NOTAS TÉCNICAS
VOLUME IV
Nº 3

ALGORÍTMOS NÃO NUMÉRICOS PARA RESOLUÇÃO DE ALGUNS PROBLEMAS NA TEORIA DE LINGUAGENS FORMAIS E AUTÔMATAS

por

A. S. PFEFFER
E
A. T. DE MEDEIROS

Rio de Janeiro, 11 de novembro de 1971

CENTRO BRASILEIRO DE PESQUISAS FÍSICAS
Av. Wenceslau Braz, 71
Rio de Janeiro,
BRASIL
1971

PREFÁCIO

Êste trabalho contêm os seguintes algorítmos:

1) Um reconhecedor geral para determinar se um dado "string" pertence a linguagem gerada por uma dada grâmatica.

2) Um reconhecedor para determinar se um dado "string" pertence a linguagem gerada por uma gramática de contexto livre ou regular.

3) Construção do mínimo autômata de estados finitos correspondente à uma dada gramática regular.

4) Construção de um "Push Down Autômata" correspondente a uma dada gramática de contexto livre.

5) Construção da MÁQUINA UNIVERSAL DE TURING.

6) Um reconhecedor de "strings" gerados por uma gramática LL(k).

O objetivo deste trabalho é o de apresentar os algorítmos tão simples quanto possível (mas com alguma otimização) de modo que o leitor possa compreender rápidamente os passos dos algorítmos. É dada uma breve descrição dos conceitos necessários aos algorítmos. Todos êstes algorítmos foram implementados na linguagem SNOBOL 3 e foram testados no IBM 1620 do Centro Brasileiro de Pesquisas Físicas.

Os autores agradecem aos professôres D. C. Binns e G. Schwachheim do Centro Brasileiro de Pesquisas Físicas e aos professôres R. L. de Carvalho e A. L. Furtado da Pontifícia Universidade Católica do Rio de Janeiro pela valiosa contribuição ao desenvolvimento dêste trabalho.

I - INTRODUÇÃO

I.1 - GRAMÁTICAS E SEUS RECONHECEDORES

Consideremos a gramática mais geral de todas e denominemo-la gramática tipo (0). Podemos fazer certas restrições sobre a natureza das produções desta gramática e então derivar três outros tipos de gramáticas (1):

Seja $G = (V_N , V_T , P, S)$ uma gramática qualquer onde:

V_N = Conjunto de símbolos não terminais

V_T = Conjunto de símbolos terminais

P = Produções da gramática

S = Símbolo inicial

Usaremos as seguintes definições a respeito das gramáticas:

a) Sensível ao contexto ou tipo 1- para qualquer produção $A \rightarrow B$ em P, temos $|B| \geq |A|$

b) Livre do contexto ou tipo 2- para qualquer produção $A \rightarrow B$ em P, A é uma variável simples e B é qualquer "String", excetuando-se ε (vazio).

c) Regular ou tipo 3- para qualquer produção A → B em P, A é uma variável simples e |B| é 1 ou 2. Se |B| = 1, B é um terminal e se |B| = 2, B é composto por um terminal e uma variável, nesta ordem.

d) $V_N \cup V_T$ é a união do conjunto de variáveis e do conjunto de terminais. O resultado é V, o alfabeto de uma dada gramática.

e) Um 'string' é qualquer sequência composta de símbolos de um alfabeto. O conjunto de todos os strings com símbolos de um alfabeto V é escrito como V^* e $V^+ = V^* - \{\epsilon\}$.

I.2 - ARMAZENAMENTO DA GRAMÁTICA

Vamos agora imaginar como uma gramática G poderia ser armazenada, usando as facilidades de linguagem SNOBOL 3(4,5,6). Podemos armazenar V_N, V_T e S como 'Strings', mas as produções são regras de tipos:

lado esquerdo → lado direito

cujo significado é:

"O lado esquerdo pode ser substituído pelo lado direito"

Assim usamos o lado esquerdo como nome de 'String' que aponta para o seu próprio lado direito. As produções que tem o mesmo lado esquerdo (com lados

direitos diferentes) são numeradas ordenadamente a partir de um número qualquer (usamos 1).

Para generalizar um nome para o lado esquerdo ou direito das produções, vamos chamar o lado esquerdo por L e o lado direito por R. Então L conterá o lado esquerdo da produção e o conteúdo de L($L) apontará para o conteúdo de R, ou seja, o próprio lado direito.

A menos que seja explícitamente escrito de outra forma, esta será a técnica usada para ler as produções em todos os algorítmos aqui apresentados e o símbolo $ será usado para significar o "conteúdo de ".

1.1 - O RECONHECEDOR GERAL

Partindo das definições apresentadas em (I.1) propomos o seguinte problema:

Dada uma gramática $G = (V_N, V_T, P, S)$ e um 'String' w, construir um algorítmo para determinar se o 'String' w, pertence ou não à linguagem gerada por G.

