ANALISADORES LÉXICOS COMPACTADOS

John Reed

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

Aprovada por:

Estevam Gilberto De Simone (COPPE/UFRJ)
Presidente

José Lucas Mourão Rangel Netto (COPPE/UFRJ)

Doris Ferraz Aragon (UFF)

Paulo Augusto Silva Veloso (COPPE/UFRJ)

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 1982

REED, JOHN

Analisadores Léxicos Compactados |Rio de Janeiro|, 1982.

IX, 67p. 29,7cm (COPPE-UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1982).

Tese - Universidade Federal do Rio de Janeiro, Fac. de Engenharia.

1. Analisadores Léxicos Compactados. I.COPPE/UFRJ, II. Analisadores Léxicos Compactados.

ã Helena

AGRADECIMENTOS

Ao professor Estevam Gilberto De Simone pela idéia e paciente orientação.

Ao professor José Lucas Mourão Rangel Netto e aos colegas Sérgio Schneider e Miguel Argollo de Teive Júnior pelas sugestões e colaboração.

Às professoras Doris Ferraz Aragon e Lígia Alves Barros pelo irrestrito apoio que me ofereceram.

Aos colegas do Programa de Engenharia de Sistemas e Computação pelo interesse e entusiasmo.

A meus amigos e em especial a Helena Muller pelo vital incentivo.

A Denise Schwartz Cupolillo pela paciência no trabalho de datilografia.

Ao CNPq, COPPE e UFF pelos recursos e oportunidade de concluir este trabalho.

RESUMO

Um dos principais problemas da implementação de reconhecedores de conjuntos regulares especificados por expressões regulares estendidas é o tamanho de suas tabelas. Este trabalho propõe um novo modelo de máquina finita - O Automato Finito com Contadores (AFC) - acrescida de dispositivos de contagem como uma forma de reduzir os tamanhos das tabelas; descreve um algoritmo para construção automática de AFCs além de sugerir novos tópicos de pesquisa relacionados ao desenvolvimento da idéia básica contida no AFC.

ABSTRACT

One of the main problems of implementing recognizers for regular sets specified by extended regular expressions is the size its tables. This work presents a new model of finite state machine - the Finite Automation with counters that includes counting devices as a means of reducing table sizes. Is also describes an algorithm for automatically constructing AFC, besides suggesting research topics related to the development of AFC basic idea.

ÍNDICE

Pág.

Pág.

Pág.

CAPÍTULO 1 - INTRODUÇÃO

I Um automato finito determinístico (AFD) para a expressão regular

$$\ell.(\ell|d)*^{5}$$

é

Esta expressão corresponde à definição léxica dos identificadores FORTRAN.

Para a linguagem ALGOL B6700 a correspondente definição léxica de identificadores será

$$\ell.(\ell|d)*^{62}$$

Deixamos de apresentar o correspondente AFD por economia de espaço.

O presente trabalho pretende desenvolver um novo modelo de máquina finita capaz de efetuar a contagem de símbolos associados a operadores *n, de modo a tornar possível sua utilização dentro de limites razoáveis de espaço ocupado.

Entretanto, como veremos a seguir, expressões regulares são descrições altamente poderosas de conjuntos regulares, tornando nosso problema bem mais complexo do que pode apresentar à primeira vista.

O AFC - AUTOMATO FINITO COM CONTADORES, proposto nesta tese, destina-se a esse objetivo.

II O capítulo 2 revisa a necessária base conceitual da nossa pesquisa.

O capítulo 3 expõe a idéia do AFC, compara-o ao AFD usual e apresenta um algoritmo de construção automática de AFC a partir da expressão regular.

No capítulo 4 discutem-se os resultados do trabalho e apresentam-se possíveis caminhos para o aperfeiçoamento da idéia básica - o Automato Finito com Contadores.

CAPÍTULO 2 - REVISÃO CONCEITUAL

2.1 - Gramáticas Regulares

Seguiremos a estruturação teórica de |Hopcroft-Ullman, 69|, |Aho-Ullman, 72| e |Aho-Ullman, 77|.

I Um ALFABETO (V) é um conjunto não-vazio de símbolos.

Ex.: $V = \{\ell, d\}$

II Uma PALAVRA ou cadeia (x) sobre V é uma sequência finita de símbolos quaisquer de V.

Ex.: $x = \ell d \ell \ell d$

III O COMPRIMENTO ($|x|$) da palavra x é o número de ocorrências de símbolos que a compõem.

Ex.: $x = \ell d \ell \ell d$ $\quad |x| = 5$

$y = ddd$ $\quad |y| = 3$

IV A PALAVRA NULA (ε) é a palavra composta de zero símbolos, logo $|\varepsilon| = 0$.

V A CONCATENAÇÃO (x.y, ou simplesmente xy) das palavras x e y é uma nova palavra z obtida pela justaposição da palavra x e então y. Temos que $|xy| = |x| + |y|$.

Ex.: $x = \ell d \ell \ell d$, $|x| = 5$

$y = ddd$, $|y| = 3$

$xy = z_1 = \ell d \ell \ell dddd$, $|z_1| = 8$

$yx = z_2 = dddld\ell\ell d$, $|z_2| = 8$

A concatenação não é comutativa, exceto em alfabetos de um só símbolo.

VI O FECHAMENTO TRANSITIVO REFLEXIVO (V*) do alfabeto V é o conjunto infinito(mas enumerável) de todas as palavras possíveis sobre V.

Ex.: $V = \{\ell, d\}$

$V^* = \{\varepsilon, \ell, d, \ell\ell, \ell d, d\ell, dd, \ell\ell\ell, \ell\ell d, \ell d\ell, \ell dd \ldots\}$

O FECHAMENTO TRANSITIVO (V^+) é definido como $V^+ = V^* - \{\varepsilon\}$.

VII Uma LINGUAGEM L é qualquer subconjunto de V*.

Ex.: $L_1 = \{\varepsilon, \ell, dd, \ell\ell d\}$

$L_2 = \{\ell, d, \ell\ell\ell, \ell d\ell, d\ell d, ddd, \ldots\}$

VIII Uma GRAMÁTICA G é um tipo de descrição finita de linguagem definida como:

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

onde N é um conjunto finito de categorias gramaticais;

Σ é um conjunto finito de palavras, com $N \cap \Sigma = \phi$;

$S \in N$ é o símbolo inicial de G;

P é o conjunto de regras gramaticais (ou sistema de reescrita) de G, da forma $\alpha \rightarrow \beta$ onde $\alpha \in (N \cup \Sigma)^+$ e $\beta \in (N \cup \Sigma)^*$.

Ex.: $G = (N, \Sigma, S, P)$

$N = \{I, J\}$

$\Sigma = \{\ell, d\}$

$S = I$

$P = \{ \quad I \rightarrow \ell J$

$J \rightarrow \ell J$

$J \rightarrow dJ$

$J \rightarrow \varepsilon \ \}$

IX A relação DERIVA ($\Longrightarrow$) em G sobre $(N \cup \Sigma)^+ X (N \cup \Sigma)^*$ definida como:

$\alpha\ \beta\ \gamma \Longrightarrow \alpha\ \delta\ \gamma$ sse $(\beta \rightarrow \delta) \in P$

Os fechamentos da relação deriva ($\overset{*}{\Longrightarrow}$, $\overset{+}{\Longrightarrow}$) são definidas da forma usual.

X A LINGUAGEM GERADA POR UMA GRAMÁTICA $G = (N,\Sigma,S,P)$, representada por $L(G)$, será:

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

XI Duas GRAMÁTICAS G e G' são ditas EQUIVALENTES($\equiv$) se geram a mesma linguagem, ou seja $L(G) = L(G')$.

Ex.: Sejam G a gramática do exemplo anterior e

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

$N' = \{I, J, K\}$

$\Sigma = \{\ell, d\}$

$S' = I$

$P' = \{ \quad I \to \ell J$

$J \to \ell K$

$J \to dK$

$J \to \varepsilon$

$K \to \ell K$

$K \to dK$

$K \to \ell$

$K \to d \quad \}$

No caso, $G' \equiv G$

XII Um RECONHECEDOR de L(G) é um algoritmo que recebendo $x \in \Sigma^*$

a) se $x \in L(G)$ emite "verdadeiro"

b) se $x \notin L(G)$ emite "falso"

XIII Uma GRAMÁTICA é dita REGULAR se as suas regras são das formas

$A \to aB$ ou $A \to a$ ou $A \to \varepsilon$, $a \in \Sigma$, $A, B \in N$

Ex.: P = { 1: $I \rightarrow \ell$

2: $I \rightarrow \ell J$

3: $J \rightarrow \ell$

4: $J \rightarrow d$

5: $J \rightarrow \ell J$

6: $J \rightarrow dJ$ }

Observações:

i) Os números que precedem as regras delas não fazem parte;

ii) omitimos a menção dos conjuntos N e Σ;

iii) o símbolo inicial é o lado esquerdo da primeira regra.

XIV Uma LINGUAGEM é dita REGULAR se puder ser descrita por uma gramática regular. Toda linguagem regular é um CONJUNTO REGULAR SOBRE Σ.

2.2 - Automato Finito

XV Um AUTOMATO FINITO DETERMINÍSTICO (AFD) é uma 5-tupla $M = (Q, \Sigma, \delta, q_0, F)$

onde Q é um conjunto finito não-vazio de estados;

Σ é um alfabeto finito de entrada;

δ é um mapeamento parcial de transições $(Q \times \Sigma) \rightarrow Q \cup \{-\}$, onde "-" indica que não há transição;

q_0 é um estado inicial único;

$F \subseteq Q$ é um conjunto de estados finais.

Um AFD atua da seguinte forma: recebe como entrada uma palavra que é lida, símbolo a símbolo, da esquerda para a direita, causando uma mudança de estado a cada leitura, em função do estado atual e do símbolo entrante.
Tal máquina é um reconhecedor e diremos que uma palavra x é aceita por M, ou é reconhecida por M se, ao término da leitura de todos os símbolos de x, o estado atual for um estado final. Caso, durante o reconhecimento, não exista transição possível ($\delta(q,a) = -$) a palavra não é aceita por M. ("-" denota $\emptyset$, o conjunto vazio).

XVI Uma CONFIGURAÇÃO de um AFD M é um elemento do conjunto $(Q \times \Sigma^*)$.

XVII Uma TRANSIÇÃO ($\vdash$) de um AFD M é uma relação sobre $(Q \times \Sigma^*) \times (Q \times \Sigma^*)$, ou seja, uma relação entre configurações, definida como:

$(q_i,ax) \underset{M}{\vdash} (q_j,x)$ sse $\delta(q_i,a) = q_j$, $q_i,q_j \in Q$, $a \in \Sigma$, $x \in \Sigma^*$

Formalmente, um AFD reconhecerá uma palavra x se

$(q_0,x) \underset{M}{\overset{*}{\vdash}} (q_j,\varepsilon)$ e $q_j \in F$

Ex.: $M = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{\ell, d\}$

$\delta =$ $\delta(q_0, \ell) = q_1$ $\delta(q_0, d) = q_2$

$\delta(q_1, \ell) = q_1$ $\delta(q_1, d) = q_2$

$\delta(q_2, \ell) = q_1$ $\delta(q_2, d) = q_3$

$\delta(q_3, \ell) = q_1$ $\delta(q_3, d) = q_3$

$F = \{q_3\}$

M é o AFD que reconhece todas as palavras sobre $\{\ell, d\}$ que terminam com dd.

