•	•	•	•	•	•	◁	•	•	•	14	•
5	•	◁	•	•	•	•	•	•	•	•	•	•	15	•
6	▷	•	•	•	•	▷	▷	▷	•	•	•	•	16	•
7	•	•	•	•	▷	•	•	•	•	•	▷	ϕ	ϕ	ϕ
8	▷	ϕ	ϕ	ϕ	═	•	ϕ	ϕ	ϕ	ϕ	◁	•	17	•
9	•	ϕ	ϕ	ϕ	ϕ	•	◁	◁	ϕ	═	ϕ	18	•	•
10	▷	ϕ	ϕ	ϕ	ϕ	▷	▷	◁	ϕ	▷	ϕ	19	•	•
11	▷	ϕ	ϕ	ϕ	ϕ	▷	◁	◁	ϕ	•	ϕ	ϕ	ϕ	ϕ
12	▷	ϕ	ϕ	ϕ	ϕ	═	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ

ϕ	2	ϕ	3	ϕ	ϕ	5	6	4	ϕ	7	AVANÇA

11	ϕ	ϕ	ϕ	13	**11**	**11**	**11**	ϕ	11	13	COPIA REDUZ(1)

	T - AÇÃO	REDUZ	CONCENTRA	ESQUERDO
1	1	ϕ	ϕ	1
2	2	*1	8	1
3	3	ϕ	ϕ	1
4	4	ϕ	ϕ	1
5	4	ϕ	ϕ	1
6	4	ϕ	ϕ	2
7	5	ϕ	12	2
8	4	ϕ	ϕ	2
9	3	ϕ	ϕ	2
10	10	ϕ	ϕ	2
11	6	10	ϕ	2
12	7	12	ϕ	3
13	8	0	9	3
14	9	ϕ	11	
15	10	7	ϕ	
16	2	9	ϕ	
17	11	3	ϕ	
18	12	4	10	
	8	5		



CAPÍTULO V

- RECUPERAÇÃO DOS ERROS -

Os analisadores sintáticos para gramáticas da classe matrizes de transição – como já citamos em capítulos anteriores – apresentam alguma dificuldade na recuperação de erros. Este capítulo final abordará este tema e pretende-se assim que a análise sintática em GMT seja abordada em todos os seus aspectos fundamentais: definição da classe de gramáticas, definição do método de análise, definição do algorítmo construtor do analisador sintático, definição dos métodos de compactação das tabelas e definição de estratégias de recuperação de erros.

5.1 - FUNDAMENTOS DE RECUPERAÇÃO DE ERROS

Boa parte do tempo dispendido em programação – e para programadores indisciplinados a maior parte desse tempo – é gasta na correção de erros do programa. Tradicionalmente costuma-se classificar estes erros em três categorias, de acordo com a origem Última do erro:

a) os erros de transcrição, causados por uma falha na comunicação à máquina do programa e/ou dados do usuário;

b) os erros de programação, causados por uma falha na tradução da solução especificada para a linguagem desejada;

c) os erros de especificação, causados por uma falha na especificação da solução geral do problema e na sua adaptação ao am-

biente computacional;

Evidentemente não estamos considerando o caso em que a solução proposta não satisfaz o problema.

Entretanto é virtualmente impossível detectar-se um erro e classificá-lo segundo tais categorias. Os erros tendem a ser apresentados ao usuário conforme a fase de execução onde tenham sido percebidos:

a) erros léxicos, quando ocorrem em *"estruturas básicas"* da linguagem, na verdade quando não satisfazem o filtro léxico que é o *"scanner"*;

b) erros sintáticos, quando em desacordo com a gramática da linguagem, na verdade quando não satisfazem o analisador sintático;

c) erros de sintaxe sensível ao contexto, chamados comumente de *"erros semânticos"*, quando não satisfazem os demais filtros de análise sintática sensível ao contexto;

d) erros semânticos, em geral chamados *"erros de execução"* ou *"erros de lógica"*, percebidos durante a execução do programa para certo conjunto de dados e que alertam uma incongruência entre as propriedades previstas e obtidas de alguma entidade do programa;

Em particular a concepção absoluta de programa *"correto"* não é definível desta maneira uma vez que a *"correção"* ou *"incorreção"* de um programa depende da maior ou menor quantidade de filtros disponíveis no compilador ou agregados pelo compilador ao programa objeto. Note também que não há relação de causa e efeito entre a primeira e a segunda classificação de erros:

um simples erro de transcrição ("*mispunching*", por exemplo) poderá ser detectado apenas quando da execução. Além disso, a definição de semântica de uma linguagem de programação é, em geral, feita de modo informal (embora o problema de formalização dessa definição esteja sendo objeto de intensa pesquisa). Por isso, muitas vezes, a Única definição precisa da linguagem aceita por um compilador é o próprio compilador, prática esta que apresenta as desvantagens conhecidas.

Costuma-se adotar como objetivo de um compilador a detecção do maior número de erros possível em uma Única análise do programa. Em particular espera-se que todos os erros que assim o admitam sejam detectados durante uma Única compilação, mais ainda, que as mensagens de erro permitam ao programador corrigi-los. Frequentemente este objetivo é abandonado frente a considerações sobre a eficiência do compilador.

Esta discussão inicial permite mostrar as vantagens de se acoplar diretamente ao gerador de analisadores sintáticos um gerador de recuperador de erros, o mais simples e o mais automatizado possível. Em resumo: fornecer ao usuário o máximo possível de informações sobre erros detectáveis durante a compilação é um dos maiores serviços que um compilador pode prestar. Entretanto, nos compiladores que normalmente usamos, este serviço é mínimo ou obtido a custa de enorme esforço dos projetistas do compilador em programar um infindável número de procedimentos "*ad-hoc*". Consideramos que o recuperador de erros sintáticos deva ser parte integrante do projeto do analisador sintático e dessa forma atacaremos o problema.

Algumas restrições devem ser feitas de imediato:

a) entender-se-á "*recuperar um erro*" com o sentido de "*recuperar-*

-se de um erro" ou seja: retornar o analisador à uma configuração tal que lhe permita continuar a análise detectando o maior número possível dos erros subsequentes;

b) iremos nos ater à recuperação de erros sintáticos, ou seja detectados pelo próprio analisador, e teremos como base de informação a gramática livre de contexto que lhe deu origem. Isto não significa menosprezar a importância de outras fontes de informação, como a tabela de símbolos, por exemplo. Apenas sua inclusão estará fora do escopo deste trabalho.

5.2 – REVISÃO BIBLIOGRÁFICA

O problema de recuperação de erros sintáticos em analisadores de gramáticas livres de contexto apresenta uma bibliografia extensa demais para que possamos abrangê-la sem seleção. |*Ciesinger, 79*|, fornece uma lista de 90 artigos sobre o assunto e tentaremos classificá-los segundo nossa visão particular do problema.

Inicialmente, vejamos uma certa categoria de artigos que fornecem dados estatísticos sobre a frequência de erros encontrados em certas linguagens de programação. Paradoxalmente o número de artigos a esse respeito é extremamente reduzido: |*Ripley & Druseikis, 78*|, |*Litecky & Davis, 76*|, |*Youngs, 72*|.

Algumas das conclusões são particularmente importantes para o direcionamento da pesquisa, retiradas de |*Ripley & Druseikis, 78*| e referentes a Pascal:

a) cêrca de 60% dos programas submetidos não apresentaram erros detectáveis durante a compilação;

b) cerca de 85% dos êrros poderiam ser corrigidos inserindo-se (50%), removendo-se (14%) ou trocando-se (21%) um Único elemento sintático:

c) cerca de 80% das "*frases*" (no caso, declarações ou comandos) continham um Único erro;

d) apenas cerca de 8% dos erros foram devidos a soletração ("*misspelling*" ou "*mispunching*");

e) cerca de 20% dos erros foram a falta do ";" final da frase;

Isto nos permite tirar algumas conclusões bastante práticas sobre o nosso problema:

a) a distribuição dos erros parece estar concentrada fortemente em certas construções da linguagem, o que levanta dúvidas sobre estratégias que partem da hipótese de distribuição aleatória de erros;

b) a distribuição dos erros pelo programa parece ser bastante "*esparsa*" – no sentido de não haver tendência a concentração de erros em um ponto do programa – o que sugere fortemente o uso do contexto em torno do erro como informação extremamente válida; e, também, que não faz muito sentido sofisticar a estratégia para detecção de erros muito próximos (mesmo porque o trecho já terá de ser corrigido);

c) o repertório de erros com alguma probabilidade de ocorrência é extremamente pequeno e é preferível deixar os improváveis para serem tratados por alguma espécia de "*panic-mode*" (que funcionou adequadamente em 85% dos casos);

Voltemos, então, a análise da bibliografia. Um certo

grupo de artigos, especialmente os livros texto sobre compiladores, propõem a elaboração de procedimentos "ad-hoc" |Aho & Ullman, 77|, |Gries,71|, |Lewis, Rosenkrantz & Stearns,73|. Os artigos mais antigos também seguem esta linha |Conway & Wilcox, 73| e, em particular, |Gries, 68| para GMT. A essas propostas se dirige com precisão a crítica de |Rhodes, 73|: "o uso de um esquema automático de recuperação de erro não apenas é de implementação mais simples como, em geral, fornece também uma recuperação de melhor qualidade".