Em termos informais, as principais idéias envolvidas são:

a) testar se os símbolos de w, pertencem a V_T

b) criar uma lista com todos os lados direitos das produções que tenham S como lado esquerdo. Com esta lista e as produções criar novos 'Strings' cujo comprimento não exceda o comprimento de w.

expandindo o item B é necessário fazer os seguintes testes:

a) se $|\text{'String'}| > |w|$, o 'string' recém criado deve ser retirado da lista.

b) se os elementos do 'String' recém criados pertencem sòmente a V_T, êle deve ser comparado a w. Se não forem iguais deve ser retirado da lista e em caso contrário deve ser aceito.

c) se o 'String' recém criado não pertence a lista, êle deve ser colocado nela.

O último dêstes ítens (c) provoca uma pergunta a respeito das listas. Quantas listas serão necessárias para testar o 'String'. Ou ainda, todos os 'Strings' criados devem ser armazenados até a aceitação ou rejeição de w ?

A resposta depende se a gramática é recursiva ou não. Se a gramática não for recursiva necessitamos de duas listas:

uma para armazenagem e comparações
outra para gerar os 'Strings'

Se a gramática for recursiva necessitamos armazenar todas as listas produzidas e usá-las para comparações.

1.2 - O ALGORÍTMO

Após estas considerações podemos apresentar o algorítmo. Usaremos os seguintes nomes:

VARIABLE — conjunto de variáveis (V_N)

TERMINAL — conjunto de terminais (V_T)

LEFTSIDE — conjunto dos lados esquerdos das produções

WORKLIST — lista onde são produzidos os 'Strings'

STORLIST — lista onde são armazenados os 'Strings'

INICIALIZAÇÃO

A1) Leia o 'String' w e a gramática da forma explicada em I.2 e produza a lista LEFTSIDE com o lado esquerdo das produções e STORLIST com o lado direito das produções cujo lado esquerdo tem sòmente o símbolo inicial.

EM QUALQUER PONTO

B1) Retire um 'String' de STORLIST. Se STORLIST for vazio Vá para D1

B2) Retire um elemento de LEFTSIDE e teste se faz parte do 'String' retirado do STORLIST. Em caso afirmativo substitua, no 'String', o elemento pelo seu conteúdo e vá para C1. Em caso contrário vá para B2. Se LEFTSIDE for vazio, vá para B1.

ANÁLISE DO 'STRING' RECÉM PRODUZIDO

C1) Se |'String'| > |w|, retorne a B1

C2) Se o 'String' pertence sòmente ao conjunto de terminais e é igual a w, aceite w. Em caso contrário retorne a B1

C3) Se WORKLIST ou STORLIST tem outro 'string' igual ao 'String' que está sendo testado, retorne a B1

C4) Concatene WORKLIST com o 'String' em WOKLIST e retorne a B1

ANÁLISE DAS LISTAS

D1) Se WORKLIST for vazio, rejeite w .

D2) Se a gramática não é recursiva faça STORLIST ← WORKLIST. Em caso contrário faça STORLIST ← STORLIST concatenado com WORKLIST e apague WORKLIST. Em qualquer caso retorne a B1.

2 - RECONHECEDOR DE 'STRINGS' GERADOS POR GRAMÁTICAS REGULARES OU DE CONTEXTO LIVRE

A maior contribuição à ineficiência do algorítmo apresentado em 1.2 é dada pelo fato de serem gerados todos os possíveis 'Strings' cujo comprimento não excede o comprimento do dado 'String' w. Vamos agora introduzir um algorítmo para reconhecer um dado 'string' produzido por uma dada gramática regular ou livre do contexto.

Este algorítmo seleciona uma produção baseando-se numa análise feita sôbre:

a) Os comprimentos da produção (lado direito) e da parte não examinada do 'String'

b) Os símbolos da produção (lado direito) e o 'String', iniciando com o símbolo que está sendo examinado.

As produções usadas são armazenadas numa pilha, de modo que se não for possível prosseguir o reconhecimento, a produção no tôpo da pilha é removida, uma outra é testada, eventualmente armazenada e assim por diante.

O 'String' será aceito se todos os seus símbolos foram examinados e a pilha não ficar vazia. Se estas condições não ocorrerem o 'String' será rejeitado.

2.1 - O ALGORÍTMO

Usaremos os seguintes nomes:

VARIABLE — conjunto de variáveis

TERMINAL — conjunto de terminais

INICIAL — símbolo inicial

W — nome do 'String' de entrada

SIZET, SIZEW — conterão comprimentos atuais de TVAR e W

HVAR, TVAR, XVAR, MVAR — 'Strings' auxiliares

RULEA, RULEB — 'Strings' de armazenamento e trabalho, respectivamente, dos lados direitos das produções.

INICIALIZAÇÃO

A1) Leia a gramática conforme em I.2 e o 'string' w a testar.