Reconhecendo $x = d\ell ddd$

$(q_0, d\ell ddd) \vdash_M (q_2, \ell ddd) \vdash_M (q_1, ddd) \vdash_M (q_2, dd)$
$\vdash_M (q_3, d) \vdash_M (q_3, \varepsilon)$

$q_3 \in F \therefore x \in L$

XVIII A todo AFD $M = (Q, \Sigma, \delta, q_0, F)$ está associado um DIAGRAMA DE TRANSIÇÕES que é um grafo direcionado $g = (Z, X)$ com atributos nos vértices e rótulos nas arestas, definido como:

a) existe uma correspondência bi-unívoca entre Q e Z;

b) todo vértice correspondente a um estado $q \in F$ tem um atributo 'final' no diagrama;

c) o vértice correspondente ao estado inicial q_0 tem o atributo 'inicial' no diagrama;

d) existe uma correspondência bi-unívoca entre as arestas X e a função de transição δ de modo que (q_i, q_j) com rótulo a está em X sse $\delta(q_i, a) = q_j$.

Ex.: No exemplo, o atributo inicial é indicado por * e o atributo 'final' é indicada por círculos duplos.

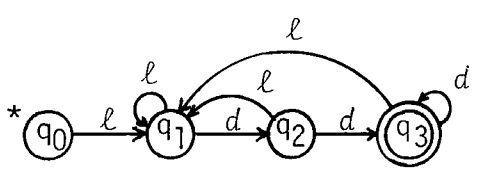


Diagrama de transições do AFD do exemplo anterior.

XIX Um AFD também pode ser descrito na forma de TABELA DE TRANSIÇÕES, contendo o mapeamento δ e com os estados inicial e finais assinalados com marcas próprias: * e F , respectivamente.

Ex.:

		ℓ	d	
*	q_0	q_1	q_2	(estado inicial)
	q_1	q_1	q_2	
	q_2	q_1	q_3	
	q_3	q_1	q_3	F (estado final)

Tabela de transições do exemplo anterior .

XX Demonstra-se que o conjunto de palavras aceito por um AFD M qualquer é um conjunto regular |Hopcroft-Ullman, 79| . A esse conjunto será dado o nome $\tau(M)$.

XXI A gramática regular correspondente a um AFD M é obtenível, por exemplo, do seu diagrama de transições:

a) N será um conjunto de categorias gramaticais com correspondência bi-unívoca aos vértices de Z;

b) Σ será o conjunto de palavras com correspondência bi-unívoca aos rótulos das arestas de X;

c) para cada aresta (q_i, q_j) com rótulo a existirá uma regra em P da forma $I \rightarrow aJ$ onde I e J são categorias gramaticais associadas a q_i e q_j respectivamente;

d) para cada aresta (q_i, q_j) com rótulo a, onde $q_j \in F$, existirá uma regra em P da forma $I \rightarrow a$ onde I é a categoria gramatical associada a q_i.

Ex.: G: (q_0) 1: $A \rightarrow \ell B$

2: $A \rightarrow d C$

(q_1) 3: $B \rightarrow \ell B$

4: $B \rightarrow d C$

(q_2) 5: $C \rightarrow \ell B$

6: $C \rightarrow d D$

7: $C \rightarrow d$

(q_3) 8: $D \rightarrow \ell B$

9: $D \rightarrow d D$

10: $D \rightarrow d$

G é uma gramática regular que descreve a linguagem aceita pelo AFD do exemplo anterior.

Diz-se que dois AFDs M e M' equivalentes ($\equiv$) sse $\tau(M) = \tau(M')$.

XXII Um AUTOMATO FINITO NÃO DETERMINÍSTICO (AFND) é uma 5-tupla $M = (Q, \Sigma, \delta, q_0, F)$ onde Q, Σ, q_0 e F são análogos ao definido para AFD, porém δ é uma função de transição de $(Q \times \Sigma) \rightarrow \mathcal{P}(Q) \cup \{-\}$, onde $\mathcal{P}(Q)$ é o conjunto dos subconjuntos de Q.

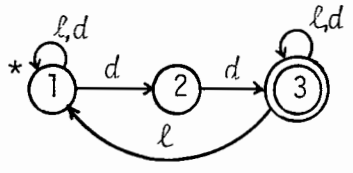
Um AFND atua de maneira similar a um AFD, porém se uma determinada transição $\delta(q, a) = \{q_i, q_j, \ldots q_l\}$ o autômato fará transições para todos os elementos deste conjunto de estados simultaneamente.

Uma configuração de um AFND é um elemento do conjunto $(\mathcal{P}(Q) \times \Sigma^*)$, analogamente ao AFD.

Uma transição de um AFND é uma relação sobre $(\mathcal{P}(Q) \times \Sigma^*) \times (\mathcal{P}(Q) \times \Sigma^*)$, definida analogamente ao que foi feito para o AFD.

Um AFND M reconhecerá uma palavra x se $(q_0, x) \vdash_M^* (A, \varepsilon)$ onde $A \in \mathcal{P}(Q)$ e $A \cap F \neq \phi$.

Ex.: M:

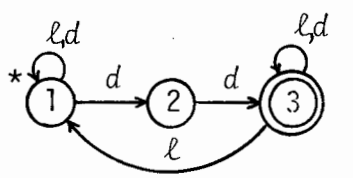


M é um AFND que aceita todas as palavras em que ocorrer dd. Contém uma indeterminação no estado 1 pois $\delta(1, d) = \{1, 2\}$ e outra no estado 3: $\delta(3, \ell) = \{1, 3\}$.

XXIII Dá-se o nome de DETERMINIZAÇÃO ao processo de se obter um AFD equivalente partindo de um AFND. Um algoritmo que efetua esta transformação pode ser encontrado em |Aho & Ullman, 72| e baseia-se no fato de que sempre é possível estabelecer-se uma correspondência bi-unívoca especial entre os estados Q' do AFD e os elementos de $\mathcal{P}(Q)$ do AFND. Portanto, AFDs, AFNDs e gramáticas regulares, são descrições equipotentes dos conjuntos regulares.

Ex.: Seja o AFND do exemplo anterior:

M_{ND}:



Vamos obter o AFD equivalente, utilizando os subscristos D e ND para diferençar transições nas máquinas determinística e não-determinística.

Os símbolos [x] onde x é uma sequência de estados do AFND indicam apenas o nome deste estado.

$1_{ND} \equiv 1_D$

$\delta_D([1], \ell) = [1]$ porque $\delta_{ND}(1, \ell) = \{1\}$

$\delta_D([1], d) = [12]$ porque $\delta_{ND}(1, d) = \{1,2\}$

$\delta_D([12], \ell) = [1]$ porque $\delta_{ND}(1, \ell) = \{1\}$

e $\delta_{ND}(2, \ell) = \phi$

$\delta_D([12], d) = [123]$ porque $\delta_{ND}(1, d) = \{1,2\}$

e $\delta_{ND}(2, d) = \{3\}$

$\delta_D([123], \ell) = [13]$ porque $\delta_{ND}(1, \ell) = \{1\}$

$\delta_{ND}(2, \ell) = \phi$

e $\delta_{ND}(3, \ell) = \{1,3\}$

$\delta_D([123], d) = [123]$ porque $\delta_{ND}(1, d) = \{1,2\}$

$\delta_{ND}(2, d) = \{3\}$

e $\delta_{ND}(3, d) = \{3\}$

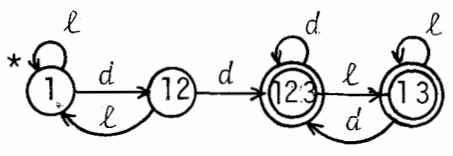
$\delta_D([13], \ell) = [13]$ porque $\delta_{ND}(1, \ell) = \{1\}$

e $\delta_{ND}(3, \ell) = \{1,3\}$

$\delta_D([13], d) = [123]$ porque $\delta_{ND}(1, d) = \{1,2\}$

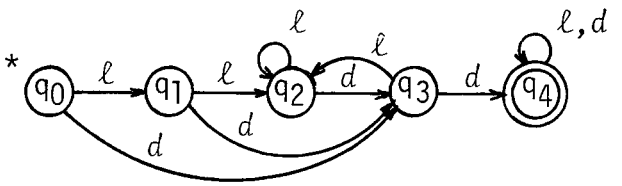
$\delta_{ND}(3, d) = \{3\}$

M_D:



AFD equivalente ao AFND do exemplo anterior.

XXIV Considere o AFD de 5 estados abaixo:



Ele reconhece o mesmo conjunto regular que o AFD obtido no exemplo anterior, de somente 4 estados. E existe ainda um outro AFD de apenas 3 estados que é equivalente a qualquer um dos dois.

Dá-se o nome de MINIMIZAÇÃO ao processo de, dado em AFD qualquer, obter-se o AFD equivalente com o menor número de estados possível, o automato mínimo, que é único, a menos de uma re-nomeação de seus estados.

XXV Diz-se que a palavra ω DISTINGUE o estado q_i do estado q_j do AFD M se

$(q_i,\omega) \vdash_M^* (q_k,\varepsilon)$ e $(q_j,\omega) \vdash_M^* (q_\ell,\varepsilon)$ e (ou $q_k \in F$ ou $q_\ell \in F$).

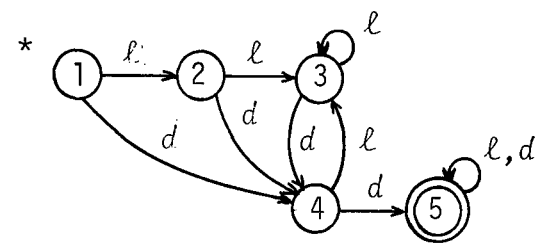
XXVI Para minimizar o AFD M normalmente são identificados todos os grupos de estados que são distinguíveis por alguma palavra ω de entrada: cada grupo cujos membros não puderem ser distinguidos por nenhuma palavra de entrada forma uma CLASSE DE EQUIVALÊNCIA de estados e seus componentes podem ser substituidos por um único estado, implicando em que o número de estados do AFD mínimo é o número de classes de equivalência de qualquer AFD equivalente.

XXVII Um algoritmo para determinar as classes de equivalência de um AFD é encontrado em |Hopcroft-Ullman, 79|.

Ex.: Seja o AFD citado acima, dado pela sua tabela de transições, renomeando seus estados:

M =

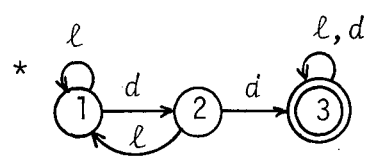
	ℓ	d	
*1	2	4	
2	3	4	
3	3	4	
4	3	5	
5	5	5	F



M' é o AFD mínimo equivalente a M, obtido pelo algoritmo de minimização:

M' =

	ℓ	d	
*1	1	2	
2	1	3	
3	3	3	F



2.3 - Expressões Regulares

XXVIII Os conjuntos regulares podem ainda ser descritos por EXPRESSÕES REGULARES. Seja Σ um alfabeto. Então as expressões regulares sobre Σ são definidas:

a) ϕ é uma expressão regular e denota o conjunto vazio;

b) ε é uma expressão regular e denota $\{\varepsilon\}$;

c) para cada a Σ , a é uma expressão regular denotando $\{a\}$;

d) se r e s são expressões regulares denotando os conjuntos R e S, respectivamente, então:

   i) (r|s) é uma expressão regular denotando R S;

   ii) (rs) ou (r.s) é uma expressão regular denotando RS = {xy | x R e y S};

   iii) (r*) é uma expressão regular e denota R*.

e) nada mais é expressão regular.

f) se p e q são expressões regulares, são admitidas , ainda, as abreviaturas:

   i) p^{+} significando (p.(p)*)

   ii) $p^{?}$ significando $(p|\varepsilon)$

   iii) $p^{£}q$ significando (p.((q.p))*)

Alguns parênteses (reduntantes) podem ser omitidos se assumirmos a precedência dos operadores "*" sobre "." e "." sobre "|".

Nota: No que se segue, suporemos que ε e $\emptyset$ só serão usadas para denotar os conjuntos $\{\varepsilon\}$ e $\emptyset$, isto é só serão usadas isoladamente. Como ε e $\emptyset$ são casos triviais, não serão considerados.