As principais idéias em recuperação automática de erros tem origem nos trabalhos de |Leinius, 70|, |Peterson, 72| |La France, 70| e |Rhodes, 73| e são, respectivamente, as seguintes:

a) uma generalização do "*panic-mode*", permitindo que uma sequência derivada de um não-terminal qualquer (e não apenas ao nível de "*frase*") possa ser descartada |Leinius, 70|;

b) uma medida da distância (em termos de trocas, inserções ou remoções) que separa uma certa sequência da sua provável definição gramatical |Peterson, 72|;

c) o uso do contexto à direita do ponto de erro, usando o próprio analisador sintático como gerador de continuações possíveis |La France, 70|;

d) o uso do contexto à direita e à esquerda do ponto de erro, usando o próprio analisador sintático como verificador da continuação |Rhodes, 73|;

Alguns conceitos foram isolados desde então e passaram a fazer parte do vocabulário usual de correção de erros:

i) "*backward move*" – um passo atrás – significando efetuar as reduções possíveis quando do encontro do erro, concentrando contexto à esquerda;

ii) "*forward move*" – um passo adiante – significando continuar a análise sintática sob supervisão do recuperador de erros, concentrando contexto à direita;

iii) "*correção pré-programada*", significando a tomada de decisão quanto a recuperação do erro apenas com as informações que o analisador sintático dispõe para a análise normal;

iv) "*custo mínimo*", significando a soma dos pesos de inserção e remoção de cada símbolo envolvido na transformação de uma sequência de símbolos em outra;

v) "*propriedade dos prefixos corretos*", significando a capacidade que certos métodos de análise sintática dispõem de detectar qualquer erro sem que se tenha lido nenhum símbolo a mais que o necessário;

vi) "*propriedade de detecção de erro imediata*", significando a capacidade que certos métodos de análise sintática dispõem de detectar qualquer erro sem que nenhuma ação seja executada pelo analisador sintático após a detecção;

O desenvolvimento posterior do estudo em detecção de erros concentrou-se bastante na sofisticação destas vertentes que apresentamos: |*Penello & De Remer, 78*| e |*Graham & Joy, 79*| na linha de |*Rhodes, 73*| ; |*Boullier, 78*| na linha de |*Lu France, 70*| ; |*Fischer, Dion & Mauney, 79*| , |*Fischer, Mauney & Milton, 79*| e diversos outros na linha de |*Peterson, 72*| ; |*Pai, 79*| e outros na linha de |*Leinius, 70*|.

Por outro lado algumas novas idéias foram desenvolvidas:

a) a proposta de se adicionar estados ao analisador capazes de suportar a continuação da análise mesmo para sentenças erradas |*Ripley & Druseikis, 76*|;

b) métodos baseados apenas – ou principalmente – em inserção de símbolos |*Fischer, Milton & Quiring, 79*| , |*Rohrich, 78*|;

c) o uso mais intensivo de "*produções de erro*", inicialmente propostas por |*Peterson, 72*| adotados mais recentemente por |*Johnson, 77* | , |*Graham & Joy, 79*| e |*Fischer & Mauney, 80*| .

Vamos agora partir para a definição de um método automático de geração de recuperadores de erro para gramáticas de matrizes de transição.

5.3 – DEFINIÇÃO INFORMAL DO RECUPERADOR

Vamos inicialmente propor algumas das qualidades desejáveis de um "*bom*" recuperador de erros:

i) deve encontrar o maior número de erros possível em uma única passagem sobre o texto, marcando a localização de cada erro da forma mais exata;

ii) a mensagem fornecida deve ser clara e dirigir o programador no sentido da correção; a ação de recuperação deve ser explícita e clara;

iii) deve ser relativamente eficiente, em tempo e espaço; programas corretos e, principalmente, pedaços corretos de

programas incorretos não devem ser penalizados pelo esforço de recuperação;

iv) o número de mensagens para cada erro deve ser minimizado; em particular não deve assinalar erros causados pelo próprio recuperador;

v) o recuperador deve ser gerado automaticamente a partir de uma descrição formal da sintaxe e, se necessário, deve fornecer maneiras de especificar formalmente outras informações adicionais;

vi) a informação básica para decisão da ação do recuperador deve ser o próprio programa do usuário; caso não seja suficiente, informações estatísticas sobre erros mais frequentes devem ser usadas;

vii) o recuperador não deve tomar decisões arbitrárias; é preferível que desista da recuperação de um trecho limitado a arriscar-se a introduzir erros originalmente inexistentes;

viii) o recuperador deve tirar proveito das características do método de análise sintática, em nosso caso o método de matrizes de transição;

O primeiro ponto a se considerar é a importância da correção de erros locais, entendidos como aqueles causados por um único terminal. Esta importância é ainda maior pela *"preferência"* dos programadores em cometer certos tipos de erros: como vimos, 20% dos erros analisados por |*Ripley & Druseikis, 78*| eram falta do símbolo ";". Um recuperador de erros locais cobriria cerca de 85% dos programas errados (cerca de 95% do total dos programas).

Dois problemas emergem quando da tentativa de se implantar o recuperador de erros locais:

a) uma certa configuração do analisador pode admitir mais de uma recuperação possível. Por exemplo:

"... a b+c ..."

corrigido distintamente em:

"... a := b+c ..."

e "... a [b+c] ..."

b) apesar de, por hipótese, admitirmos que um Único "*token*" seja a causa do erro não há qualquer limite para o tamanho do contexto que será necessário analisarmos para sanar a dúvida;

c) em alguns casos, nem todo o restante do programa sanará a dúvida. Por exemplo:

"...; a+b ;"

d) outros erros, mesmo que menos prováveis, não são erros locais. Por exemplo:

"... a := (b + else; ..."

Um ponto bastante discutido na literatura é a concentração de contexto à esquerda (o passo atrás ou "*backward move*"). |*Rhodes, 73*| o utiliza indiscriminadamente sempre que um erro é detectado e, é claro, houver uma redução possível. Já |*Penello & De Remer, 78*| preferem preservar a situação da pilha e concentrar esforços no contexto à direita. Parece-nos que essa distinção se deve aos diferentes métodos de análise sintática subjacentes a cada proposta (embora ambos se arroguem serem métodos "gerais"). |*Rhodes, 73*| trabalha sobre um analisador de precedência simples e apenas um símbolo está no topo de sua pilha; já |*Penello*

& De Remer, 78] dispõe de um analisador LR onde um estado representa totalmente uma certa configuração de pilha. Dessa forma o passo atrás prematuro pode ser inclusive perigoso. Veja-se pelo exemplo:

"... if a or b then c:= f g ..."
↑

Se a pilha for reduzida indiscriminadamente corremos o risco de considerar o trecho "... c := f ..." como um comando completo, impedindo assim de se admitir outras alternativas para continuações começando por "g...".

No caso particular das GMT temos no topo da pilha e na variável MEIO o conteúdo codificado de uma produção. Isto significa que se o terminal causador do erro estiver no "*interior*" de uma produção é conveniente que não se façam reduções na pilha. Entretanto, se o "*token*" causador do erro estiver após o final direito da produção o passo atrás será bem-vindo. Adotaremos, portanto, uma solução adequada a essas características.

Outro ponto que precisa ser justificado é a necessidade do passo a frente ("*forward* move"). Sua utilização é que distingue uma recuperação de erro local mais sofisticada da recuperação pré-programada, absolutamente trivial. Alguns autores tendem a proporcionar ao recuperador de erros a mesma quantidade de informações que proporcionam ao analisador, isto é: o topo da pilha e o símbolo entrante. Ora o fato mais evidente em qualquer recuperação de erros é que estas informações não são suficientes para decidir entre diversas opções de recuperação local. Em geral este método mais trivial termina por fazer uma escolha arbitrária de alguma continuação possível. E a consequência direta e imediata é o aparecimento de erros espúrios em cascata, dificilmente controlável. Veja-se o exemplo:

"a or b then c := d else c := f "

Um caso típico de "if" omitido que com uma recuperação pré-programada leva a:

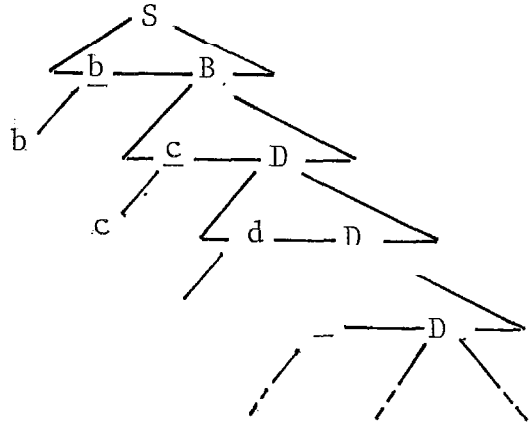
"a or b then c := d else c:= f "
↑ TROCADO POR :=
↑ TROCADO POR ;
↑ TROCADO POR ;

O exemplo por si só apresenta todas as vantagens do passo a frente como elemento de decisão da recuperação local. Note que o principal problema no exemplo acima é que o prefixo da sentença pode ser um prefixo admissivel neste contexto. Qualquer analisador em um caso destes irá detectar o erro apenas depois de ter passado pelo ponto onde a correção seria adequada (inserção de "if").