A2) Se a gramática é regular, acenda o indicador RG

A3) Faça TVAR ← INITIAL e SIZET ← |TVAR|

EM QUALQUER PONTO

B1) Retire o primeiro símbolo de TVAR e coloque-o em HVAR. Se HVAR e w forem vazios, aceite o 'String'. Se w não for vazio, vá para B8.

B2) Se HVAR for um terminal, retire o primeiro símbolo de w e coloque-o em MVAR. Se MVAR = HVAR, vá para B1. Se diferentes, faça TVAR ← HVAR concatenado com TVAR, w ← MVAR concatenado com w e vá para B8.

B3) Tome a primeira regra apontada pelo conteudo de HVAR ($HVAR) e coloque seu lado direito em RULEA. Se o indicador RG estiver apagado e RULEA tiver símbolos terminais após a variável mais a direita, teste se êstes terminais são iguais aos respectivos últimos símbolos de w. Se o teste falhar, vá para B5.

B4) Se |RULEA| + SIZET - 1 < SIZEW, faça RULEB ← RULEA e vá para B6.

B5) No conteúdo de HVAR ($HVAR) tome a primeira regra após RULEA e coloque o seu lado direito em RULEA e vá para B4. Se falhar vá para B8.

B6) Retire o primeiro símbolo de RULEB e coloque-o em XVAR. Se XVAR pertence a VARIABLE, faça TVAR ← XVAR concatenado com RULEB com TVAR e vá para B7. Se XVAR pertence a TERMINAL, retire o primeiro símbolo do $\underline{w}$ e coloque-o em MVAR. Se MVAR = XVAR vá para B6 e se diferentes, vá para B5.

B7) Aqui XVAR é vazio. Coloque HVAR, um delimitador e RULEB concatenados, nesta ordem, no topo da pilha. Faça SIZET ← |TVAR|, SIZEW ← $|\underline{w}|$ e vá para B1.

B8) Retire o topo da pilha e se falhar rejeite $\underline{w}$. Se não, decomponha-o em HVAR e RULEA. Retire de RULEA todos os símbolos terminais (da esquerda para direita) até o primeiro símbolo não terminal e concatene-os a frente de $\underline{w}$. Faça TVAR ← HVAR concatenado com TVAR, SIZET ← $|w|$ e vá para B5.

3 - O RECONHECEDOR DA GRAMÁTICA REGULAR - O AUTOMATA-DE ESTADOS FINITOS

O reconhecedor da linguagem gerada por gramática regular é o autômata de estados finitos (FSA) que é definido como um sistema quíntuplo (Q,Σ,δ,qo, F) onde:

Q - conjunto de estados finitos

Σ - alfabeto finito de entrada

δ - 'mapping' de Q estados por Σ alfabeto em Q estados

qo- estado inicial pertencente ao conjunto Q

F - subconjunto de Q, conjunto de estados finais.

O autômata de estados finitos (FSA) tem um contrôle finito capaz de ler um símbolo de cada vez na fita de entrada por meio de uma cabeça de leitura. Também é capaz de armazenar símbolos. Inicialmente o contrôle finito está no estado qo e a cabeça de leitura lê um símbolo de um 'String' de símbolos que aparecem na fita de entrada e pertencem ao conjunto Σ. O 'mapping' $\delta(q,a)=p$, para $\underline{q} \text{ e } \underline{p} \in \underline{Q}$ e $\underline{a} \in \underline{\Sigma}$ significa que o fsa no estado $\underline{q}$ e vendo $\underline{a}$, move sua cabeça de leitura para a próxima célula de fita e passa ao estado $\underline{p}$.

Quando o 'mapping' é extendido ao domínio de $Q \times \Sigma^*$, definimos o 'mapping' como:

$\hat{\delta}(q,\varepsilon)= q$

$\hat{\delta}(q,ya)= \hat{\delta}(q,y),a)$ para cada $\underline{Y} \in \underline{\Sigma}^* \text{e } \underline{a} \in \underline{\Sigma}$

A interpretação de $\hat{\delta}(q,y)= p$ é que o FSA, partindo do estado q e com o 'String' $\underline{Y}$ na fita de entrada passará ao estado $\underline{p}$ após sua cabeça de leitura mover-se para a direita sôbre a porção da fita de entrada que contém $\underline{Y}$.

Um 'String' é aceito pelo FSA se $\delta(q_0,y) = p$, para $p \in F$

Podemos também considerar um FSA não determinístico (NDFSA) cuja definição formal é semelhante àquela apresentada para o FSA. A diferença entre êles reside no fato de $\delta(q,a)$ significar que o NDFSA pode escolher não determinísticamente um estado entre um conjunto de estados, onde êste conjunto pode ser vazio. A interpretação de $(q,a) = \{p_1, p_2, \ldots\ldots, p_n\}$ é que o NDFSA no estado q e vendo a na fita de entrada move sua cabeça de leitura para a próxima célula da fita e escolhe um dos estados $p_1, p_2, \ldots\ldots, p_n$ como próximo estado. O 'String' é aceito do mesmo modo que no FSA.