XXIX Apresentamos a seguir um algoritmo de transformação de uma expressão regular na forma de árvore binária costurada |KNUTH, 68| em AFD na forma de sua tabela de transições |Simone - Pereira, 80|. É descrito com bastante detalhe porque é a base do nosso algoritmo de construção de automatos finitos com contadores, objeto de próximo capítulo.

Baseado no algoritmo construtor de itens LR(0), este algoritmo tem duas sub-rotinas recursivas para 'caminhar' na árvore e um programa principal que realiza as chamadas das subrotinas para a montagem da tabela do automato.

Será apresentado como uma subrotina de nome 'CONSTROI-AFD', forma adotada pelos autores em seu trabalho.

XXX Uma expressão regular na forma de árvore binária costurada tem seus operadores situados nos nós internos enquanto seus símbolos ocupam as folhas, as costuras apontam para o sucessor em ordem infixa e na sua geração deve ser obedecida a disposição:

pq → <.<p> <q>>

p* → <*<p>>

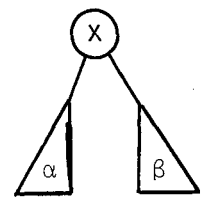
p|q → <| <p> <q>>

$p^{?}$ → <?<p>>

p^{+} → <.<p><*<p>>>

$p^{£}q$ → <.<p><*<.<q><p>>>>

notação: <X<α><β>>



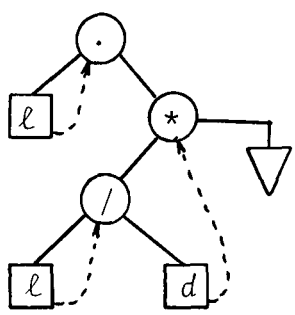
X = elemento raiz

α = sub-árvore esquerda

β = sub-árvore direita

Note que na árvore já estão substituidas as abreviaturas "+" e "£", porém a abreviatura "?" permanece.

Ex.:

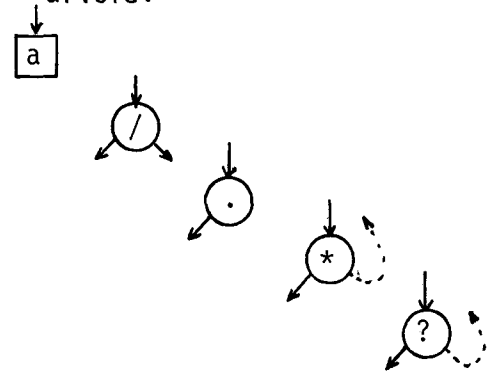


Árvore binária costurada corresponde à expressão regular do exemplo anterior: $\ell.(\ell|d)^*$, onde as folhas, contendo um símbolo, são representadas por quadrados; os nós internos, contendo operadores, são representados por círculos e o triângulo representa o delimitador

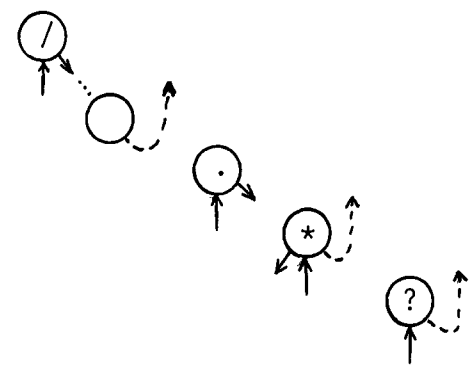
do fim da expressão, que é tratado como um símbolo. O algoritmo, como já foi dito, baseado no construtor de itens LR(0), determina, sobre a expressão regular, que símbolos podem ser aceitos a uma certa altura da análise, ou seja, que folhas estão ativas, naquele momento. A aceitação de um determinado símbolo é simulada pelo 'avanço' em todas as folhas ativas que contêm tal símbolo, buscando seus sucessores e gerando uma nova configuração de folhas ativas.

XXXI Os procedimentos recursivos SUBIR e DESCER do algoritmo, simplesmente caminham pela árvore, de acordo com as regras abaixo, (onde o já consagrado símbolo "." foi substituido por "∇" apenas em nome da legibilidade).

Especificação das regras da subrotina DESCER.

Expressão regular na forma infixa:	Expressão regular na árvore:
∇(a) → (∇a)	
∇(a|b) → (∇a|∇b)	
∇(a.b) → (∇a.b)	
∇(a*) → (∇a*)∇	
∇(a?) → (∇a?)∇	

Especificação das regras da subrotina SUBIR

Expressão regular na forma infixa:	Expressão regular na árvore:
$(a\nabla \mid b \mid \ldots) \rightarrow (a \mid b \mid \ldots)\nabla$	
$(a\nabla . b) \rightarrow (a . \nabla b)$	
$(a\nabla^{*}) \rightarrow (\nabla a^{*})\nabla$	
$(a\nabla^{?}) \rightarrow (a^{?})\nabla$	

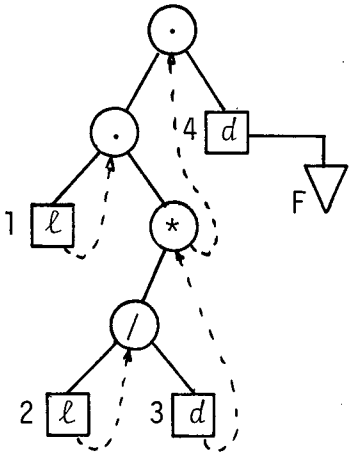
XXXII O algoritmo inicia realizando um DESCER (RAIZ), gerando a primeira configuração de folhas ativas, a definição do estado inicial. A partir dela e enquanto houverem configurações a serem investigadas, o algoritmo simula a aceitação de cada símbolo do alfabeto, criando novas configurações de folhas ativas, que correspondem a novos estados. Se um determinado estado q não possui transição com a então $\delta(q, a) = -$. A cada nova configuração de folhas ativas gerada, o algoritmo verifica se ela já existe; se já existe, simplesmente põe o seu nome na posição da tabela de transições correspondente ao estado (configuração de folhas ativas) sob investigação com tal símbolo do alfabeto como entrada; se não existe, põe a nova configuração de folhas ativas na tabela, para posterior investigação e preenche com o seu nome a posição adequada da tabela de transições.

O algoritmo termina quando não há mais configurações de folhas ativas a serem investigadas e os estados fi

nais são aqueles que tem na sua configuração de estado a folha que contém o símbolo 'fim de expressão'.

Ex.: Seja a expressão regular $\ell.(\ell \mid d)*.d$

A árvore binária costurada para a expressão com suas folhas numeradas e com 'F' representando a folha que contém o símbolo 'fim de expressão':



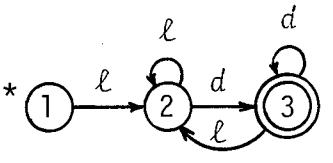
A tabela do automato finito determinístico gerada pelo algoritmo e seu diagrama de transições:

DEFEST

	ℓ	d
1	2	-
2	2	3
3	2	3

(FOLHAS ATIVAS)

1
2 3 4
2 3 4 F → FINAL



O algoritmo percorre todos os pontos da tabela de transições uma única vez, visitando um subconjunto de nós da árvore a cada vez. Sua complexidade de tempo é portanto: $\#(Q) X \#(\Sigma) X \#(T)$ onde $\#(T)$ é o número de nós da árvore.

Como $\#(\Sigma) \leq C_1 \#(Q)$ e $\#T \leq C_2 \#(Q)$, o algoritmo será $\mathcal{O}((\#(Q))^3)$. Como $\#(Q) \leq 2^{(f+1)}$, onde f é o número de folhas (símbolos) da árvore (expressão regular), o algoritmo será $\mathcal{O}(2^{3f})$ em complexidade de tempo. Analogamente, espaço ocupado será $(\#(Q) X (\#(\Sigma) + (f + 1))$ e sua complexidade de espaço será $\mathcal{O}(f \cdot 2^{2f})$.

```
XXXIII  (* CONSTROIAFD                                          *)
        (*   CONTEXTO                                          *)
        (* CONST  M = # (NOS);                                 *)
        (*        E = # (ESTADOS);                             *)
        (*        V = # (VOCABULARIO);                         *)
        (*        M1 = M + 1;                                  *)
        (* TYPE  PNODE = ↑ NODE;                               *)
        (*       NODE  = RECORD                                *)
        (*                 CODE: (CONC,ALT,FECH,OPT,SIMB);     *)
        (*                 SYMB:  0 .. V;                      *)
        (*                 COSTURA: BOOLEAN;                   *)
        (*                 RLINK,                              *)
        (*                 LLINK: PNODE;                       *)
        (*                 NUMB:  1 .. M;                      *)
        (*                 END;                                *)
```

```
(*       TREE = ARRAY [1 .. M] OF NODE;                    *)
(*       ESTADO = 0 .. E;                                  *)
(* VAR AUTOMATO: ARRAY [1 .. E, 0 .. V] OF 0 .. E;        *)
(*     FINAL: SET OF ESTADO;                               *)
(*     ENTRY: ESTADO;                                      *)
(*                                FIM CONTEXTO             *)

PROCEDURE CONSTROIAFD (ARV: TREE; ROOT: PNODE);
(* DEFINIÇÃO ESTADO. M1 IN DEFEST = FINAL                  *)
VAR DEFEST: ARRAY [0 .. E] OF SET OF 1 .. ML;
    I,J: 0 .. E;  (* PONTEIROS PARA DEFEST                 *)
    X: 1 .. V;              (* PONTEIROS P/ VARREDURA      *)
    Y: 1 .. M;              (* DO AUTOMATO                 *)
    VISITADO:  SET OF ESTADO;

FUNCTION EXISTE: ESTADO;                                    *)
(* VERIFICA SE EXISTE K TAL QUE                            *)
(*             DEFEST[K] = DEFEST[J]                       *)
(*   SE K < > J ENTÃO DEFEST[J] := [ ]                     *)
(*              SENÃO J:= J + 1;                           *)
(*   EXISTE:= K                                            *)

PROCEDURE SUBIR (Z: PNODE); FORWARD;

PROCEDURE DESCER (Z: PNODE);
BEGIN
IF Z NOT IN VISITADO
THEN BEGIN
     VISITADO: = VISITADO + [Z];
```

```
        WITH Z↑ DO
            CASE CODE OF
             FECH, OPT: BEGIN
                        DESCER(LLINK); SUBIR(RLINK)
                        END;
                  CONC:  DESCER(LLINK);
                    ALT: BEGIN
                         DESCER(LLINK); DESCER(RLINK)
                         END;
                 SIMB: DEFEST[J]:= DEFEST[J] + [NUMB];
            END
        END;
END       (* DESCER *)

PROCEDURE SUBIR;
BEGIN
IF Z = NIL
THEN DEFEST[J]:= DEFEST[J] + [M1]
ELSE WITH Z↑ DO
        CASE CODE OF
             FECH: BEGIN
                   DESCER(LLINK); SUBIR(RLINK)
                   END;
              OPT: SUBIR(RLINK);
             CONC: DESCER(RLINK);
              ALT: BEGIN
                   WHILE NOT COSTURA DO Z:= RLINK;
                   SUBIR(RLINK)
```

```
                END;
    END
END; (* SUBIR *)

BEGIN          (* ESTADO 0 = ERRO; 0 NOT IN FINAL  *)
J:= 1;                    (* J=PONTEIRO P/NOVO ESTADO*)
VISITADO:= [ ];
DESCER(ROOT);
ENTRY:= J;                              (* ESTADO INICIAL*)
I:= J;                (* I= PONTEIRO P/ESTADO VARRIDO*)
J:= J + 1;
WHILE I < J DO
    BEGIN
    FOR X: 1 TO V DO              (* VARRE AUTOMATO *)
        BEGIN
        FOR Y:= 1 TO M DO
            IF (Y IN DEFEST [I])
                AND (ARV[Y]. SYMB= X)
            THEN BEGIN
                VISITADO:= [ ];
                SUBIR(ARV[Y].RLINK); (* SHIFT *)
                END;
        AUTOMATO[I,X]:= EXISTE;
        END;
    I:= I + 1;
    END;
```

```
FOR J:= 1 TO I - 1 DO
     IF M1 IN DEFEST[J] THEN FINAL:= FINAL + [J];
END;     (* CONSTROIADFD *)
```

XXXIV Temos agora que gramática regulares, AFDs, AFNDs, e expressões regulares são descrições finitas equipotentes de conjuntos regulares.