Isto nos leva a discussão da propriedade dos prefixos corretos e sua utilidade em detecção de erros. As gramáticas de matrizes de transição, como frisado em capítulos anteriores, não possuem a propriedade dos prefixos corretos e muito menos de detecção imediata de erro. O exemplo a seguir ilustra o fato:

1:	S → aB	1:	S → a B
2:	S → bC	2:	S → b C
3:	B → cD	3:	B → c D
4:	C → c	4:	C → c
5:	D → dD	5:	D → d D
6:	D → d	6:	D → d
		7:	a → a
		8:	b → b
		9:	c → c
		10:	d → d

a sentença errada é
"b c dddd ..."



A análise sintática seria:

PILHA	MEIO	AÇÃO	ENTRADA
#	E	≪ b	b c d d d d...
# b	E	≪ c	c d d d d...
# b c	E	≪ d	d d d d...
# b c d	E	≪ d	d d d...
# b c d d	E	≪ d	d d...

O erro será percebido em ponto arbitrariamente distante do primeiro ponto onde poderia ser detectado.

Esta propriedade é fundamental para satisfazer adequadamente nossos objetivos de apontar com a máxima precisão para o ponto do programa onde ocorreu o erro. Algum esforço adicional será feito por nosso recuperador para melhorar sua precisão. Infelizmente, o método de matriz de transição não é talhado para satisfazer tal requisito. Entretanto há alguma compensação: é extremamente difícil – como apontado por |*Rhodes, 73*| e |*Ripley & Druseikis, 76*| – reencetar a análise, depois de detectado o erro, em um analisador que goze da propriedade dos prefixos corretos, simplesmente porque nesses analisadores cada movimento pressupõe e exige um prefixo correto no interior da pilha. A solução de tal problema é o ponto central dos artigos de |*Ripley & Druseikis, 76*| e |*Graham & Joy, 79*|.

Vamos agora definir as características fundamentais do nosso recuperador:

a) far-se-á uso intensivo do passo a frente, de modo a se tomar a decisão de qual será a ação do recuperador no Último instante possível. Em particular o recuperador permitirá ao

analisador continuar com a análise do trecho seguinte ao erro mesmo que a ação completa do recuperador não esteja ainda decidida. Isto significa que o restante da entrada será analisada – inclusive todo o restante da entrada – permitindo que novos erros sejam encontrados;

b) dar-se-á preferência a solução *"inserir um símbolo"*, porque omitir um símbolo é o erro mais comum (50%); apesar disso o recuperador terá ainda capacidade de efetuar a remoção de um símbolo e de efetuar a troca de um símbolo combinando remoção e inserção;

c) diferentemente dos métodos conhecidos o recuperador poderá inverter a ordem de concentração dos contextos: só se fará o passo atrás quando a configuração do analisador indicar que não há possibilidade de tentar primeiro o passo a frente. Se este for possível será feito, deixando-se para completar a recuperação de erro mais tarde (vide algorítmo a seguir);

d) o recuperador não necessitará de qualquer tabela adicional baseando-se exclusivamente na tabela obtida do analisador sintático, sendo portanto um algorítmo adequado a qualquer gramática de matriz de transição; em particular não usará de informações adicionais do tipo custos de inserção ou remoção;

e) o recuperador será dedicado à recuperação de erros locais o que significa que necessita da existência de um segundo nível de tratamento de erros, sendo alguma variante do *"panic mude"* perfeitamente indicada;

f) o recuperador terá, ainda, um procedimento especial para reduzir ao mínimo o efeito da detecção postergada de erros devido ao método de matrizes de transição;

Estas características permitem classificá-lo como membro da família de recuperadores de |*Rhodes, 73*|.

Antes de descrever o comportamento do recuperador frente a uma situação de erro, vamos determinar quais as classes de erros que poderão ocorrer em uma análise pelo método GMT:

i) a AÇÃO não é definida: neste caso encontramos um símbolo na entrada que não se coaduna com o conteúdo da pilha para formar qualquer sentença válida. É importantíssimo distinguir nesse instante o conteúdo da variável MEIO (que contém um não-terminal):
   - se MEIO=ε é provável estarmos frente a um erro local causado por omissão de algum símbolo ou pela impropriedade do terminal entrante;
   - se MEIO ≠ ε estaremos frente a um caso diverso uma vez que, garantidamente, o "*token*" entrante é um seguidor do não-terminal MEIO (visto que a redução foi feita) *e*, principalmente, o não-terminal MEIO é um seguidor do estrelado no topo da pilha (visto que não há erro em GOTO);

   A maior probabilidade de termos MEIO = ε é indicada pelo maior número de pontos de erro na tabela ação no quadrante (U, ε). (No exemplo do capítulo anterior 202 pontos de erro em (LL, ε) contra 11 em (U , A)).

ii) GOTO não é definida: neste caso o analisador permitiu a redução de uma subsequência indevidamente, deixando de anotar a presença do erro em um ponto anterior (no início ou no interior da subsequência) ou – para gramáticas não unicamente inversíveis – foi efetuada uma redução baseada no terminal entrante. Temos aqui um erro imprecisamente localizado pela falta

da propriedade dos prefixos corretos do. analisador, Entretanto, este tipo de erro se manifesta de forma particular – erro em GOTO – o que permitirá algum tipo específico de tratamento;

Antes de descrevermos formalmente o algorítmo (na seção seguinte) vamos tentar expor sua idéia básica, para erros que não sejam erros em GOTO.

No momento da detecção do erro o analisador está em configuração não permitida de (TOPO, MEIO, ENTRADA), digamos (U, B, a). Nossa hipótese fundamental é que apenas um terminal é o causador do erro, provavelmente por omissão, não desprezando, porém, a possibilidade de inserção ou troca.

Nosso objetivo será analisar o trecho a direita – que começa por a – de modo que possamos isolar um trecho, parcialmente analisado, que seria:

U B || A c

onde || simboliza o terminal errado, A é o não-terminal reduzido a partir do ponto do erro (que pode se iniciar com a) e c é o terminal mais à direita onde tivermos que chegar nossa redução à frente.

Conseguida essa configuração, que representando toda a pilha seria:

$$\cdots \underline{U}_k \underline{U}_{k-1} \cdots \underline{U}_1 \underline{U} \; B \quad || \quad A \quad c$$

tentaríamos resolver a situação, possivelmente fazendo uso do passo atrás (reduções no conteúdo da pilha, começando por U B).

Entretanto isso não é tão simples como parece:

i) dada a configuração de erro (U, B, a) – correspondente a forma sentencial $\alpha U B a \omega$ – é possível que possamos inserir imediatamente um terminal f construindo

$$\alpha\ U\ B\ f\ a\ \omega$$

que resolve o problema sem precisar do passo a frente. Entretanto, se $B = \varepsilon$ estaríamos tomando a decisão com muito pouco contexto. Optamos então por tentar inserção imediata apenas quando $B \neq \varepsilon$;

ii) algumas vezes o estado (U, B) não admite um não terminal seguidor – ou seja-

$$\$\ A \in N \mid \text{GOTO}\ [U, B]\ \#\ \text{erro}$$

Neste caso é conveniente que façamos inicialmente o passo atrás;

iii) muitas vezes o terminal da entrada não admite ser colocado na pilha, ou seja, não é início de nenhuma produção. Se não tivermos alguma opção para passo atrás seremos obrigados a eliminá-lo;

iv) note que todo nosso processo de recuperação se baseia na possibilidade de admitirmos o terminal de entrada na pilha, mesmo que isso signifique admitir um prefixo incorreto. Estaremos usando uma deficiência do método de análise sintática para aperfeiçoar o recuperador de erros;

O próximo algorítmo descreve detalhada e formalmente o processo. Nesse algorítmo está incluido um procedimento especial para erros em GOTO.