Para simplificar faremos a construção do FSA em dois passos:

1) construção do NDFSA a partir da gramática regular

2) construção do FSA a partir do NDFSA construído em (1)

Além dêstes dois passos o NDFSA poderá ser reduzido e minimizado.

3.1 - CONSTRUÇÃO DE UM AUTÔMATA NÃO DETERMINÍSTICO DE ESTADOS FINITOS

Nosso propósito será construir um autômata não determinístico de estados finitos (NDFSA) a partir de uma gramática regular e que reconheça strings pertencentes a linguagem gerada por ela. Podemos dividir nosso algorítmo em três partes:

1) Determinação do conjunto de estados, conjunto de estados finais e do estado inicial

2) Determinação dos 'mappings'

3) Saída do NDFSA

Antes de apresentar o algorítmo vamos considerar alguns fatos. Cada símbolo de V_N deve ser relacionado com um estado (1). Para simplificar o algorítmo usaremos os nomes dos elementos de V_N como nomes de estados. Assim cada elemento de V_N tem dupla interpretação:

a) como um elemento da gramática êle pertence a V_N

b) como um elemento do NDFSA êle pertence ao conjunto Q de estados

Deste modo podemos escrever $Q = V_N \cup \{F\}$ onde $\{F\}$ é o conjunto de estados finais. Êste conjunto é determinado pela seguinte regra:

"Para cada produção do tipo $A \rightarrow B$ onde $A \in V_n$ deve ser criado um novo símbolo diferente de qualquer um pertencente ao conjunto V_n. Este novo símbolo passará a integrar o conjunto de estados finais".

Quanto ao estado inicial êle será representado pelo símbolo inicial.

3.1.1 - O ALGORÍTMO

Usaremos os seguintes nomes:

VARIABLE - conjunto de variáveis

TERMINAL - conunto de terminais

STATELIST - conjunto de estados

ENDLIST - conjunto de estados finais

SYMBOL - qualquer símbolo copiado de VARIABLE ou STATELIST

NEWSYMBOL - qualquer símbolo criado

TERM - qualquer símbolo do conjunto de terminais

INICIALIZAÇÃO

A1) Leia a gramática

A2) Faça ENDLIST vazia e copie para STATELIST todos os símbolos de VARIABLE.

DETERMINAÇÃO DOS CONJUNTOS DE ESTADOS E ESTADOS FINAIS

B1) Tome uma produção. Se falhar vá para C1.

B2) Se a produção for composta (lado direito) de terminal e variável, vá B1.

B3) Se o conteúdo apontado pelo nome do terminal ($TERM) não for vazio, vá para B5.

B4) Crie um novo símbolo (NEWSYMBOL) diferente dos símbolos existentes em STATELIST e coloque-o nos conjuntos STATELIST e ENDLIST. Faça $TERM ← NEWSYMBOL

B5) Concatene NEWSYMBOL ao lado direito da produção que está sendo analisada e vá para B1.

CONSTRUÇÃO DOS 'MAPPINGS'

C1) Refaça o conjunto TERMINAL e coloque um elemento do conjunto VARIABLE em SYMB. Se VARIABLE for vazio vá para D1.

C2) Tome um elemento do conjunto TERMINAL e coloque-o em TERM. Se falhar vá para C1.

C3) Procure em todas as produções apontadas por SYMB ($SYMB) aquelas cujos lados direitos tenham o mesmo terminal que TERM contém.

C4) Faça SYMB ← SYMB concatenado com TERM e faça o conteúdo de SYMB ($SYMB) apontar para as variáveis que são pares de TERM no lado direito das produções relacionadas em C3. Vá para C2.

SAÍDA DO NDFSA

D1) Imprima STATELIST e ENDLIST como os conjuntos de estados e estados finais, respectivamente. Imprima o símbolo inicial da gramática como estado inicial do NDFSA. Imprima o conjunto de terminais como o alfabeto de entrada do NDFSA.

D2) Concatene cada um dos componentes do conjunto STATELIST com cada um dos componentes de TERMINAL e imprima o conteúdo destas concatenações como o conjunto de 'mappings'.

3.2 - CONSTRUÇÃO DO FSA A PARTIR DO NDFSA

Nosso próximo passo será a construção do FSA a partir do NDFSA produzido pelo último algorítmo.

A entrada será o próprio NDFSA, isto é, o conjunto de estados K, o conjunto de estados finais F, o alfabeto Σ, o estado inicial e os 'mappings' $K \times \Sigma \rightarrow K$.

As principais idéias sôbre êste algorítmo são:

1) produção do conjunto de estados Q do FSA a partir do conjunto de estado K do NDFSA.

2) produção dos 'mappings' $Q\times\Sigma\rightarrow Q$ do FSA usando os conjuntos de estados K e Q e as 'mappings' $K\times\Sigma\rightarrow K$ do NDFSA.