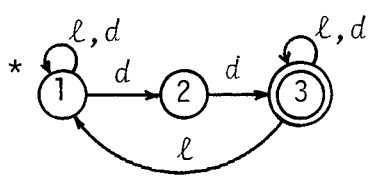
Ex.: As descrições abaixo são equipotentes:

a) gramática regular:

P = { 1: A → ℓA
2: A → dA
3: A → dB
4: B → d
5: B → dC
6: C → ℓA
7: C → ℓC
8: C → ℓ
9: C → dC
10: C → d }

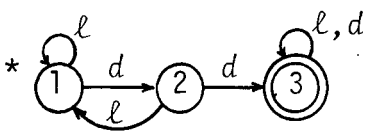
b) AFND:

		ℓ	d	
*	1	1	1,2	
	2	-	3	
	3	1,3	3	F



c) AFD:

		ℓ	d	
*	1	1	2	
	2	1	3	
	3	3	3	F



d) expressão regular:

$\ell^*.(d.\ell)^*\ .dd.\ (\ell|d)^*$

CAPÍTULO 3 - AUTOMATO FINITO COM CONTADORES

3.1 - Preliminares

I Denominam-se EXPRESSÕES REGULARES ESTENDIDAS as expressões regulares em que o fechamento, o operador * (que denota "zero ou mais ocorrências consecutivas" da expressão a que ele se refere) aparece seguido de um limite máximo de ocorrência, n, passando a significar "de zero até n ocorrências consecutivas" da expressão por ele operada. Expressões regulares estendidas são particularmente adequadas para especificação de conjuntos regulares em linguagens de programação, onde os tamanhos das palavras, que são parâmetros de projeto dos compiladores, devem ser finitos.

Ex.: $\ell.(\ell|d)*^{5}$

Expressão regular estendida que descreve identificadores em algumas implementações do FORTRAN, se ℓ for entendido como qualquer letra e d como qualquer digito.

Da mesma maneira, +n significará, em uma expressão regular estendida, "de uma até n ocorrências" da expressão por ele coberta e será uma abreviatura de $r.r*^{n-1}$, se r for tal expressão. $r*^{0}$ é equivalente a ε.

Formalmente, se Σ é um alfabeto, as expressões regulares estendidas sobre Σ são definidas:

a) se r é uma expressão regular, então r é uma expressão regular estendida, denotando R.

b) se r é uma expressão regular estendida, então:

   i) $r*^{n}$ é uma expressão regular estendida, denotando $\{\varepsilon\} \cup R \cup RR \cup \ldots \cup R^{n}$, onde R^{n} é uma sequência de n R's.

   ii) r^{+n} é uma expressão regular estendida, denotando $R \cup RR \cup \ldots \cup R^{n}$.

c) se r e s são expressões regulares estendidas, denotando os conjuntos R e S, respectivamente, então:

   i) $(r \cap s)$ é uma expressão regular estendida denotando $R \cap S$

   ii) $(r - s)$ é uma expressão regular estendida denotando $R - S$

d) nada mais é expressão regular estendida

e) se p e q são expressões regulares estendidas, é admitida, ainda, a abreviação $p^{£n}q$ siginificando $(p\ ((q.p))^{*n-1})$.

Para fins de construção de reconhecedores de conjuntos descritos por expressões regulares estendidas não consideraremos expressões definidas em c, dos tipos $r \cap s$ e $r - s$; tais conjuntos podem ser reconhecidos submetendo os sequencial e adequadamente aos reconhecedores de R e S.

II Normalmente, o automato finito que reconhece o conjunto descrito por uma expressão regular estendida do tipo r^{*n} é composto por n cópias do automato finito que reconhece R e mais um estado inicial; se r for uma expressão complexa, teremos um automato finito de tamanho considerável.

Construir uma máquina alternativa que, com auxílio de de contadores, reduza o número de estados em comparação ao AFD usual é a proposta fundamental deste trabalho.

3.2 - Automatos Finitos com Contadores (AFC)

III Informalmente, são as seguintes as CARACTERÍSTICAS DA MÁQUINA:

a) a cada operador *n está associado um contador;

b) cada contador é inicializado (sob controle da máquina) quando for iniciado o reconhecimento da expressão coberta pelo operador *n associado a esse contador;

c) cada reconhecimento completo da sub-expressão r coberta por um operador *n implica numa decrementação do seu contador;

d) terá comportamentos distintos caso algum contador atinja seu limite ou não;

e) cada estado tem duas informações associadas:

i) uma que guarda semelhança a um estado de um AFD usual;

ii) outra que será associada ao conjunto de valores

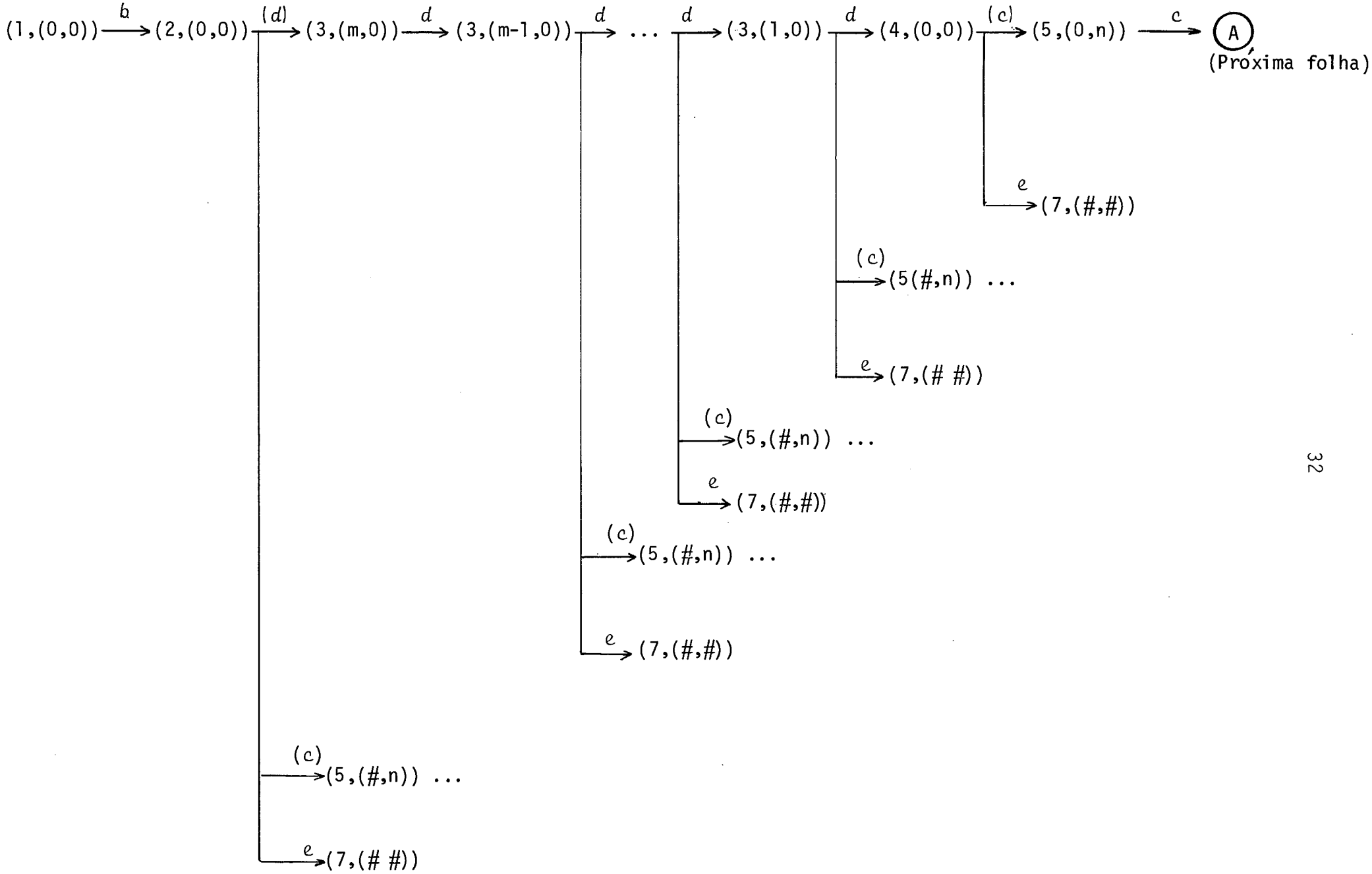
dos contadores no momento;

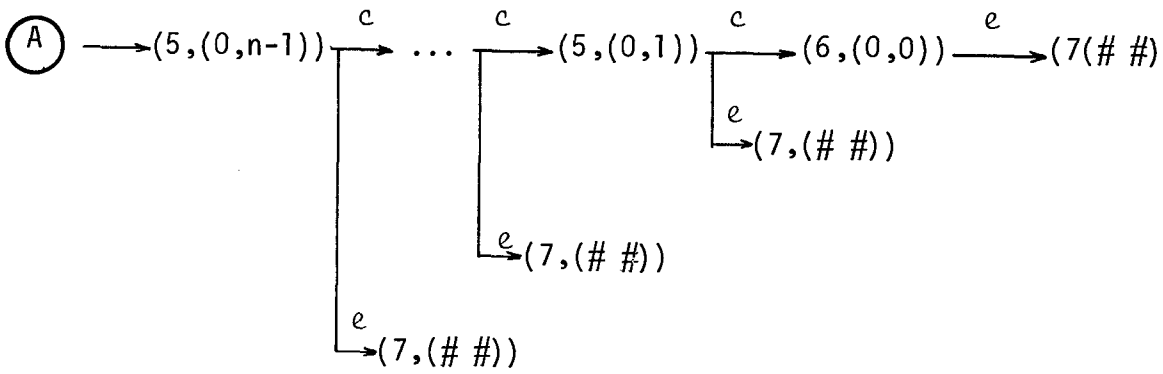
f) podem existir transições especiais, sem avanço na entrada, no caso de (re) inicializações de constadores ou (possivelmente) decremento de contadores;

g) será uma máquina determinística (quanto aos contadores), se a expressão regular estendida a partir da qual foi construída não contiver opção entre sub-expressões com mesmo prefixo envolvendo contagem, por exemplo, no caso de $((\alpha.\beta)^{*m}\ \gamma)|((\alpha.\delta)^{*n}.\theta)$, com $\alpha \neq \varepsilon$. (Ver também 3.3 XV).

IV EXEMPLO 1: $b.d^{*m}.c^{*n}.e$

Vamos supor que o automato finito com contadores encontra-se no estado 1 inicial e com a configuração de contadores $n_1 = n_2 = 0$ correspondendo aos valores iniciais dos contadores. Representamos este estado (1,(0,0)). Segue o grafo das transições do AFC:


(1,(0,0))
b
(2,(0,0))
(d)
(3,(m,0))
d
(3,(m-1,0))
d
...
d
(3,(1,0))
d
(4,(0,0))
(c)
(5,(0,n))
c
A
(Próxima folha)
e
(7,(#,#))
(c)
(5(#,n)) ...
e
(7,(# #))
(c)
(5,(#,n)) ...
e
(7,(#,#))
(c)
(5,(#,n)) ...
e
(7,(#,#))
(c)
(5,(#,n)) ...
e
(7,(# #))



onde (α) representam transições em que o símbolo α está na entrada e não é feito avanço (SCAN)

e # representa um valor de contador irrelevante a esta altura do processo.