5.4 — ALGORÍTMO DE RECUPERAÇÃO

O algorítmo a seguir admite ser chamado em três situações distintas, sob controle do analisador:

a) Erro na tabela AÇÃO — significando que um novo erro foi detectado;

b) Topo da pilha é o marcador || — este marcador é colocado pelo próprio recuperador assinalando um ponto de erro que não foi resolvido ainda porque se preferiu iniciar um passo a frente; o analisador efetua tal chamada quando o passo a frente foi completado;

c) Erro em GOTO — significando que um novo erro foi detectado, porém não temos certeza de sua localização;

Algorítmo 7 (LXXV)

RECUPERADOR DE ERROS SINTÁTICOS GMT

Passo 1: (determina SITUAÇÃO)
Caso SITUAÇÃO seja:

ERROAÇÃO : Faça B = MEIO e MEIO = E;
Execute o passo 2;

ERROGOTO : Execute o passo 4;

TOPO || : Execute o passo 5;

Passo 2: (escolhe caminho a tomar)
(configuração: TOPO, B, ENTRADA)
Se B ≠ ε então execute passo 4; (inserção)
Se AJUSTADO = verdadeiro volte ao analisador;

Se B = ε e não existe $X \in N$ | GOTO [TOPO,X] ≠ erro

ou B ≠ ε e não existe $b \in \Sigma$|AÇÃO [GOTO [TOPO,B] ,b] ≠ erro

então: execute o passo 3; (passo atrás inevitável)

se AJUSTADO = verdadeiro

então repita o passo 2;

Se existe E = (TOPO, ε) | AÇÃO [E, ENTRADA] = ≪ Ê

então: – PUSH(B); PUSH(||); PUSH(Ê); (passo a frente)

– Avance entrada;

– Volte ao analisador;

senão: – escreva "IGNORADO ENTRADA"; (remoção)

– avance entrada;

– recomece passo 2;

Passo 3: (passo atrás)

Faça F = GOTO [TOPO, B] ;

Se existe b|AÇÃO [F,b] = ⇒ j e

GOTO [TOPO-1, ESQUERDO [j]] ≠ erro

então: – faça AJUSTADO = verdadeiro;

B:= ESQUERDO [j] ;

POP;

senão faça AJUSTADO = falso

Passo 4: (inserção de um terminal)

(configuração: TOPO, B, MEIO, ENTRADA)

Se B = ε e AÇÃO [GOTO [TOPO,MEIO] , ENTRADA] # erro

ou MEIO=ε e AÇÃO [GOTO [TOPO,B] , ENTRADA] # erro

então faca: se MEIO = ε então MEIO := B;

AJUSTADO = verdadeiro;

senão:
se existe: c|AÇÃO [GOTO [TOPO,B] ,c] = $\lessdot \hat{E}$ OU $\overset{*}{=} \hat{E}$
e AÇÃO [GOTO [$\hat{E}$,MEIO] , ENTRADA] # erro

então: – escreva "INSERIDO c";
– se AÇÃO [GOTO [TOPO,B] ,c] = $\overset{*}{=} \hat{E}$ então POP;
– PUSH ($\hat{E}$) ; faça AJUSTADO = verdadeiro;
senão faça AJUSTADO = falso;

Passo 5: (marcador || no topo da pilha)
POP; faça B = TOPO; POP;

Passo 6: (decisão final de recuperação)
Execute o passo 4; (inserção)
Se AJUSTADO = verdadeiro volte ao analisador;
Execute o passo 3; (passo atrás)
Se AJUSTADO = verdadeiro repita o passo 6
Senão PANICMODE;

Passo 7: (erro em GOTO)
Faça B = E e execute o passo 4; (inserção)
Se AJUSTADO = verdadeiro volte ao analisador
Senão PANICMODE;

■

Note que diversos testes não implicam em varrer uma linha ou coluna da tabela pois podemos utilizar a informação já compactada nas funções AVANÇA, REDUZ e CONCENTRA. Isto quer dizer que os seguintes testes são equivalentes:

(i) $\exists E$| AÇÃO [E,a] = $\lessdot$, ou seja varrer uma coluna da tabela, é equivalente a AVANÇA(ENTRADA) $\neq \Phi$;

(ii) $\exists b$|AÇÃO [E,b] = $\gtrdot$ ou AÇAO [E, B] = $\doteq$ é equivalente a REDUZ(E) $\neq \phi$ ou CONCENTRA(E) $\neq \phi$;

Estes testes simplificados melhoram bastante a eficiência do algorítmo.

Por outro lado, para o funcionamento adequado do recuperador são necessárias duas alterações no processo de compactação:

(i) que não se faça fusão entre estados do primeiro quadrante (U, E) com estados do terceiro quadrante (U, A);

(ii) que, em consequência, a função AVANÇA seja desdobrada em duas linhas, uma para cada quadrante acima referido;

Estas providências afetam muito pouco a eficiência da compactação.

Vamos agora acompanhar o funcionamento do recuperador através de alguns exemplos. Os exemplos foram selecionados procurando mostrar a recuperação de erros no quadrante (U, A), no quadrante (U, E), em GOTO e erros por inserção e troca de símbolos. Os exemplos mostram sentenças com mais de um erro apenas por facilidade de exposição, uma vez que o recuperador pretende apenas tratar erros locais. Para que os exemplos possam ser adequadamente seguidos vamos antes descrever o analisador sintático adaptado ao uso do recuperador.

Algoritmo 8 (vide Alg. 4) (LXXVI)

ANALISADOR SINTÁTICO GMT COM RECUPERAÇÃO DE ERROS

Passo 1: (Inicialização)

PUSH(#); Faça MEIO = E; SCAN;

Passo 2: (Ciclo de Análise)

Faça X = topo da pilha;

Se X = || chame RECUPERADOR (TOPO ||) e repita o passo 2;

Se MEIO ≠ E então

se GOTO [X, MEIO]= não definido

então chame RECUPERADOR (ERROGOTO) e repita o passo 2;

senão faça X = GOTO [X, MEIO] ;

Caso AÇÃO [X; ENTRADA] seja

◁ : PUSH (AVANÇAREDUZ [X, ENTRADA]);

Faça MEIO = E ; SCAN; repita o passo 2;

≐ : POP; PUSH (AVANÇAREDUZ [X, ENTRADA]);

Faça MEIO = E; SCAN; Repita o passo 2;

▷ : Faça J = AVANÇAREDUZ [X, ENTRADA] ; escreva J;

POP; Faça MEIO ESQUERDO[J]; repita o passo 2;

"pare" : escreva "ACEITO" e pare;

"erro" : chame RECUPERADOR (ERROAÇÃO); repita o passo 2;

Exemplo (LXVII)

RECUPERAÇÃO DE ERROS

A gramática é a mesma do capítulo anterior (XLIV) e a tabela de controle é a obtida na saída do gerador de analisadores sintáticos (LXII).

(i) Sentença: # id := id
else if id
then if id := id) + (id #

P I L H A	B	MEIO	ENTR.	AÇÃO	RECUPERAÇÃO
# 1,		ε	id	<* 2	
# 1, id 2,		ε	:=	=* 9	
# 1, id:= 9		ε	if	<* 2	
# 1, id:= 9, id 2,		ε	else	*> 11	
# 1, id:= 9,		P	else	*> 3	
# 1,		A	else	ERRO	Erro em AÇÃO
# 1,	A	ε	else		Tenta inserção sem sucesso
					Tenta passo a frente sem sucesso
					Ignora else
# 1,	A	ε	if		Tenta inserção sem sucesso
					Tenta passo a frente. Com sucesso
# 1, A, ||, if 3,		ε	id	<* 2	
# 1, A, ||, if 3, id 2		ε	then	*> 13	
# 1, A, ||, if 3		B	then	=* 10	
# 1, A, ||, ifBthen 10,		ε	id	<* 2	
# 1, A, ||, ifBthen 10, id 2		ε	:=	=* 9	
# 1, A, ||, ifBthen 10, id:= 9		ε	id	<* 2	
# 1, A, ||, ifBthen 10, id:= 9, id 2		ε	)	ERRO	Erro em AÇÃO
# 1, A, ||, ifBthen 10, id:= 9, id 2	ε	ε	)		Tenta passo atrás com sucesso *> 11

P I L H A	B	MEIO	ENTR.	AÇÃO	RECUPERAÇÃO
# 1, A, || , ifBthen 10, id:= 8,	P	ε	)		Tenta inserção sem sucesso
					Tenta passo a frente sem sucesso
					Ignorado ") "
# 1, A, || , ifBthen 10, id:= 9,	P	ε	+		Tenta inserção com sucesso
# 1, A, || , ifBthen 10, id:= 9, E+ 5,		ε	(	<* 4	
# 1, A, || , ifBthen 10, id:= 9, E+ 5, (4		ε	id	<* 2	
# 1, A, || , ifBthen 10, id:= 9, E+ 5, (4, id 2		ε	#	*> 11	
# 1, A, || , ifBthen 10, id:= 9, E+ 5, (4		P	#	ERRO	Erro em AÇÃO
# 1, A, || , ifBthen 10, id:= 9, E+ 5, (4	P	ε	#		Tenta inserção com sucesso
# 1, A, || , ifBthen 10, id:= 9, E+ 5, (E) 12		ε	#	*> 10	Inserido ") "
# 1, A, || , ifBthen 10, id:= 9, E+ 5		P	#	*> 7	
# 1, A, || , ifBthen 10, id:= 9,		E	#	*> 3	
# 1, A, || , ifBthen 10,		A	#	*> 4	

P I L H A	B	MEIO	ENTR.	AÇÃO	RECUPERAÇÃO
# 1, A, ||,		C	#	TOPO ||	
# 1	A	C	#		Tenta inserção sem sucesso
					Tenta passo atrás sem sucesso
					PANIC MODE

Os nomes dos estrelados aparecem na pilha apenas para visualização.