3.2.1 O ALGORÍTMO

Usaremos os seguintes nomes:

SETK - conjunto de estados do NDFSA

SETQ - conjunto de estados do FSA

ENDK - conjunto de estados finais do NDFSA

SETΣ - alfabeto de entrada

INITIALK - estado inicial do NDFSA

ALIST - nome de 'string' de trabalho.

INICIALIZAÇÃO

A1) Leia SETK, ENDK, INITIALK e SETΣ. Leia os 'mappings' $K \times \Sigma$ do mesmo modo como explicado em I.2 para as produções das gramáticas.

CONSTRUÇÃO DO CONJUNTO DE ESTADOS DO FSA

B1) A partir do conjunto K de estados (ndfsa) gere o seu conjunto potência que será o conjunto Q(fsa). Para cada subconjunto crie um novo estado diferente daqueles existentes em SETK, coloque-o em SETQ e faça o novo estado apontar para a combinação e vice-versa ($ COMBINAÇÃO ← NEWSTATE e $NEWSTATE ← COMBINAÇÃO). Ponha em SETQ mais um estado de número de ordem imediatamente superior ao último.

CONSTRUÇÃO DOS 'MAPPINGS' DO fsa

C1) Retire um estado do conjunto SETQ e coloque-o em STATEQ. Se SETQ for vazio, vá para D1.

C2) Retire um elemento de SETΣ. Se SETΣ for vazio redefina-o e vá para C1.

C3) Separe o conteúdo dos estados ($STATEQ) nos estados originais do NDFSA e concatene cada um deles com o elemento removido de SETΣ.

C4) Para cada concatenação tome o seu conteúdo e coloque-o em ALIST. Elimine todos estados repetidos.

C5) Se ALIST for vazio, faça o conteúdo de STATEQ apontar para ε($STATEQ ← ε). Em caso contrário faça o conteúdo de STATEQ apontar para o conteúdo de ALIST($STATEQ ← ALIST) e vá para C1.

SAÍDA DO fsa

D1) Imprima SETQ, SETΣ e o conteúdo de INITIALK ($INITIALK) como o conjunto de estados, alfabeto e estado inicial, respectivamente.

D2) Retire um estado de SETQ e coloque-o em STATEQ. Se SETQ for vazio, vá para D4.

D3) Se o conteúdo de STATEQ($STATEQ) tem algum símbolo que pertença a ENDK, imprima STATEQ como um dos estados finais e va para D2.

D4) Imprima o conteúdo do 'mapping' de SETQ x SETΣ → SETQ como os 'mappings' do FSA.

3.3 - REDUÇÃO DO FSA

O FSA obtido em 3.2.1 tem alguns estados inacessíveis e outros acessíveis apenas através deles próprios. Isto significa que há 'mappings' do tipo:

$\delta(q_i,a) = q_i$ ou $\delta(q_k,a) = q_j$ e $\delta(q_j,a) = q_k$

Dizer que q_i é um estado inacessível significa dizer que não há 'mapping' do tipo $\delta(q_m,a) = q_i$. Para $\delta(q_k,a) = q_j$ e $\delta(q_j,a) = q_k$ não há 'mapping' do tipo $\delta(q_m,a) = q_k$ e $\delta(q_m,a) = q_j$, para $m \neq k$ e $m \neq j$.

Para reduzir o FSA usaremos o fato de que os estados que não acessíveis podem ser atingidos através do estado inicial. Dêste modo procuraremos os estados obtidos a partir do estado inicial, usaremos êstes estados para procurar outros e assim por diante.

3.3.1 - O ALGORÍTMO

Usaremos os seguintes nomes:

SETQ - conjunto de estados do FSA não reduzido

SETQ1, DELTA - nomes de 'Strings' de trabalho

SETQ2 - conjunto de estados do FSA reduzido

SETF - conjunto de estados do FSA não reduzido

SETF1 - conjunto de estados do FSA reduzido

SETSIGMA - alfabeto do FSA

INICIALIZAÇÃO

A1) Leia o FSA não reduzido. Os 'mappings' devem ser lidos fazendo o lado esquerdo apontar para o seu lado direito.

A2) Faça SETQ2 E SETF1 vazios e coloque o estado inicial em SETQ1.

PESQUISA DOS ESTADOS ACESSÍVEIS

B1) Retire o primeiro estado de SETQ1 e coloque-o em SETQ2. Se falhar vá para C1.

B2) Retire um elemento de SETSIGMA. Se falhar redefina SETSIGMA e vá para B1.

B3) Concatene o estado (de B1) e o elemento (de B2) em DELTA e tome o estado apontado pelo conteúdo de DELTA($DELTA). Se êste estado não pertencer a SETQ1 ou SETQ2, concatene-o com os que já existem em SETQ1. Em qualquer hipótese vá para B2.

PESQUISA DOS ESTADOS FINAIS

C1) Retire um estado de SETF. Se falhar vá para D1.