Note que os estados são distintos quando o restante da palavra a entrar é distinto. No caso:

Estado 1: estado inicial, aceita um *b*;

2: aceita *d*'s, contando-os;

3: aceita *c*'s, contando-os;

4: aceita *e*;

5: estado final, não aceita mais nada.

Com isto, eliminamos m+n-2 estados que são distinguidos pelo conjunto de valores dos contadores.

V EXEMPLO 2: $(a^{*m})^{*n}$

O AFC tem o seguinte grafo de transições:

$$(1,(0,0)) \xrightarrow{(a)} (2,(m,n)) \xrightarrow{a} (2,(m-1,n)) \xrightarrow{a} \ldots \xrightarrow{a}$$
$$(2,(1,n)) \xrightarrow{a} (2,(m,n-1)) \xrightarrow{a} (2,(m-1,n-1)) \xrightarrow{a} \ldots \xrightarrow{a}$$
$$(2,(1,n-1)) \xrightarrow{a} \ldots \xrightarrow{a} (2,(1,1)) \xrightarrow{a} (3,(\#,0))$$

A cada vez que o contador de *m atinge sem limite, o contador de *n (externo) é decrementado e o interno é reinicializado, voltando a contagem a ser feita no contador interno (*m). O estado 3 é final.

VI COMPARAÇÃO INFORMAL entre AFD e AFC

Apresentamos a seguir o AFD e o AFC para algumas expressões regulares estendidas.

VII EXEMPLO 3: $b.d^{*10}.c^{*15}.e$

O AFD usual terá 1 + 1 + 10 + 15 + 1 = 28 estados, sendo 1 inicial, 1 para reconhecer b, 10 para reconhecer as 10 possíveis ocorrências de d, 15 para as 15 possíveis ocorrências de c e 1, final, reconhecendo e.

O AFC, como já vimos no exemplo 1, terá apenas 7 estados - teria os mesmos 7 estados mesmo se a expressão a ser reconhecida fosse $b.d^{*100}.c^{*150}.e$: mudariam apenas os valores limites para os dois contadores - e terá também um único estado final.

Pode-se estabelecer as seguintes equivalências entre os estados do AFD usual e do AFC para este exemplo:

AFD	AFC	
1	1	estado inicial; aceita b
-	2	inicializa contador de *10
2-11	3	conta as 10 possíveis ocorrências de d
-	4	inicializa contador de *15
12-26	5	conta as 15 possíveis ocorrências de c
27	6	aceita e
28	7	estado final; não aceita mais nada.

VIII EXEMPLO 4: $(a^{*5})^{*3}$

AFD usual com $1 + (5 \times 3) = 16$ estados, todos finais.

AFC com 3 estados (exemplo 2), todos finais.

AFD	AFC	
-	1	estado inicial; inicializa os contadores de ^{*5} e ^{*3}.
1-15	2	conta as 3 ocorrências (externas) das 5 ocorrências (internas) de a; conta também as 5 ocorrências internas de a; reinicializada o contador de ^{*5}.
16	3	não aceita mais nada.

IX AFC Determinístico - Definição Formal.

Um AFC é uma 9-tupla:

$M_C = (Q, \Sigma, S, V, C, \delta, \theta, q_0, F)$

onde Q é um conjunto finito de estados;

Σ é um alfabeto finito de entrada;

S é um conjunto finito de ações semânticas; a cada contador corresponde uma ação semântica e $Q \cap S = \phi$

V é uma p-tupla de valores atuais de contadores assumindo valores em $\{0,1,2,\ldots,m\}^p$.

C é uma função valor limite

$C: S \rightarrow \{1,2,\ldots,m\}$

onde C(i) representa o valor limite do contador;

δ é um mapeamento parcial de transições

$\delta: (Q \cup S) \times (\Sigma \cup \{N,E\}) \rightarrow (Q \cup S \cup \{-\})$

onde para $q \in Q$ e $a \in \Sigma$

$\delta(q,a)$ representa a transição do estado q com entrada a;

para $i \in S$

$\delta(i,N)$ representa a transição do contador i, se $v_i \neq 0$;

$\delta(i,E)$ representa a transição do contador i, se $v_i = 0$;

"-" indica que não há transição

θ é a função de controle do avanço sobre a entrada, com ou sem inicialização de contadores

$\theta: (Q \times \Sigma) \cup (S \times \{N,E\}) \rightarrow \{"R","\uparrow"," "\}$

representando

"R" reinicializar o contador e transição sem avançar a entrada;

"$\uparrow$" transição sem avançar a entrada;

" " transição avançando a entrada

q_0 é o estado inicial

$F \subseteq Q$, é o conjunto de estados finais.

X Uma CONFIGURAÇÃO (π) de um AFC é uma tripla

$\pi = (q,V,w)$

onde $q \in Q$, estado atual

$V \in \{0,1,2,\ldots,m\}^p$, p-tupla de valores atuais dos contadores

$w \in \Sigma^*$, entrada restante.

A configuração $\pi_0 = (q_0,(0,0,\ldots,0),w)$ é dita inicial.

As configurações $\pi_j = (q_j, V, \varepsilon)$ tais que $q_j \in F$ são ditas finais.

XI Uma TRANSIÇÃO sobre as configurações $\pi_i \underset{M_c}{\vdash\!\!\!-\!\!\!-} \pi_{i+1}$ é uma relação binária definida como:

Seja $\pi_i = (q, (v_1, v_2, \ldots, v_p), aw)$

$\pi_{i+1} = (q', (v'_1, v'_2, \ldots, v'_p), x)$

Faça $v'_i \leftarrow v_i$, para $1 \leq i \leq p$

$x \leftarrow aw$

$q' \leftarrow q$

$j \leftarrow a$

faça $i \leftarrow \delta(q', j)$

se $\theta(q', j) =$ "R"

então $v'_i \leftarrow C(i)$

$j \leftarrow N$

senão se $i \in S$

então se $\theta(q', j) =$ " " então $x \leftarrow w$

$v'_i \leftarrow v'_i - 1$

se $v'_i \neq 0$ então $j \leftarrow N$

senão $j \leftarrow E$

senão se $q' \in Q$ então $x \leftarrow w$

$q' \leftarrow i$

até que $q' \in Q$

Informalmente, são as seguintes as regras a serem observadas em uma transição:

Seja $\pi_i = (q, (v_1, v_2, \ldots, v_p), aw)$

Seja $q' \in Q$ e $i, i' \in S$

1) se $\theta(q,a) =$ " ", $\delta(q,a)$ será q' ou i: transição com avanço de entrada.
2) se $\theta(q,a) =$ "↑" , $\delta(q,a)$ será i: transição sem avanço de entrada.
3) se $\theta(q,a) =$ "R" ou $\theta(i',N) =$ "R" ou $\theta(i',E) =$ "R", $\delta(q,a)$ ou $\delta(i',N)$ ou $\delta(i',E)$ será, em qualquer caso, i: (re) inicialização do contador i sem avanço de entrada; o próximo estado será indicado por $\delta(i,N)$.
4) se $\theta(i,N) =$ " " ou $\theta(i,E) =$ " ", $\delta(i,N)$ será q' ou $\delta(i,E)$ será q' ou i'; não há avanço de entrada.
5) se $\theta(i,N) =$ "R" ou $\theta(i,E) =$ "R", $\delta(i,N)$ ou $\delta(i,E)$ será, em ambos os casos, i'; não há avanço de entrada.
6) se $\delta(q,a) = q'$: tansição normal, o próximo estado será q'.
7) se $\delta(q,a) = i$ ou $\delta(i',E) = i$, transição com contagem: decrementar o contador i; se $v'_i \neq 0$ o próximo estado será indicado por $\delta(i,N)$; se $v'_i = 0$ o próximo estado será indicado por $\delta(i,E)$.
8) se $\delta(i,N) = q'$ ou $\delta(i,E) = q'$, o próximo estado será q'.

XII EXEMPLO 5: a^{*3}

$M_c = (Q,\Sigma,S,V,C,\delta,\theta,q_0,F)$

onde $Q = \{1,2,3\}$

$\Sigma = \{a\}$

$S = \{4\}$

$V = (0)$ inicialmente

C: $4 \rightarrow 3$

δ:

$Q \cup S$ \ $\Sigma \cup \{N,E\}$	a	N	E
1	4		
2	4		
3	-		
4		2	3

θ:

Q \ Σ	a
1	R
2	
3	

$\cup$

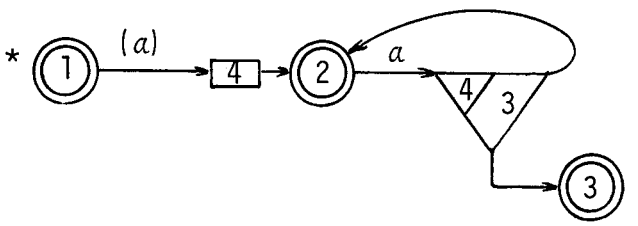
S \	N	E
4		

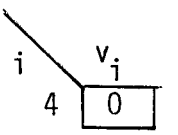
$q_0 = 1$

$F = \{1,2,3\}$

Observação: δ deve ser implementada como duas tabelas disitintas (Q x Σ) e (S x {N,E}), pois as áreas hachuradas não tem significado e nunca são utilizadas. Optamos por esta forma expandida, apenas para facilitar o formalismo.

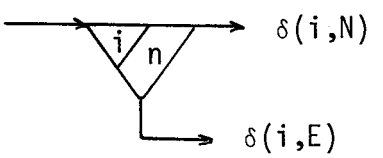
Representação Gráfica





Legenda:

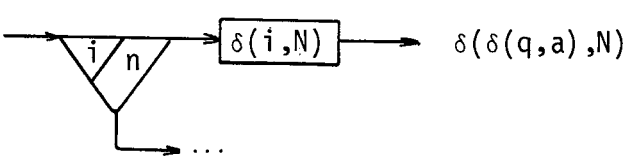
Q : (q)

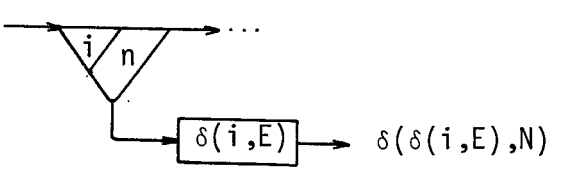
S : 

onde i é o nº do contador

n = C(i) é o limite do contador

$\theta(q,a)$ = "R": (q) → $\delta(q,a)$ → $\delta(\delta(q,a),N)$

$\theta(i,N)$ = "R": 

$\theta(i,E)$ = "R": 

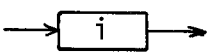
$\theta(q,a)$ = "↑": (q) —(a)→ $\delta(q,a)$

q_0 : *(q_0)

$q_j \in F$: (q_j)

Observações:

1) $\xrightarrow{(a)}$: transição sem avanço de entrada

2)  : inicialização do contador i

XIII EXEMPLO 6: $(a^{*2}.\ a^{*3})^{*4}$

$M_c = (Q, \Sigma, S, V, C, \delta, \theta, q_0, F)$

onde $Q = \{1,2,3,4\}$

$\Sigma = \{a\}$

$S = \{5,6,7\}$

$V = (0,0,0)$ inicialmente

$C =$

i	C(i)
5	4
6	2
7	3

δ

Q \ Σ	a
1	5
2	-
3	6
4	7

S	N	E
5	6	2
6	3	7
7	4	5

θ :