Saída obtida: (sem mensagem de erro)

```
   id := id
   else if id
- - - -↑ IGNORADO "else"
         then id := id ) + ( id #
- - - - - - - - - - - - ↑IGNORADO " ) "
- - - - - - - - - - - - - - - - - - ↑  INSERIDO " )"
```

(ii) Sentença " if id id
then then id id # "

P I L H A	B	MEIO	ENTR.	AÇÃO	RECUPERAÇÃO
#1,		ε	if	⋖* 3	
#1, if 3,		ε	id	⋖* 2	
#1, if 3, id 2,		ε	id	ERRO	Erro em AÇÃO
#1, if 3, id 2,	ε	ε	id		Tenta passo atrás com sucesso *⋗ 13
#1, if 3,	B	ε	id		Tenta inserção com sucesso
#1, if 3, Bor 7,		ε	id	≐ 13	Inserido "or"
#1, if 3, Borid 13,		ε	then	*⋗ 12	
#1, if 3,		B	then	≐ 10	
#1, ifBthen 10,		ε	then	ERRO	Erro em AÇÃO
#1, ifBthen 10,	ε	ε	then		Tenta passo atrás sem sucesso
					Tenta passo a frente sem sucesso. Salta then

P I L H A	B	MEIO	ENTR.	AÇÃO	RECUPERAÇÃO
# 1, ifBthen 10,	ε	ε	id		Tenta passo a frente com sucesso
# 1, ifBthen 10, ε, || , id 2,		ε	id	ERRO	Erro em AÇÃO
# 1, ifBthen 10, ε, || , id 2,	ε	ε	id		Tenta passo atrás
					Erros muito próximos
					PANIC MODE

Saída obtida: (sem mensagem de erro)

```
   if id id
- - - - ↑ INSERIDO "or"
   then then  id id #
            ↑ IGNORADO "then"
- - - - - - - - - - ↑
```

(iii) Sentença: " id or id
then id id := id # "

P I L H A	B	MEIO	ENTR.	AÇÃO	RECUPERAÇÃO
# 1,		ε	id	⋖* 2	
# 1, id 2,		ε	or	*⋗ 13	
# 1,		B	or	ERRO	Erro em GOTO
					Tenta AÇÃO [1, id] , descarta
					AÇÃO [1, if] , OK
# 1, if M,		B	or	⋖* 7	
# 1, if M, Bor 7		ε	id	*= 13	
# 1, if M, Borid 13,		ε	then	*⋗ 12	
# 1, if 3		B	then	*= 10	
# 1, ifBthen 10		ε	id	⋖* 2	
# 1, ifBthen 10, id 2		ε	id	ERRO	Erro em AÇÃO
# 1, ifBthen 10, id 2	ε	ε	id		Tenta passo atrás sem sucesso
					Tenta passo a frente com sucesso

P I L H A	B	MEIO	ENTR.	AÇÃO	RECUPERAÇÃO
# 1, ifBthen 10, id 2, ε, ||, id 2		ε	:=	= 9	
# 1, ifBthen 10, id 2, ε, ||, id:= 9		ε	id	< 2	
# 1, ifBthen 10, id 2, ε, ||, id:= 9, id 2		ε	#	> 11	
# 1, ifBthen 10, id 2, ε, ||, id:= 9		P	#	> 3	
# 1, ifBthen 10, id 2, ε, ||		A	#	TOPO ||	Fim de passo a frente
# 1, ifBthen 10, id 2,	ε	A	#		Tenta inserção sem sucesso
					Tenta passo atrás sem sucesso
					PANIC MODE

Saída obtida: (sem mensagem de erro)

```
# id or id
  ↑       ↑ INSERIDO "if"
  then id id := id #
               t
```

iv) Sentença: "id := id or id # "

P I L H A	B	METO	ENTR.	AÇÃO	RECUPERAÇÃO
# 1,		E	id	⩽ 2	
# 1, id 2,		E	:=	= 9	
# 1, id:= 9,		E	id	⩽ 2	
# 1, id:= 9, id 2		E	or	⩾ 13	
# 1, id:= 9,		B	or	ERRO	Erro em GOTO
					Tenta AÇÃO [9,id] = ⩽ 2, descarta
					Tenta AÇÃO [9, ( ] = ⩽ 4, descarta
					PANIC MODE

O exemplo mostra o perigo de se efetuar recuperação quando o erro é na tabela GOTO. Houve redução equivocada para o não-terminal B e não há possibilidade de se determinar se corresponde a um Único terminal (como no caso) ou a um grande trecho da sentença de entrada.

Saída obtida: (sem mensagem de erro)

id := id or id
↑

Vejamos agora alguns exemplos de mau funcionamento do recuperador.

Exemplo (LXVIII)

Mesma gramática e tabelas do exemplo anterior.

(i) Sentença "id := (id +) # "

P I L H A	B	MEIO	ENTR.	AÇÃO	RECUPERAÇÃO
# 1,		ε	id	⋖ 2	
# 1, id 2,		ε	:=	≐ 9	
# 1, id:= 9,		ε	(	⋖ 4	
# 1, id:= 9, (4,		ε	id	⋖ 2	
# 1, id:= 9, (4, id 2,		ε	+	⋗ 11	
# 1, id:= 9, (9,		P	+	⋖ 5	
# 1, id:= 9, (9, E+ 5,		ε	)	ERRO	Erro em AÇÃO
# 1, id:= 9, (4, E+ 5	ε	ε	)		Tenta passo a frente sem sucesso Ignorado ") "
# 1, id:= 9, (4, E+ 5,	ε	ε	#		Tenta passo a frente sem sucesso PANIC MODE

Felizmente o erro estava demasiadamente próximo do fim da sentença para que o passo a frente aplicado errôneamente causasse problemas.

Sentença obtida: "id := (id +) #

↑ IGNORADO)

(ii) Sentença: "id := (id +) * id # "

P I L H A	B	MEIO	ENTR.	AÇÃO	RECUPERAÇÃO
⋮	⋮	⋮	⋮	⋮	
# 1, id:= 9, (4, E+ 5,	ε	ε	)		Ignorado)
# 1, id:= 9, (4, E+ 5,	ε	ε	*		Passo a frente sem sucesso
					Ignorado "*"
# 1, id:= 9, (4, E+ 5,	ε	ε	id		Passo a frente com sucesso
# 1, id:= 9, (4, E+ 5, ε, || , id 2		ε	#	=> 11	
# 1, id:= 9, (4, E+ 5, ε, || ,		P	#	TOPO ||	
# 1, id:= 9, (4, E+ 5	ε	P	#	=> 7	Tenta inserção com sucesso
# 1, id:= 9, (4,		E	#		Tenta inserção com sucesso
# 1, id:= 9, (E) 12		ε	#		INSERIDO)
					etc

Sentença obtida: " id := (id +) * id #

↑ IGNORADO)

↑IGNORADO *

↑ INSERIDO)