C2) Se o estado pertence a SETQ2, coloque-o em SETF1. Em qualquer hipótese vá para C1.

SAÍDA DO FSA REDUZIDO

D1) Imprima SETQ2, SETSIGMA, SETF1 e o estado inicial.

D2) Retire um estado de SETQ2. Se falhar pare.

D3) Retire um elemento do SETSGIMA. Se falhar redefina SETSIGMA e vá para D2.

D4) Concatene o estado (de D2) e o elemento (de D3) em DELTA. Se o conteúdo de DELTA ($DELTA) for vazio vá para D3.

D5) Imprima DELTA e seu conteúdo ($DELTA) como um dos 'mappings' e vá para D2.

3.4 MINIMIZAÇÃO DO FSA

Vamos mostrar agora como obter um FSA mínimo a partir de um FSA reduzido. Esta operação é denominada minimização e é o último passo da produção de um FSA partindo de uma gramática regular.

A idéia é substituir, se possível, um conjunto de estados equivalentes por sòmente um estado que execute a mesma função que êste conjunto de estados. Dividimos esta idéia em cinco partes principais:

1) Após a leitura do FSA reduzido, separar o conjunto {Q} de estados em duas partes:o conjunto {Q-F} e o conjunto {F}

2) Para todos os 'mappings', tome o estado para o qual êle aponta e verifique a qual dos dois conjuntos êste estado pertence.

3) Agrupe os estados de acôrdo com os conjuntos {Q-F} ou {F} a que êles pertençam. Repita os passos 2 e 3 até que o número de estados seja igual ao número de conjuntos ou que dois consecutivos grupos de conjuntos sejam iguais.

4) No último grupo de conjuntos, tome cada um dos conjuntos e elimine todos os estados, exceto um deles. Apague os 'mappings' que tenham os estados eliminados e reaponte os 'mappings' que apontam para os estados eliminados.

5) Imprima o FSA.

3.4.1 - O ALGORÍTMO

Os 'mappings' serão lidos do mesmo modo pelo qual foram lidos no último algorítmo. Os nomes SETQ, SETF, etc, serão usados como nomes de 'Strings' de trabalho e podem ser refeitos. Os nomes BLOCKA e BLOCKB são nomes de 'Strings' para os grupos de estados usados nos itens 2 e 3 do parágrafo 3.4. NORDER é um

string de trabalho usado para a formação de BLOCKA. CONJ é um nome de string usado para analisar NORDER. DELTA é um nome de string de trabalho.

INICIALIZAÇÃO

A1) Leia o FSA.SETQ é o conjunto de estados, SETF é o conjunto de estados finais, SETΣ é o alfabeto de entrada do FSA e S é o estado inicial.

A2) Coloque os conjuntos {Q-F} e {F} em BLOCKA separados por um delimitador.

CONSTRUÇÃO DOS NOVOS CONJUNTOS

B1) Enumere os conjuntos que estão em BLOCKA

B2) Retire um estado de SETQ. Se falhar vá para C1.

B3) Retire um elemento de SETΣ. Se falhar coloque um delimitador em NORDER, redefina SETΣ e vá para B2.

B4) Faça DELTA igual ao estado (de B2) concatenado ao elemento (de B3) do alfabeto e procure em BLOCKA pelo número de ordem do grupo de estados ao qual pertence o estado apontado por DELTA($DELTA). Concatene êste número em NORDER e vá para B3.

CONSTRUÇÃO DOS GRUPOS DE ESTADOS

C1) Numere os conjuntos em NORDER.

C2) Copie o primeiro conjunto de NORDER e coloque-o em CONJ. Se falhar,vá para C4.

C3) Retire todos os estados iguais a CONJ de NORDER e coloque o número de ordem deste conjunto em BLOCKB. Coloque o delimitador e vá para C2.

C4) Se BLOCKA e BLOCKB são iguais, vá para D1. Se o número de conjuntos de BLOCKAB for igual ao número de estados de SETQ, vá para E1. Em caso contrário vá para B1.

CONSTRUÇÃO DO FSA MÍNIMO

D1) Retire um conjunto de BLOCKB. Se êste conjunto tem mais de um estado coloque o conjunto em CONJ e um dos seus estados em STATE. Se houver apenas um estado, vá para D1. Se BLOCKB for vazio vá para E1.

D2) Procure em SETQ e SETF pelo primeiro estado que pertença simultâneamente a SETQ e CONJ ou a SETF e CONJ e substitua-o por STATE. Apague em SETQ e SETF qualquer outro estado que pertença simultâneamente a SETQ e CONJ ou a SETF e CONJ. Se o estado inicial pertence a CONJ, substitua-o por STATE:

D3) Para cada estado em SETQ, concatene em DELTA o estado com cada um dos elementos do alfabeto e tome o estado apontado por conteúdo de DELTA($DELTA).

D4) Se êste estado (apontado por $DELTA) pertencer a SETQ faça $DELTA ← STATE e vá para D1.