Q \ Σ	a
1	R
2	
3	
4	

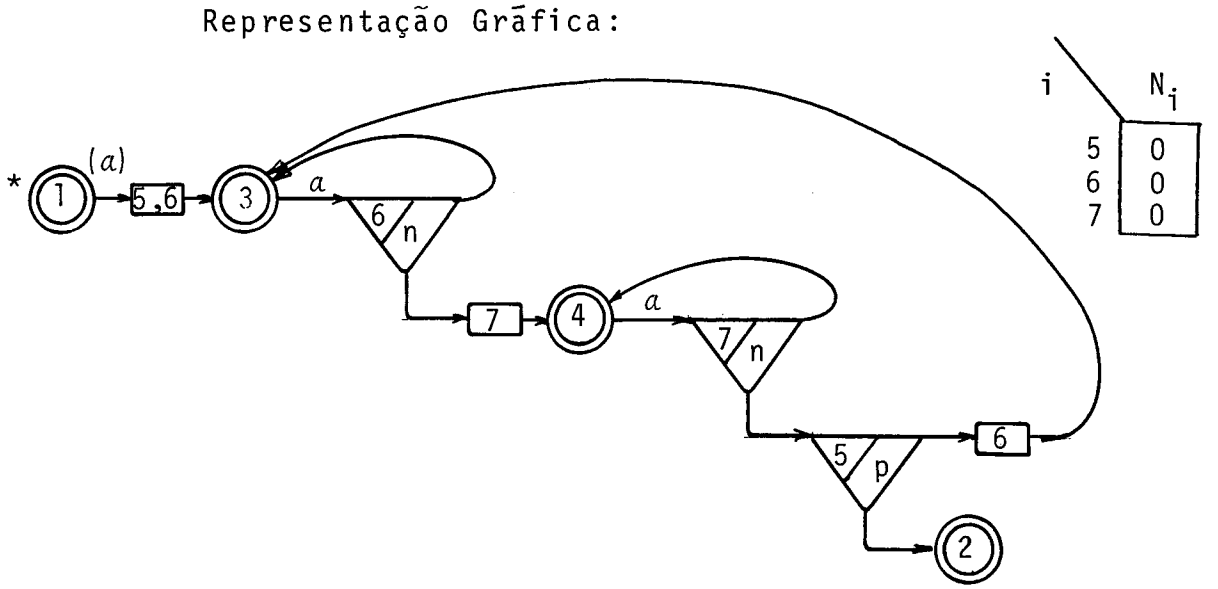
S	N	E
5	R	
6		R
7		

$q_0 = 1$

$F = \{1,2,3,4\}$

Observação: δ na forma de duas tabelas. Ver observação do exemplo anterior.

Representação Gráfica:



XIV EXEMPLO 7: $(a^{*2} \mid b^{*3})^{*4}$

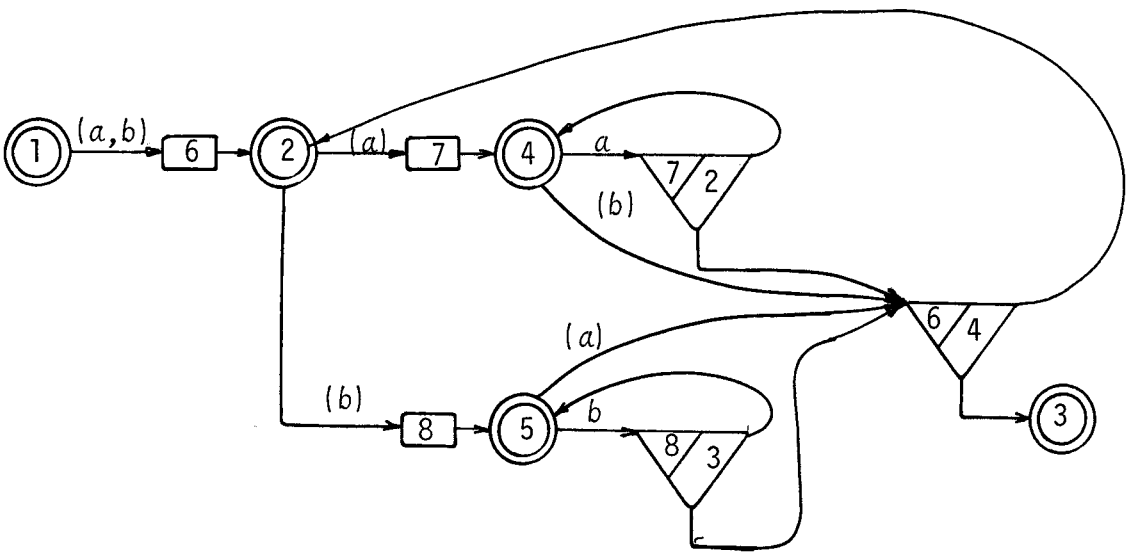
$M_c = (Q, \Sigma, S, V, C, \delta, \theta, F)$

Q \ Σ	a	b	
→ 1	R 6	R 6	F
2	R 7	R 8	F
3	-	-	F
4	6	↑ 6	F
5	↑ 6	8	F

S	N	E	C	V
6	2	3	4	0
7	4	6	2	0
8	5	6	3	0

Observação: adotamos aqui uma representação compactada, na forma de duas tabelas, agrupando todas as informações do AFC. θ e δ foram sobrepostas, uma vez que seus contra-domínios são disjuntos; q_0 é assinalado pela seta horizontal (à esquerda); q_j F é assinalado por "F" (à direita). Esta será a forma de representação tabular de AFC adotada doravante neste trabalho.

Representação Gráfica:



Reconhecendo a palavra *aababbb*:

$(1,(0,0,0), aababbb) \underset{M_c}{\vdash\!\!-\!\!-} (2,(4,0,0), aababbb) \underset{M_c}{\vdash\!\!-\!\!-}$

$(4,(4,2,0), aababbb) \underset{M_c}{\vdash\!\!-\!\!-} (4,(4,1,0), ababbb) \underset{M_c}{\vdash\!\!-\!\!-}$

$(2,(3,0,0), babbb) \underset{M_c}{\vdash\!\!-\!\!-} (5,(3,0,3), babbb) \underset{M_c}{\vdash\!\!-\!\!-}$

$(5,(3,0,2), abbb) \underset{M_c}{\vdash\!\!-\!\!-} (2,(2,0,2), abbb) \underset{M_c}{\vdash\!\!-\!\!-}$

$(4,(2,2,2), abbb) \underset{M_c}{\vdash\!\!-\!\!-} (4,(2,1,2), bbb) \underset{M_c}{\vdash\!\!-\!\!-}$

$(2,(1,0,2), bbb) \underset{M_c}{\vdash\!\!-\!\!-} (5,(1,0,3), bbb) \underset{M_c}{\vdash\!\!-\!\!-}$

$(5,(1,0,2), bb) \underset{M_c}{\vdash\!\!-\!\!-} (5,(1,0,1), b) \underset{M_c}{\vdash\!\!-\!\!-} (3,(0,0,0), \varepsilon)$

3 é um estado final: aceito

3.3 - Algoritmo para Construir o AFC a Partir da Expressão Regular Estendida "bem comportada" (CONSTROICUIA).

XV O algoritmo, como já foi dito anteriormente, utiliza a estrutura apresentada no algoritmo CONSTROIAFD, em |Simone-Pereira, 80|. Para tratar dos contadores, introduzimos modificações que atendem aos seguintes fatos:

i) quando, na ação de percorrer a árvore, atingimos por cima um nó que contém um *n, criamos um estado para inicializar seu contador;

ii) quando, de forma contrária, atingimos um nó contendo um *n por baixo, criamos uma ação semântica de decrementar seu contador.

iii) em decorrência de i e ii a definição de um estado passa a poder conter, também, contadores.

Além disso, para que o algoritmo forneça os resultados esperados e produza efetivamente um AFC na saída, é preciso que a expressão regular estendida que lhe é submetida à entrada seja "bem comportada", ou seja,

i) não contenha concatenação de sub-expressões denotando linguagens com elementos que tenham prefixos comuns em que a primeita sub-expressão seja contada, como por exemplo $(\alpha.\beta)^{*m}.(\alpha.\gamma)$ ou $(\alpha\ \beta)^{*m}.(\alpha.\gamma)^{*n}$;

ii) não contenha opção entre sub-expressões denotando linguagems com elementos que tem prefixos comuns envolvendo contagem, como por exemplo $(\alpha.\beta^{*m}|(\alpha.\gamma)^{*n}$ e $(\alpha^{*m}.\gamma)|(\alpha^{*n}.\ \theta)$.

Algumas dessas expressões, entretanto, são facilmente recuperáveis, podendo ser transformadas em "bem comportadas" como nos exemplos:

$(\alpha.\gamma)^{*n}.\ (\alpha.\theta) \rightarrow \alpha.(\gamma.\alpha)^{*n}.\theta$

$((\alpha.\gamma)^{*n}.\beta)|(\alpha.\theta) \rightarrow (\alpha.((\gamma.(\alpha.\gamma)^{*n-1}.\beta)|\theta))|\beta$

Existem expressões, entretanto, em que só é possível transformá-la em "bem comportada" se desmembrarmos os contadores, desaparecendo, portanto, a vantagem da economia de espaço do AFC sobre o AFD. Estas restrições do algoritmo são consequência dos seguintes fatos:

i) a inicialização de um contador dar-se-á quando for reconhecido o primeiro terminal da sub expressão a ser contada;

ii) o operador *n admite a possibilidade da ocorrência de zero vezes a expressão por ele coberta.

Em consequência,

i) no reconhecimento de uma palavra descrita por $(\alpha)^{*n}.\beta$, por exemplo, se o primeiro terminal de α for também o primeiro terminal de β, duas ações poderiam ser tomadas: iniciar o reconhecimento de α, inicializando-se seu contador ou saltar a sub-expressão $(\alpha)^{*n}$ (isto é, α

ocorrer zero vezes) e iniciar o reconhecimento de β. Nosso algoritmo não é capaz de decidir qual das duas ações deve ser tomada;

ii) No reconhecimento de uma palavra descrita, por $(\alpha^{*m})|(\beta^{*n})$, por exemplo, se o primeiro terminal de α for também o primeiro terminal de β, duas ações de inicialização de contador teriam que ser realizadas quando esse terminal fosse reconhecido. Nosso algoritmo só é capaz de indicar uma ação de inicialização de contador por terminal reconhecido (note-se que uma versão não-determinística, que não consideraremos aqui, não teria esse problema).

XVI Recebe como entrada, como o CONSTROIAFD, uma árvore binária costurada em cuja geração deve ser obedecida a seguinte disposição:

pq → <.<p><q>>

p|q → <|<p><q>>

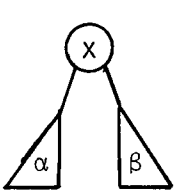
p^* → $<^*C_m<p>>$

p^+ → $<.<p><^*C_m<p>>>$

$p^{£}q$ → $<.<p><^*C_m<.<q><p>>>>$

$p^?$ → $<^*1<p>>$

notação: <x<α><β>>



x = elemento raiz

α = sub-árvore esquerda

β = sub-árvore direita

$p*^{n}$ → <*n<p>>

p^{+n} → <.<p><*n-1<p>>>

$p^{£n}q$ → <.<p><*n-1<.<q><p>>>

onde C_m é uma constante inteira adequadamente grande, que depende da implementação.

Tal disposição garante as substituição das abreviaturas existentes na expressão regular estendida além de preparar os operadores *, + e ? para sua submissão ao algoritmo construtor de AFC.

Precisaremos de duas tabelas auxiliares, FIRSTESQ definida aqui como o conjunto dos símbolos que iniciam a sub-expressão operada por um *n e FOLLOW, o conjunto de símbolos que iniciam a sub-expressão que segue um operador *n. FOLLOW pode conter, portanto, o símbolo 'fim de expressão', não pertencente à expressão, acrescentado aqui somente para fins de construção do AFC. A montagem destas tabelas é o primeiro passo do algoritmo CONSTROICUIA.

Temos ainda uma subrotina adicional, POE(I,Y,A',K') utilizada para resolver eventuais colisões no preenchimento da tabela do automato, segundo as prioridades:

I : estado, linha da tabela a ser preenchida;

Y : símbolo, coluna da tabela a ser preenchida;

A': ação a ser colocada na tabela;

K': transição a ser colocada na tabela.