CAPÍTULO VI

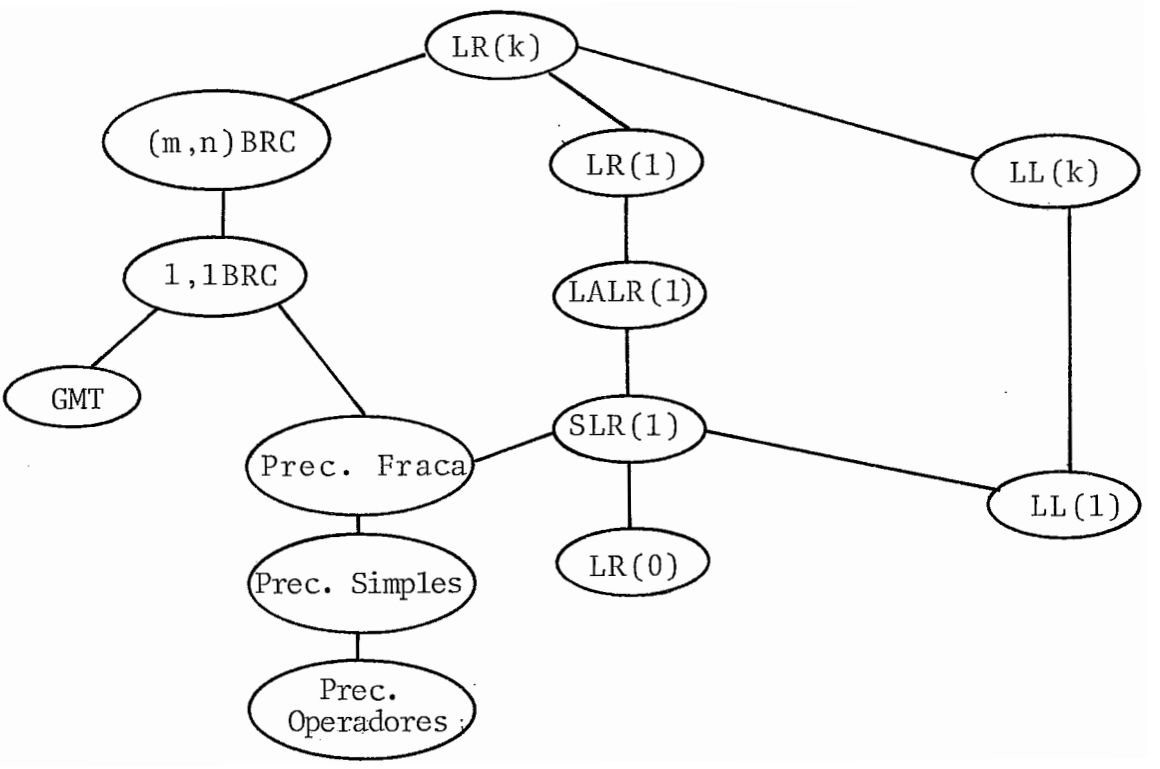
– AVALIAÇÃO E CONCLUSÕES –

6.1 – SOBRE O ANALISADOR SINTÁTICO

Nesta secção iremos verificar o comportamento do analisador sintático GMT e de seu construtor comparativamente ao método de análise sintática para gramáticas da classe SLR(1). A escolha desta Última classe para fins de comparação se deve ao fato que, juntamente com os analisadores LALR(1), serem os métodos atualmente utilizados para análise sintática ascendente. Suporemos que o leitor esteja familiarizado com estas subclasses das gramáticas LR e o remetemos à vasta bibliografia sobre o assunto |*Knuth, 65*| |*De Remer, 69*| |*De Remer, 71*| |*Lalonde, 71*| |*Anderson, 73*| |*Aho & Ullman, 72*| |*Horning, 74*| |*Aho & Johnson, 74*|.

– Amplitude da classe de gramáticas

|*Gries, 68*| coloca as GMT como um subconjunto próprio das gramáticas (1,1) Bounded Right Context. A exigência adicional das GMT de modo que não cobrem precisamente as (1,1) BRC é a necessidade de partirmos de uma gramática de operadores. Ressaltemos, entretanto, que o método é muito mais poderoso que precedência de operadores e mesmo precedência simples. A hierarquia das classes de gramáticas referidas, segundo |*Horning, 74*| será:



Não há qualquer critério absoluto para a escolha ou abandono de uma certa classe de gramáticas. As SLR(1) parecem ser uma classe suficientemente ampla para tratar a maioria das linguagens de programação usuais. O mesmo parece-nos se dar com as GMT, pois nossas experiências práticas com o gerador de analisadores sintáticos GMT implementado há cerca de dois anos no gerador de compiladores NHÃONHÃO da COPPE/UFRJ, nos permitem concluir que pertencem a classe da GMT gramáticas para PASCAL, SIMULA, ALGOL-60, um vasto subconjunto do ALGOL EXTENDED B-6700, além de diversas linguagens de pequeno e médio porte aqui projetadas. A rigor podemos afirmar que para cerca de 10 gramáticas submetidas

ao gerador não encontramos até aqui nenhuma gramática não-ambígua que deixasse de ser GMT. Parece, portanto, que é uma classe de gramáticas Útil e adequada aos padrões usuais das linguagens de programação.

– Eficiência do Construtor

Conforme vimos em secção precedente o construtor visita os pontos da tabela de controle apenas uma vez e apenas para os pontos onde as funções assumem valores diferentes de *"indefinido"* ou *"erro"*, e este passo é o dominante em complexidade de tempo. Temos, portanto, um algorítmo o mais eficiente possível.

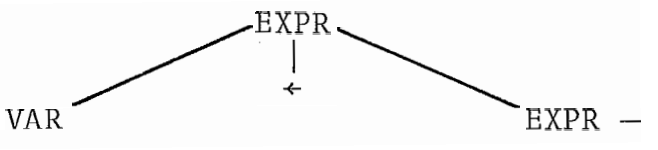
A implementação do construtor em execução no gerador de compiladores NHÃONHÃO, em Algol Extended B-6700, permite a obtenção das tabelas em tempo bastante razoável, sendo valores característicos tempos de execução de 60 a 90 segundos para a geração das tabelas de uma linguagem do porte do Pascal. Ainda assim cerca de 50% desse tempo é devido ao processamento da gramática em Backus-Naur Form (terminais e não-terminais alfanuméricos) e para a impressão detalhada das diversas saídas intermediárias do construtor. O construtor em sí consome cerca de 30 segundos para gerar as tabelas GMT do Pascal (da ordem de 9.000.000 de instruções).

– Eficiência do Analisador

O analisador sintático GMT é um programa com tempo de execução total proporcional ao comprimento da sentença a ser analisada. Entretanto, devido a sua capacidade de produzir um parse

esparso – a nos basearmos no clássico trabalho de |*Joliat, 73*| – deverá ser cerca de 40% mais rápido que o correspondente analisador SLR(1), para linguagens usuais.

Para que esta estimativa não pareça exagerada, citamos de |*Rushby, 77*|, um exemplo típico de derivação de sentença em EULER |*Wirth, 66*|, com a sentença " A ← B ":



VAR
VER –
λ
A

EXPR –
CATENA
DISJ
CONJ
CONJ –
NEGATION
RELATION
CHOICE
CHOICE –
SUM
SUM –
TERM
TERM –
FACTOR
FACTOR –
PRIMARY
VAR
VAR –
λ
B

O analisador SLR(1) faria esta análise em 26 passos enquanto que o analisador GMT faria o mesmo em 6 passos. Entretanto os "*otimizadores*" de analisadores SLR(1) permitem contornar este problema, a custa de uma geração mais elaborada. O mais com

pleto destes otimizadores é o já citado |*Rushby, 77*|.

– Espaço Ocupado pelas Tabelas de Controle

Um ponto vantajoso do método aqui proposto é o fato de suas tabelas serem demonstradamente menores que seus concorrentes diretos em análise ascendente e comparáveis às tabelas de analisadores descendentes. Vamos verificar a eficiência do método de compactação com os resultados abaixo, obtidos para a gramática exemplo do Capítulo IV.

COMPARAÇÃO DA EFICIÊNCIA DE COMPACTAÇÃO DAS TABELAS (bytes)			
GMT |*Gries, 68*|	GMT MODIFICADA SEM COMPACTAÇÃO	GMT MODIFICADA PSEUDO MÍNIMA	GMT MODIFICADA FINAL
Matriz de transição: 70 Procedimentos: 588 + Total 658 + Estimado por baixo	Ação : 83 Avança-Reduz : 165 Goto : 57 Esquerdo: 7 Total 312	Ação : 53 Avança-Reduz : 104 Goto : 23 Esquerdo: 5 Total 185	Ação : 33 T-Ação : 9 Avança : 6 Reduz : 18 Concentra: 10 Goto : 23 Esquerdo : 5 Total 104
SLR(1) |*De Remer, 69*|	SLR(1) |*Anderson,73*|	LL(1) |*Lewis,68*|	RRPSLR(1)|*Simone,80*|
Action : 72 Shift-Reduce : 179 Goto : 130 Esquerdo: 6 Total 387	Action : 36 T-Action: 9 Shift-Reduce : 72 Goto : 24 N-Goto : 4 Esquerdo: 6 Total 151	Controle : 76 Produções: 24 Esquerdo : 13 ~ ~ t a l 113	Action : 50 Shift-Reduce : 149 Goto : 41 Esquerdo: 1 Pop : 3 Total 244

Na gramática exemplo a compactação reduziu a tabela a menos de 15% do espaço ocupado pelo método original de |Gries, 68|. A memória ocupada pelo algorítmo em nosso esquema é desprezível frente ao espaço ocupado pelas tabelas. Vemos, também, que o analisador GMT é cerca de 35% menor que o correspondente SLR(1), neste exemplo.

Para que não fiquemos apenas no nível de um exemplo vamos demonstrar que para uma gramática $G \in (\mathcal{C}_{GMT} \cap \mathcal{C}_{SLR(1)})$ o analisador GMT é sempre menor que o analisador SLR(1), mesmo antes de compactado.

■ Teorema 16 (LXXIX)

Seja $G \in (\mathcal{C}_{GMT} \cap \mathcal{C}_{SLR(1)})$ e sejam M_1 e M_2 as máquinas características mínimas dos correspondentes analisadores sintáticos GMT e SLR(1).