IMPRESSÃO DO fsa MINIMIZADO

E1) Imprima SETQ, SETF e SETΣ como o conjunto de estados, o conjunto de estados finais e o alfabeto, respectivamente. Imprima o estado inicial.

E2) Para cada estado em SETQ, concatene em DELTA um estado e um elemento do alfabeto. Imprima DELTA e o seu conteúdo ($DELTA) como o 'mapping' associado a êste estado e aquele elemento do alfabeto. Repita êste passo até que SETQ se torne vazio.

4. - O RECONHECEDOR DA GRAMÁTICA LIVRE DO CONTEXTO - O AUTÔMATA PUSH DOWN

O reconhecedor de gramáticas livres do contexto é o autômata PUSH DOWN (PDA). Êste autômata é essencialmente um autômata de estados finitos com contrôle sôbre a fita de entrada e sôbre a armazenagem em pilha.

O PDA é definido como um sistema $(Q,\Sigma,\Gamma,\delta,q_o,Z_o,F)$ onde:

Q - conjunto finito de estados

Σ - alfabeto de entrada, um conjunto finito de símbolos

Γ - alfabeto da pilha, um conjunto finito de símbolos

qo - estado inicial, pertencente a Q

Zo - símbolo inicial do pda, pertencente a Γ

F - conjunto de estados finais, pertencentes a Q

- 'mapping' de Q x (ΣU{ε} Γ

4.1 - OPERAÇÃO DO PDA

Há dois tipos de movimentos. No primeiro tipo examina-se um símbolo da fita de entrada. Dependendo deste símbolo, do símbolo no tôpo da pilha e do estado do contrôle finito, a função δ pode tomar diversos valores. O contrôle finito recebe o estado escolhido e o símbolo no tôpo da pilha é substituído por um 'String' de símbolos, 'String'êste que pode ser vazio. Após êste movimento a cabeça de leitura avança para o próximo símbolo.

No segundo tipo do movimento, os símbolos de entrada não são usados e a cabeça de leitura não se move. Neste caso o PDA manipula a pilha sem ler símbolos de entrada.

Assim nós temos dois modos para definir a linguagem aceita pelo PDA. No primeiro, a linguagem será aceita se, após a sequência de movimento, a pilha estiver vazia. No segundo, a linguagem será aceita se, após a sequência de movimentos, o PDA passar a um estado final.

Para mostrar os movimentos do PDA, usaremos as seguintes convenções:

símbolo de entrada - a,b,c,d,...

'string' de símbolos de entrada - p,q,r,s,...

símbolos da pilha - A,B,C,D,...

'String' de símbolos da pilha - $\alpha,\beta,\delta,...$

A interpretação de - $(q,a,Z) = (p_1,\gamma_1),(p_2,\gamma_2),...,(p_n,\gamma_n)$ onde, q e p_i para $1< i < n \in Q$, a $\in \Sigma$, Z C Γ e γ_i, para $1 < i < n \in \Gamma^*$

é que o PDA, no estado q com o símbolo de entrada a e o símbolo Z no tôpo da pilha, pode, para qualquer i, entrar no estado p_i, substituir Z por γ_i e avançar a cabeça de leitura para o próximo símbolo.

Para $\delta(q,\varepsilon,Z) = (p_1,\gamma_1), (p_2,\gamma_2)... (p_n,\gamma_n)$ onde $1 < i < n$ e ε= vazio, o PDA, no estado q, com o símbolo Z no tôpo da pilha e para qualquer i, pode passar ao estado p_i, substituir Z por γ_i independentemente do símbolo de entrada. Neste caso, a cabeça de leitura não se move.

Se $G = (V_N,V_T,P,S)$ é uma linguagem livre do contexto, podemos definir um PDA M tal que:

$M = (Q, V_T, V_nUV_T,\delta,q,S,\phi)$ onde $q \in \{Q\}$ e:

$\delta(q,\varepsilon,y) = \{(q,z)$ y → z e P e z e $V_T\}$

$\delta(q,a,a) = (q,\varepsilon)$ para cada a $\in V_T$

4.2 - ALGORÍTMO

Apresentaremos um algorítmo para construir um PDA não determinístico que opera por pilha vazia para reconhecer uma linguagem livre do contexto. Usaremos os seguintes nomes:

SETΣ - alfabeto de entrada

SETΓ - alfabeto do PDA

INITIALSTORE - símbolo inicial na pilha

DELTA - cada um dos 'mappings'

SETQ - conjunto de estados

VARIABLE - conjunto de variáveis

TERMINAL - conjunto de terminais

INITIAL - símbolo inicial da gramática

INICIALIZAÇÃO

A1) Leia a gramática

CONSTRUÇÃO DO PDA NÃO DETERMINÍSTICO

B1) Imprima a letra Q como o conjunto de estados, o conjunto TERMINAL como SETΣ, VARIABLE concatenado com TERMINAL como SETΓ, letra Q como estado inicial, INITIAL como INITIALSTORE (Z_o) e a letra Q como conjunto de estados finais.