TABELA DE RESOLUÇÃO DE COLISÕES

COLISÃO				PREVALECE	
A	K	A'	K'	A_p	K_p
RESETA	i	SEGURA	j	RESETA	i
NORMAL	i	SEGURA	j	NORMAL	i
SEGURA	i	SEGURA	j	SEGURA	i

onde A e K já se encontram na tabela

Os demais casos, RESETA i / RESETA j e RESETA i / NORMAL j denotam um não determinismo de contadores, não ocorrerá se forem observadas as restrições indicadas em 3.2.II-g); NORMAL i / NORMAL j nunca ocorrem, por construção.

Como regra geral, podemos dizer informalmente que RESETA e NORMAL prevalecem sobre SEGURA e entre duas ações SEGURA, prevalece a mais antiga.

Apenas para facilitar o algoritmo, estamos supondo que as folhas na árvore estão numeradas em ordem crescente 1,2,..., NFOLHAS e os nós que contem *n estão numerados a partir daí: NFOLHAS+1, NFOLHAS+2,..., NFOLHAS + NCONT.

XVII Os procedimentos recursivos SUBIR e DESCER do CONSTROICUIA 'caminham' pela árvore de acordo com as regras abaixo:

Especificação das regras da subrotina DESCER para o CONSTROICUIA:

Expresão regular na forma infixa:

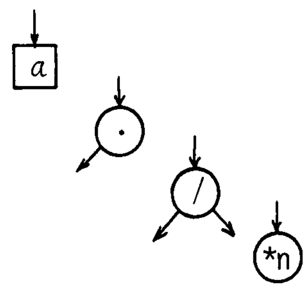
$\nabla(a) \rightarrow (\nabla a)$

$\nabla(ab) \rightarrow (\nabla ab)$

$\nabla(a|b) \rightarrow (\nabla a|\nabla b)$

$\nabla(a*^{n}) \rightarrow ((a)^{\nabla}*^{n})$

Expressão regular na árvore:



Especificação das regras da subrotina SUBIR para o CONSTROICUIA:

Expressão regular na forma infixa:

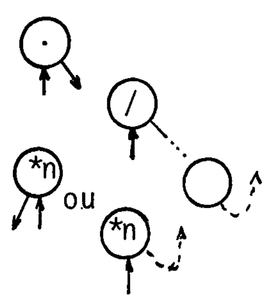
$(a\nabla.b) \rightarrow (a.\nabla b)$

$(a\nabla|b|\ldots) \rightarrow (a|b|\ \ldots)\nabla$

$(a\nabla)*^{n} \rightarrow (\nabla a)*^{n}$

$\quad\quad\quad\quad |(a)*^{n}\nabla$

Expressão regular na árvore:



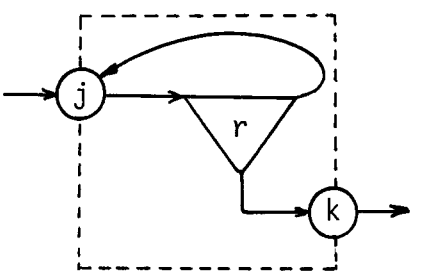
Caberá ao programa principal do CONSTROICUIA a execução da regra:

$$(a)^{\nabla}*^{n} \rightarrow (\nabla a)*^{n}$$

$$\rightarrow (a)*^{n}\nabla$$

XVIII Contadores são incorporados a AFDs segundo a filosofia de que a contagem é feita em uma máquina própria, composta por um estado de entrada, um estado de escape e um mecanismo de contagem capaz de decidir se o controle permanece na máquina de contagem ou se a contagem já terminou e o controle deve voltar ao ambiente externo à máquina de contagem.

Ex.: a máquina de contagem



Assim, a ocorrência de um contador na definição de estado de um certo estado causará a criação de uma máquina de contagem, ou seja, um estado de entrada j, um estado de saída k e um mecanismo decisório r. Esta construção é efetuada pelo programa principal no CONSTROICUIA, sendo esta a principal modificação em relação ao CONSTROIAFD.

Analogamente ao CONSTROIAFD, o CONSTROICUIA é $\mathcal{O}(2^{3f})$ em complexidade de tempo e $\mathcal{O}((f + s)\ 2^{2f} + s)$ em complexidade de espaço onde f é o número de folhas s é o número de operadores *n na árvore.

```
XIX     (*  CONSTROICUIA                                          *)
        (*  CONTEXTO                                              *)
        (* CONST NNOS    = # (NOS);                               *)
        (*       NFOLHAS = # (FOLHAS);                            *)
        (*       NCONT   = # (CONTADORES);                        *)
        (*       NSIMB   = # (VOCABULARIO);                       *)
        (*       NEST    = # (ESTADOS);                           *)
        (* TYPE  PNO     : ↑ NO;                                  *)
        (*       NO      : RECORD                                 *)
        (*                   OPERADOR : (CONC,ALT,ESTRN,SIMB)     *)
        (*                   CODIGO,                              *)
        (*                   NUMB: INTEGER;                       *)
        (*                   COSTURA: BOOLEAN;                    *)
        (*                   LLINK,  RLINK: PNO;                  *)
        (*                   END;                                 *)
        (*       ARVORE  : ARRAY [1...NNOS] OF NO;                *)
        (*       ESTADO  : 0..NEST;                               *)
        (*       SEM     : NEST+1..NEST+NCONT;                    *)
        (*       ESTOUSEM: 1..NEST+CONT;                          *)
        (* VAR   AUTOMATO: RECORD                                 *)
        (*                   DELTA: ARRAY [1..NEST, 1..NSIMB]     *)
        (*                          OF RECORD                     *)
        (*                             AÇÃO: (RESETA,SEGURA,NORMAL)*)
        (*                             TRANSIÇÃO:1..NEST+NCONT;   *)
        (*                             END;                       *)
        (*                   FINAL: SET OF ESTADO;                *)
        (*                   CONTAGEM: ARRAY [SEM]                *)
        (*                             OF RECORD                  *)
```

```
(*                          NORMAL,                          *)
(*                          ESCAPE: ESTOUSEM;                *)
(*                          LIMITE,                          *)
(*                          VALOR: INTEGER;                  *)
(*                          END;                             *)
(*               END;                                        *)
(*               FIM DO CONTEXTO                             *)
```

```
PROCEDURE CONSTROICUIA (ARV: ARVORE,ROOT:PNO);
(* DEFINICAO ESTADO. 0 EM DEFEST = FINAL                              *)
VAR DEFEST: ARRAY [ESTOUSEM] OF SER OF 0..NFOLHAS+NCONT;
    FIRSTESQ, FOLLOW: SET OF INTEGER;
    SUBINDO: BOOLEAN;
    I,P: ESTADO;
    S  : SEM;
    X,Y: INTEGER;
    T  : ESTOUSEM;
FUNCTION EXISTE: ESTOUSEM;
BEGIM
IF SUBINDO
THEN BEGIN
     SUBINDO:= FALSE;
     (*VERIFICA SE EXISTE L EM Q TAL QUE                               *)
     (*         DEFEST[L]= DEFEST[P]                                   *)
     IF L ≠ P THEN DEFEST[P]:= [ ]
              ELSE P:= P+1
     END
ELSE BEGIN
     (* ENCONTRA L EM S TAL QUE DEFEST[L] = DEFEST [P]                 *)
     DEFEST[P]:=  [ ]
     END;
EXISTE:= L;
END  (* EXISTE *)
```

```
PROCEDURE SUBIR (Z : PNO); FORWARD;

PROCEDURE DESCER (Z : PNO);
BEGIN
WITH Z↑ DO
   CASE OPERADOR OF
   CONC : DESCER (LLINK);
   ALT  : BEGIN
          DESCER(LLINK); DESCER(RLINK);
          END;
ESTRN,SIMB: DEFEST[P]:= DEFEST[P]+[NUMB];
    END CASE
END; (*DESCER*)

PROCEDURE SUBIR;
BEGIN
IF Z = NIL
THEN BEGIN
     DEFEST[P]:= DEFEST[P] + [0];
     FINAL:= FINAL  + [P];
     END
ELSE WITH Z DO
          CASE OPERADOR OF
               CONC: DESCER (RLINK);
               ALT : BEGIN
                     WHILE NOT COSTURA DO Z:= RLINK;
                     SUBIR (RLINK);
                     END;
```

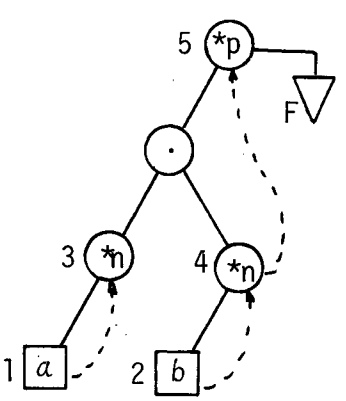
```
                ESTRN : BEGIN
                        DEFEST[P]:= DEFEST[P] +[NUMB];
                        SUBINDO:= TRUE;
                        END;
                END CASE;
END; (* SUBIR *)

PROCEDURE POE(I: ESTADO,Y: SIMB,A: ACAO,K: ESTOUSEM)
BEGIN
(* POE NA TABELA DO AUTOMATO, NA LINHA I,                    *)
(* COLUNA Y, O PAR A,K, RESOLVENDO AS                        *)
(* EVENTUAIS COLISOES                                        *)
END; (* POE *)

BEGIN (* ESTADO 0 = ERRO, 0 NOT IN FINAL                     *)
FOR X:= NFOLHAS+1 TO NFOLHAS+NCONT DO
    BEGIN
    (* MONTA FIRSTESQ[X] *)
    (* MONTA FOLLOW  [X] *)
    END;
SUBINDO:= FALSE;
P:= 1;
DESCER (ROOT);
I:= 1;
P:= 2;
S:= NEST+1;
WHILE I < P DO
   BEGIN
```

```
FOR X:= NFOLHAS+1 TO NFOLHAS+NCONT DO (* PROCURA CONTADOR *)
        IF X IN DEFEST[I]
        THEN BEGIN     (* CONTADOR EM DEFEST *)
             DESCER (ARV[X]. LLINK);        (*J*)
             J:= EXISTE;
             SUBIR (ARV[X]. RLINK);
             K:= EXISTE;
             FOR Y = I TO NFOLHAS DO
                 IF Y IN DEFEST [J]
                 THEN FOR YI = I TO NFOLHAS DO
                     IF YI IN DEFEST[J]
                     THEN IF ARV[Y]. CODIGO = ARV[YI]. CODIGO
                          THEN STOP (*EXP.REG.EST.INTRATAVEL*)
             FOR Y:= 1 TO NSIMB DO
                IF Y IN FOLLOW[X]
                THEN BEGIN
                     POE (I,Y,SEGURA,K);
                     POE (J,Y,SEGURA,K);
                     END;
             IF NSIMB+1 IN FOLLOW[X] THEN FINAL:= FINAL+[I]+[J];
             DEFEST[S]:= X;
             CONTAGEM[S]. NORMAL:= J;
             CONTAGEM[S]. ESCAPE:= K;
             CONTAGEM[S]. LIMITE:= ARV[X]. CODIGO;
             FOR Y:= 1 TO NSIMB DO
                     IF Y IN FIRSTESQ[X]
                     THEN POE (I,Y,RESETA,S)
             S:= S+1;
```

```
                END;
        FOR X:= 1 TO NSIMB DO                    (* PROCURA FOLHAS *)
            BEGIN
            FOR Y:= 1 TO NFOLHAS DO
               IF (Y IN DEFEST[I]) AND (ARV[Y]. CODIGO = X)
               THEN SUBIR (ARV[Y].RLINK);
            T:= EXISTE;
            POE (I,X,NORMAL,T);
            END;
        I:= I+1;
        END;    (* WHILE I < P *)
    END  (* CONSTROICUIA *)
```