Se $M_1 = (Q_1, \pi, \delta_1, q_{01}, F_1)$ e $M_2 = (Q_2, \pi, \delta_2, q_{02}, F_2)$ então $\#(Q_1) \leqslant \#(Q_2)$.

Demonstração

Notemos, inicialmente, que $\pi = (N \cup C)$ para ambas as máquinas o que implica em termos uma tabela maior para o analisador com maior número de estados.

Sabemos, pela definição LXVII, o que é uma máquina característica GMT. chamá-la-emos mínima quando submetida ao algorítmo clássico de minimização por classes de equivalência, levando em conta as entradas inacessíveis ("ϕ").

Uma máquina característica SLR(1) mínima será obtida da mesma forma, usando-se as funções ACTION, SHIFTREDUCE e GOTO

(na nomenclatura de |*Aho & Ullman, 76*|), com os pontos inacessíveis determinados conforme |*Anderson, 73*|.

A demonstração é simples e vamos supor que G possua uma produção da forma:

A → a B c d e F g

Pelo próprio funcionamento do algorítmo construtor SLR(1) – que é o construtor do conjunto de ítens LR(0) – sabemos que os seguintes estados existirão:

1: A → • a B c d e F g

2: A → a • B c d e F g

3: A → a B • c d e F g

4: A → a B c • d e F g

5: A → a B c d • e F g

6: A → a B c d e • F g

7: A → a B c d e F • g

8: A → a B c d e F g •

Pelo funcionamento do algorítmo construtor GMT teremos os seguintes estados:

1: (a , ε)

2: (a , B)

3: (aBc , ε)

4: (aBcd , ε)

5: (aBcde , ε)

6: (aBcde , F)

7: (aBcdeFg , ε)

Aparentemente, temos um estado a mais no analisador SLR(1). Isto não é verdade pois o ítem correspondente ao primei-

ro estado – A → • a B c d e F g – ocorre no interior do conjunto de Itens de outro estado, surgindo por ocasião do fechamento do conjunto de ítens.

A distinção se dá quando ambos tratam produções com prefixos comuns. Digamos:

A → a B c d

C → a B c f

No SLR(1) temos duas hipóteses:

(I) D → g S

S → A

S → C

A → a B c d

C → a B c f

com os estados

0: D → • g S

1: D → g • S

S → • A

S → • C

A → • a B c d

C → • a B c f

2: A → a • B c d

C → a • B c f

3: A → a B • c d

C → a B • c f

4: A → a B c • d

C → a B c • f

5: A → a B c d •

6: C → a B c f •

(II) $D \to A$ com 1: $D \to \cdot A$ 6: $S \to \cdot C$

$S \to C$ $A \to \cdot a\ B\ c\ d$ $C \to \cdot a\ B\ c\ f$

$A \to a\ B\ c\ d$ 2: $A \to a \cdot B\ c\ d$ 7: $C \to a \cdot B\ c\ f$

$C \to a\ B\ c\ f$ 3: $A \to a\ B \cdot c\ d$ 8: $C \to a\ B \cdot c\ f$

4: $A \to a\ B\ c \cdot d$ 9: $C \to a\ B\ c \cdot f$

5: $A \to a\ B\ c\ d \cdot$ 10: $C \to a\ B\ c\ f \cdot$

Na GMT sempre teremos

1: ($\underline{a}$, ε) 4: ($\underline{aBcd}$, ε)

2: ($\underline{a}$, B) 5: ($\underline{aBcf}$, ε)

3: ($\underline{aBc}$, ε)

Note que, no caso (I), os conjuntos de estados GMT e SLR(1) tem o mesmo número de elementos. Porém, no caso (II), um mesmo estado GMT corresponde a mais de um estado SLR(1) como por exemplo 3: (aBc, ε) que corresponde à 4: ($A \to a\ B\ c \cdot d$) e 9: ($C \to a\ B\ c * f$).

Note, também que este fato implica por um lado na diminuição do tamanho das tabelas de controle e por outro lado na perda da propriedade dos prefixos corretos.

Demonstramos que o construtor SLR(1) não cria menos estados que o construtor GMT (pois usamos uma produção genérica) e que há pelo menos um caso em que cria mais estados.

Logo $\#(Q_1) \leqslant \#(Q_2)$, c.q.d.

_____ ■

6.2 – SOBRE O RECUPERADOR DE ERROS

Vimos na secção anterior como se pode obter um analisador GMT que é menor e mais rápido que o correspondente SLR(1). Vimos, também, que isto se dá às custas da perda da propriedade dos prefixos corretos.

O recuperador proposto no Capítulo V, entretanto, faz uso exatamente dessa imperfeição para montar um esquema de recuperação barato e sofisticado: é exatamente por podermos dar entrada na pilha a um estado (a, E), independentemente do conteúdo da pilha, que podemos iniciar o passo a frente e verificar o contexto a direita, se necessário descobrindo novos erros. |*Penello & De Remer, 78*| e |*Ripley & Druseikis, 78*| demonstram que isto não pode ser feito em um analisador SLR(1) sem que seja preciso criar novos estados (aumentando o espaço em tabelas) para admitir a continuação.

O recuperador, portanto, possui qualidades invejáveis:

a) em primeiro ligar é constituído apenas por um algorítmo, isto é: pode ser acoplado a qualquer analisador GMT, não ocupa espaço com tabelas e serve para qualquer gramática GMT;

b) não interfere no funcionamento do analisador para programas corretos ou pedaços corretos de programas incorretos;

c) toma a decisão de recuperação a partir da análise do próprio programa do usuário, levando em consideração contexto à direita e esquerda do ponto de erro que podem ser arbitrariamente longos;

d) como o recuperador adia até o Último momento a decisão de recuperação torna-se razoavelmente imune a criar novos erros, pois a análise do contexto imediato à direita não é afetada fundamentalmente pela correção de erros anteriores;

e) permite corrigir, até certo ponto, as imprecisões do analisador; entretanto a ação do recuperador nesse caso foi feita intencionalmente tímida;

f) o recuperador está adequado aos tipos de erros estatisticamente mais comuns;

Acreditamos que este trabalho tenha conseguido tornar o método de análise sintática por matrizes de transição uma técnica atraente, outra vez, para uso em compiladores. O contexto mais adequado onde se poderia adotá-la seria para compiladores com severas restrições de memória e tempo de execução, onde se exige ainda uma recuperação de erros de qualidade. Seus concorrentes mais diretos são os métodos descendentes, que sabidamente ocupam menor espaço que os ascendentes. Porém tais métodos não admitem recuperação de erros barata e de qualidade.

De uma maneira geral os principais grupos de pesquisa em compiladores optaram por analisadores ascendentes, em particular LALR(1) e SLR(1): Univ. California Berkeley, Bell Laboratories, Univ. California Santa Cruz, Univ. Toronto, Univ. Karlsruhe, Univ. Munich, IRIA Paris e Univ. NewCastle. Seu trabalho, entretanto, visa compiladores para máquinas de grande porte. Nossas necessidades de inovação tecnológica são de outra espécie, com problemas até mais difíceis e com exigências maiores.

Esperamos ter demonstrado o quanto ainda pode ser extraído das GMT, através de um trabalho teórico de .profundidade dirigido a objetivos práticos. Não estamos propondo, todavia sua utilização indiscriminada mas a análise objetiva da escolha do método de análise sintática dentro das condições e especificações de cada compilador. Fica aqui mais uma opção, renovada.

BIBLIOGRAFIA

1. |*Aho & Johnson, 74*| – A. Aho & S. C. *Johnson*
"LR Parsing"
Comp. Surveys 6, nº 2, Jun/974.

2. |*Aho &* URRman, *72*| – A.Aho *& J.D.* URRman
"The theory of parsing, translation and compiling"
Prentice-Hall, 1972

3. |*Aho &* URRman, *73*| – A. Aho *& J.D.* URRman
"A technique for speeding up LR(k) parsers"
SIAM J. Comp. 2, p. 106, 1973

4. |*Aho &* URRman, 76) – A. *Aho & J.D.* URRman
"Principles of compiler design"
Prentice-Hall, 1976

5. |*Anderson*, 73| – *T. Anderson*, J. *Eve* & J.J. *Horning*
"Efficient LR(1) Parsers"
Acta Informatiea 2, p. 12, 1973

6. |*Backhouse*, 76| – R.C. *Backhouse*
"An alternative approach to the improvement of LR(k) parsers"
Acta Informatica 6, p. 277, 1976

7. |*Boullier*, 79| – P. *Boullier*
"Automatic syntatic error recovery for LR parsers"
State of art and future trends in compilation, p. 239
Montpellier, 1978.

8. |*Ciesinger, 79*| – *I. Ciesinger*
"A bibliography of error-handling"
Sigplan Notices, Ago 1979

9. |*Conway, 63*| – *M.E. Conway*
"Design of a separable transition diagram compiler"
CACM 6, p. 396, Jul 1963

10. |*Conway & Wilcox, 73*| – *M. E. Conwuy & T.R. Wilcox*
"Design and implementation of a diagnostic compiler for PL/I"
CACM 16, p. 169, Mar 1973

11. |*De Remen, 69*| – *F.L. De Remer*
"Practical Translator for LR(k) languages"
Ph.D. Thesis, M.I.T., 1969

12. |*De Remen, 70*| – *F.L. De Remen*
"Extended LR(k) grammars and their parsers"
Univ. California, Sta. Cruz, 1970

13. |*De Remen, 71*| – *F.L. De Remer*
"Simple LR(k) grammars"
CACM 14, p. 453, 1971

14. |*Demers, 75*| – *A.I. Demers*
"Elimination of single productions and merging nonterminal symbols of LR(1) grammars"
Comp. Lang. 1, p. 105, 1975.

15. |*Fischer, Dion & Mauney, 79*| – *C.N. Fischer, R.A. Dion & J. Mauney*
"A locally least-cost LR error corrector"

Tech. Rep. 363, Comp.Sci.Dept.Univ. Wisconsin - Madison, 1979.

16. |Fischer, Mauney & Milton, 79|— C.N.Fischer, J.Mauney & 0.17. Milton
"A locally least-cost LL(1) error corrector"
Tech. Rep. 371, Comp. Sci.Dept. Univ. Wisconsin, 1979.

17. |Fischer, Milton & Quiruig, 79| — C.N. Fischer, D.R. Milton, S.R. Quiruig
"Efficient LL(1) error correction and recovery using only insertions"
Comp.Sci.Dept. Univ. Wisconsin - Madison, 1979

18. |Fischer & Mauney, 80| — C.N. Fischer & J. Mauney
"On the role of error productions in Syntatic error correction"
Tech. Rep. 364, Comp.Sci.Dept. Univ. Wisconsin - Madison, 1980.

19. |Floyd, 63| — R. W. Floyd
"Syntatic analysis and operator precedence"
JACM 10, p. 316, Jul 1963.

20. |Graham & Joy, 79| — S.L. Graham, C.B. Haley & W.N. Joy
"Practical LR Error Recovery"
Proc. Comp. Constr. Conf., Denver
Sigplan Notices 8, Ago. 1979

21. |Gries, 68| — D. Gries
"Use of transition matrices in compiling"

CACM 11, p. 26, Jan 1968

22. |Gries, 71| – D. Gries
"Compiler construction for digital computers"
Wiley, 1971

23. |Hopcroft & Ullman, 79| – J.E. Hopcroft & J.V. Ullman
"Introduction to automata theory languages and Computation"
Addison-Wesley, 1979.

24. |Horning, 74| – J.J. Horning
"LR Grammars and analysers"
in 'Compiler Construction: an Advanced Course'
Springer-Verlag, 1974

25. |Hunt, 77| – H.B. Hunt III, T.G. Szymanski & J.D. Ullman
"Operations on sparse relations"
CACM 20, p. 171, Mar 1977

26. |Joliat, 73| – M.L. Joliat
"On the reduced matrix representation of LR(k) parser tabels"
Ph.D. Thesis, Univ. of Toronto, 1973

27. |Johnson, 77| – S.C. Johnson
"YACC - Yet another compiler compiler"
Bell Labs., Murray Hill, N.J., 1977

28. |Knuth, 65| – D.E. Knuth
"On the translation of languages from left to right"
Inf. and Control. 8, p. 607, Out 1965

29. |*La France, 70*| – *J.E. La France*
"Syntax directed error recovery for compilers"
Ph.D. Thesis, Univ. of Illinois.. at Urbana, 1970

30. |*LaLonde, 71*| – *W.R. LaLonde*
"An efficient LALR parser generator"
Tech. Rep. CSRG-2, Univ. Toronto, 1971

31. |*LaLonde, 75*| – *W.R. LaLonde*
"Practical LR analysis of regular right part grammars"
Ph.D. Thesis, Univ. of Waterloo, 1975

32. |*LaLonde, 76*| – *W.R. LaLonde*
"On directly constructing LR(k) parsers without chain productions"
3rd Symp. on Princ. of Progr. Languages, p.127, 1976

33. |*Leinius, 70*| – *R.P. Leinius*
"Error detection and recovery for syntax directed compiler systems"
Ph.D. Thesis, Univ. Wisconsin - Madison, 1970.

34. |*Lewis, Rosenkrantz & Stearns, 76*| – *P.M. Lewis, D.J. Rosenkrantz & R.E. Stearns*
"Compiler design theory"
Addison – Wesley, 1976

35. |*Litecky & Davis, 76*| – *C.R. Litecky e G.R. Davis*
"A study of errors, error-proneness and error diagnosis in COBOL"
CACM 19, p. 33, 1976

36. |*Pager, 74*| – D. *Pager*
"On eliminating unit productions from LR(k) parsers"in 'Lecture Notes in Computer Science' 14, p. 242
Springer-Verlag, 1974.

37. |*Pai, 79*| – A.B. Pai & R.B. *Kieburtz*
"Global contex recovery: a new strategy for parser recovery from syntax errors"
Proc. Comp. Constr. Conf., Denver
Sigplan Notices 8, Ago 1979

38. |*Penello* & De *Remer, 78*| – *T. Penello* & *F.* L De *Remer*
"A forward move for LR error recovery"
5th Symp. Princ. Progr. Lang. Jan 1978.

39. |*Peterson, 72*| – T.C. *Peterson*
"Syntax error detection, correction and recovery in parsers"
Ph.D. Thesis, Stevens, 1972

40. |*Rhodes, 73*| – S.P. *Rhodes*
"Practical syntatic error recovery for programming languages"
Ph.D. Thesis, Univ. California, Berkeley, 1973

41. |*Ripley* & *Druseikis, 76*| – G.D. *Ripley* & *F.C. Druseikis*
"Error recovery for simple LR(k) parsers"
3rd Symp. Princ. Progr. Lang., 1976

42. |*Ripley* & *Druseikis, 78*| – G.D. *Ripley* & *F.C. Druseikis*
"A statistical analysis of sintax errors"
Comp. Lang. 3, p. 227, 1978

43. |*Rohrich, 78*| – *J.Rohrich*
"Automatic construction of error correcting parsers"
Dr. Rer.Nat. thesis, Univ. Karlsruhe, 1978

44. |*Rushby, 77*| – *J.M. Rushby*
"LR(k) Sparse parsers and their optimizations
Ph.D. Thesis, Univ. NewCastle, 1977

45. |*Samelson & Bauer, 60*| – *K.Samelson & F.L.Bauer*
"Sequential formula translation"
CACM 3, p. 76, Fev 60

46. |*Simone, 80*| – *E. De Simone & L. C. Pereira*
"Algorítmos para Gramáticas RRP SLR(1)"
VII Sem. Int. Soft. Hard., Campinas, Jul 80

47. |*Soisalon-Soininen, 77*| – *E. Soisalon-Soininen*
"Elimination of single productions from LR parsers in conjunction with the use of default reductions"
4th Symp. Princ. Progr. Lang., 1977

48. |*Wirth & Weber, 66*| – *N. Wirth & H. Weber*
"EULER: a generalization of ALGOL and its formal definition"
CACM 8, p. 13, Jan 1966

49. |*Youngs, 72*| – *E.A. Younga*
"Error-proneness in programming"
Tech. Rep., Univ. North-Carolina, 1972.

SIMBOLOGIA ADOTADA

{ }	conjunto
#( )	número de elementos de um conjunto
■	início / fim de parágrafo
C	classe
∈	pertence
∩	intersecção
∪	união
⊂	continência
|	tal que
∧	e
∨	ou
¬	não
→	produz
⟹	deriva diretamente
⟸	reduz diretamente
$\overset{*}{\Longrightarrow}$	deriva em zero ou mais passos
$\overset{*}{\Longleftarrow}$	reduz em zero ou mais passos
$\overset{+}{\Longrightarrow}$	deriva em um ou mais passos
$\overset{+}{\Longleftarrow}$	reduz em um ou mais passos
$\overset{DIR}{\Longrightarrow}$	deriva à direita
$\overset{ESQ}{\Longrightarrow}$	deriva à esquerda
Φ	conjunto vazio ou "não importa"
■⇨	se ... então
⇦■⇨	se e somente se
A x B	produto cartesiano
ε	sequência vazia
R + S	soma
R^n	potência
R^{-1}	transposição
R^+	fechamento transitivo
R*	fechamento transitivo reflexivo
#	início / fim de sentença
O( )	ordem

NOVAS TÉCNICAS PARA ANALISADORES SINTÁTICOS TIPO MATRIZ DE TRANSIÇÃO

ESTEVAM GILBERTO DE SIMONE

PTS-12/81