XX. EXEMPLO 8: $(a^{*m} \cdot b^{*n})^{*p}$

A árvore:



	FIRST ESQ 1	2	F
3	a		
4		b	
5	a	b	

	FOLLOW 1	2	F
3	a	b	⊣
4	a	b	⊣
5			⊣

CONTEUDOS

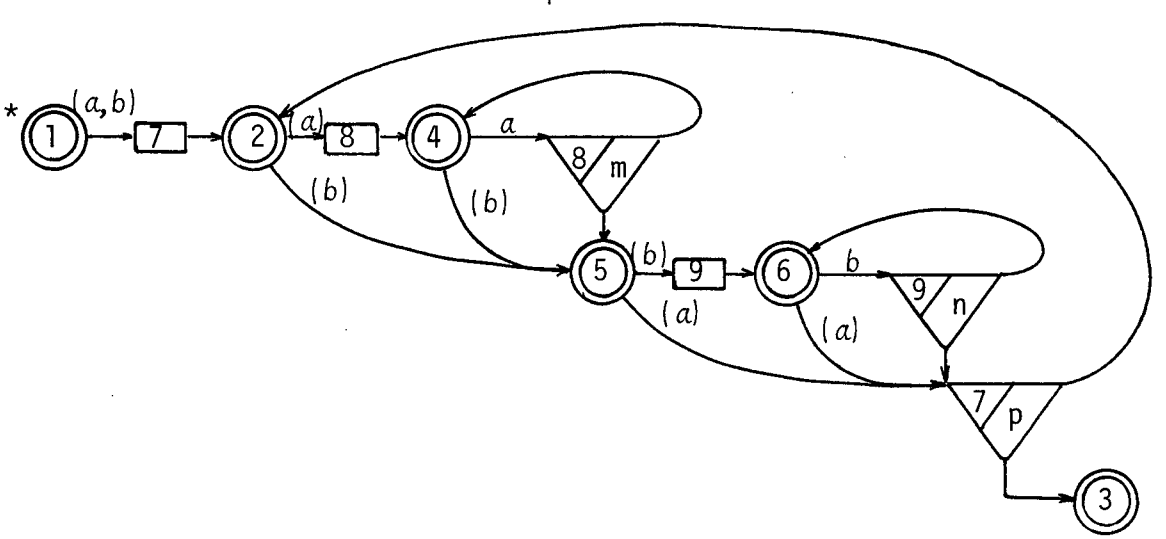
a	b	⊣	m	n	p

			a	b	DEFEST	1	2	F	3	4	5
	I →	1	R 7	R 7	F						5
I	J	2	~~5~~ R 8	↑ 5	F				3		
I	K	3	-	-	F			F			
I	J	4	~~5~~ 8	↑ 5	F	1					
I	K	5	↑ 7	~~9~~ R 9	F					4	
I	J	6	↑ 7	~~9~~ 9	F		2				

			N	E	C	V
K	R	7	2	3	p	
T	R	8	4	5	m	
T	R	9	6	7	n	

			5
3			
	4		

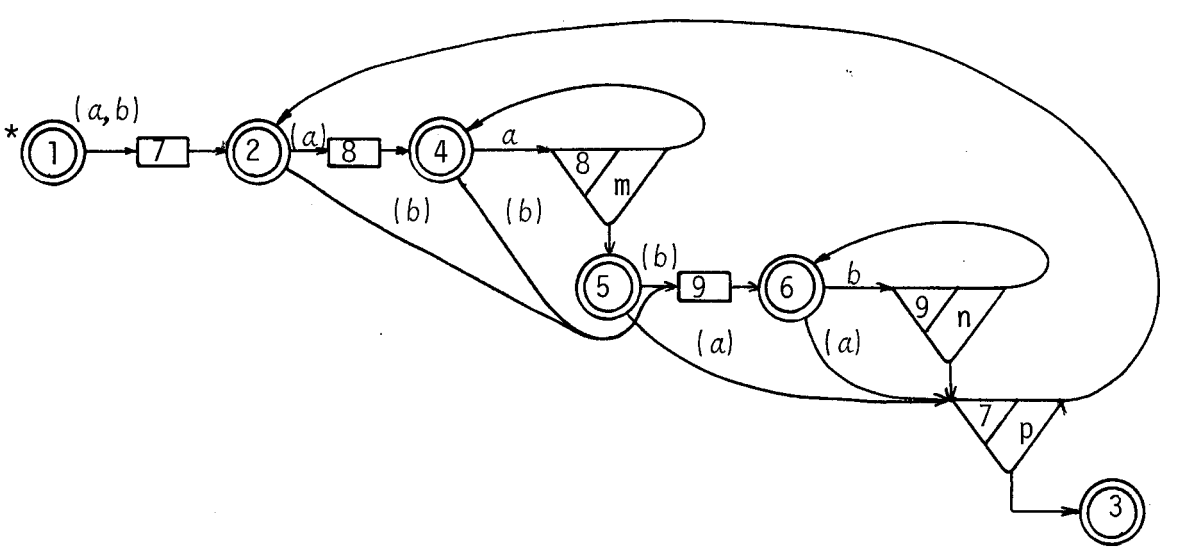
Exemplo completo, com as variáveis utilizadas pelo CONSTROICUIA e a resolução das colisões. "⊣" é o símbolo de 'fim de expressão'.



Mas este AFC ainda pode ser melhorado, tornando sua utilização mais eficiente, através de uma rotina que elimine as ocorrências do tipo ↑q - transição para o estado q sem avanço na entrada:

	a	*b*	
→ 1	R 7	R 7	F
2	R 8	R 9	F
3	-	-	F
4	8	R 9	F
5	↑ 7	R 9	F
6	↑ 7	9	F

	N	E	C	V
7	2	3	p	
8	4	5	m	
9	6	7	n	



Ocorrências do tipo ↑s em que s é uma ação semântica não podem ser eliminadas uma vez que a transição depende do valor do contador após o decremento.

Suponhamos p = 4, m = 2, n = 3. Vamos reconhecer *aabab*.

$(1,(0,0,0), aabab) \underset{M_C}{\vdash\!\!-\!\!-} (2,(4,0,0), aabab) \underset{M_C}{\vdash\!\!-\!\!-}$

$(4,(4,2,0), aabab) \underset{M_C}{\vdash\!\!-\!\!-} (4,(4,1,0), abab) \underset{M_C}{\vdash\!\!-\!\!-}$

$(5,(4,0,0), bab) \underset{M_C}{\vdash\!\!-\!\!-} (6,(4,0,3), bab) \underset{M_C}{\vdash\!\!-\!\!-}$

$(6,(4,0,2), ab) \underset{M_C}{\vdash\!\!-\!\!-} (2,(3,0,2), ab) \underset{M_C}{\vdash\!\!-\!\!-}$

$(4,(3,2,2), ab) \underset{M_C}{\vdash\!\!-\!\!-} (4,(3,1,2), b) \underset{M_C}{\vdash\!\!-\!\!-}$

$(6,(3,1,3), b) \underset{M_C}{\vdash\!\!-\!\!-} (6,(3,1,2), \varepsilon)$

6 é um estado final: aceito.

CAPÍTULO 4 - CONCLUSÕES E PERSPECTIVAS

I Automato finito com contadores ainda é uma idéia em embrião: seu formalismo ainda deve ser melhorado e sua teoria não está completamente desenvolvida. Mas apresenta algumas VANTAGENS significativas em relação ao AFD usual:

1) É um modelo mais realista quanto à aplicações: dada a natureza finita das máquinas computadoras, o operador *, na prática, é sempre implementado como *n.

2) Pode ser construido automaticamente, a partir da expressão regular estendida, através do algoritmo CONSTROICUIA. É uma ferramenta poderosa na construção de analisadores léxicos, aplicação prática típica do operador *n.

3) É muito eficiente no que concerne a sua proposição básica: salvar espaço. No exemplo clássico $(p)^{*n}$, onde p é uma expressão regular descrita por um automato de m estados, o AFD usual terá $\mathcal{O}(mn)$ estados e o AFC terá $\mathcal{O}(m)$ estados. E, ainda, se a expressão regular for $((p)^{*n})^{*q}$, o AFD correspondente terá $\mathcal{O}(mnq)$ estados, enquanto o AFC terá $\mathcal{O}(m)$ estados. Portanto, sua economia de espaço depende não apenas da estrutura da expressão regular como dos valores limite dos contadores.

Por exemplo:

Expressão Regular	AFD (estados)	AFC (estados+semântica)	Redução de espaço
$\ell(\ell\|d)^{*5}$	7	4 + 1	29%
$\ell(\ell\|d)^{*62}$	64	4 + 1	92%
$((ab)^{*4})^{*5}$	60	5 + 2	88%

II Sua principal DESVANTAGEM é seu tempo total de reconhecimento ligeiramente maior, embora $\mathcal{O}(|x|)$ onde x é a sentença, que o de um AFD usual, em decorrência do fato de o AFC realizar testes adicionais e um maior número de acessos às tabelas.

III O automato finito com contadores foi aqui apresentado com uma RESTRIÇÃO: expressões do tipo $(\alpha.\beta)^{*m}.(\alpha\ \delta)^{*n}$ são intratáveis e não dispõem ainda de AFC. Além disso, nosso algoritmo CONSTROICUIA exige que as expressões 'recuperáveis' sejam transformadas em "bem comportadas" antes de serem processadas por ele: é um cuidado que o usuário deve tomar. (Ver 3.3 XV).

IV Sendo um assunto vasto, a teoria de automatos finitos com contadores não foi esgotada neste trabalho, havendo ainda alguns importantes TÓPICOS A DESENVOLVER:

1) Formalização mais adequada do AFC.

A formalização apresentada neste trabalho é um pouco pesada, difícil de trabalhar e pode, possivelmente , ser melhorada.

2) Minimização de AFCs.

Tópico importante, porquanto o objetivo de um AFC é justamente reduzir o espaço ao mínimo. Não podemos afirmar, até este ponto da pesquisa, se o AFC produzido por nosso construtor é ou não é mínimo, embora haja indicações de que o seja.

3) Determinização de AFC's.

Por construção, não ocorrem indeterminações entre estados de AFC's obtidos através do CONSTROICUIA; expressões do tipo $((\alpha.\beta)^{*n}.\gamma)|((\alpha.\delta)^{*m}.\theta)$, porém levam a indeterminações entre contadores ainda que γ e/ou θ sejam ε (Ver 3.3 XV).

4) Novas aplicações para o AFC.

Dispomos agora de uma nova e poderosa ferramenta para construção e implementação de analisadores léxicos: o AFC. Entretanto, tendemos a crer que suas aplicações não param aí: buscar novas aplicações para o modelo é um problema em aberto. Se tirarmos proveito do fato de que o limite de um contador é um número armazenado em uma posição de memória, que pode

ser alterado a qualquer momento que se queira, suspeitamos que o AFC possa ser utilizado também em análise sintática.

BIBLIOGRAFIA

- Hopcroft-Ullman |1969|: "Formal Languages and Their Relation to Automata", Addison-Wesley, Reading, Mass.

- Aho-Ullman |1972|: "The Theory of Parsing, Translation and Compiling, Vol. I, Parsing". Prentice-Hall, Englewood Cliffs, N.J.

- Aho-Ullman |1977|: "Principles of Compiler Design", Addison-Wesley, Reading, Mass.

- Hopcroft-Ullman |1979|: "Introduction to Automata Theory, Languages and Computation", Addison-Wesley, Reading, Mass.

- Gries|1971|: "Compiler Construction for Digital Computers", John Wiley & Sons, inc., N.Y.

- Simone-Pereira|80|: "Algoritmos para Gramáticas RRP SLR(1)" in Anais do 7º SEMISH, SBC. Campinas, 1980.

- Knuth|1968|: "The Art of Computer Programming", Vol. I: Fundamental Algorithms, Addison-Wesley, Reading, Mass.