B2) Retire um elemento de SETΓ. Se falhar, pare.

B3) Se o elemento pertencer a VARIABLE, faça DELTA ← 'Q' concatenado com ε (vazio) e o elemento.

B4) Tome a regra apontada pelo elemento. Se falhar, vá para B7.

B5) Faça o conteúdo de DELTA igual ao conteúdo de DELTA concatenado com o delimitador, letra 'Q' e a regra. Vá para B4. ($DELTA ← $DELTA delimitados 'Q' Regra)

B6) Se o elemento pertence a TERMINAL, atribua a DELTA a letra 'Q' concatenado com o elemento e novamente com o elemento (duas vêzes). Atribua ao conteúdo de DELTA a letra 'Q' concatenado com ε(vazio).

B7) Imprima DELTA e seu conteúdo como o 'mapping' associado a êste elemento. Vá para B2.

5 - A MÁQUINA UNIVERSAL DE TURING

A máquina universal de TURING é outro tipo de reconhecedor. Para apresentá-la usaremos a definição de Hopcroft e Ullman

A máquina de Turing é usada como um modêlo matemático para descrever procedimentos. O modêlo básico tem um contrôle finito, uma fita de entrada dividida em células e uma cabeça de leitura que pode examinar uma célula de cada vez. A fita tem um princípio (à esquerda) mas pode extender-se indefinidamente para a direita e cada célula da fita pode conter exatamente um símbolo. As n celulas mais a esquerda (para qualquer n finito) contêm um 'String' de símbolos denominados símbolos de entrada. Os símbolos deste 'String' são um subconjunto de símbolos da fita e as restantes células da fita contêm brancos, um símbolo especial da fita e que não pertence ao conjunto de símbolos de entrada.

O movimento da máquina de Turing depende de dois fatôres:

1) do símbolo examinado pela cabeça da leitura
2) do estado do contrôle finito

A máquina de Turing pode executar as seguintes operações:

1) trocar de estado
2) imprimir um símbolo diferente do branco na célula examinada, substituindo aquele que foi examinado.
3) mover sua cabeça de leitura uma célula para a direita ou para a esquerda

Formalmente escrevemos:

TM = $(Q, \Sigma, \Gamma, \delta, q_0, F)$, onde:

Q = conjunto finito de estados

Γ = conjunto finito de símbolos da fita, incluindo o branco

Σ = símbolos de entrada

δ = o movimento da máquina, um 'mapping' de $Q \times \Gamma$ para $Q \times (\Gamma-\{B\}) \times \{L,R\}$, onde L e R indicam movimentos da cabeça de leitura para a esquerda ou direita, respectivamente.

q_0 = estado inicial pertencente ao conjunto Q

F = conjunto de estados finais e subconjunto de Q.

A configuração de uma máquina de Turing é representada por (q,α,i), onde:

q - estado de máquina e pertencente a Q

α - 'String' que não contém {B} e pertencente a Γ

i - inteiro que representa a distância da cabeça de leitura ao princípio da fita.

Podemos construir várias máquinas de Turing para diversas operações, tais como adições, subtrações, reconhecimento de 'Strings' que pertencem a alguma gramática, etc. A máquina universal de Turing (UTM) simula qualquer máquina de Turing e executa as suas funções. Para esta finalidade a máquina universal de Turing (UTM) usa sòmente dois símbolos, o zero (0) e o um (1). Assim, a entrada para a máquina universal de TURING é a própria máquina de Turing que irá simular e o 'String' a ser operado, ambos codificados em zeros e uns. Esta codificação pode ser feita de vários modos e aqui apresentamos um deles.

5.1 - DESCRIÇÃO DOS DADOS E DA ENTRADA DA UTM

Seja TM_1 a TM a ser simulada pelo UTM.

dados: a UTM aceita um alfabeto em {0,1,B,L,R}. Dêste modo os dados podem ser

codificados assim:

1) Todos os 'mappings' são representados. O código será zero (0) se a máquina de Turing não descreve um 'mapping' para um dado elemento do alfabeto.

2) Os 'mappings' dos estados finais são representados por zero (0)

3) Uma marca separa os símbolos do 'String' e o movimento da máquina de Turing para um estado e um símbolo da UTM.

4) Duas marcas separam os q_i x {B,0,1}

5) Três marcas separam o início da máquina de Turing e o fim dos códigos.

6) Quatro marcas separam a máquina de Turing do 'String'.

7) Cinco marcas separam o 'String' dos códigos

8) Os símbolos aceitos pela máquina de Turing são codificados em zero (0) e uns (1). Dêste modo representamos A como (01), B como (010), C como(011), D como (0101) e assim por diante. Os dígitos são representados zero como (0), 1 como (1), Z como (11), 3 como (111) e assim por diante.

9) O movimento contém o próximo estado, o movimento da cabeça de leitura para a direita ou esquerda e o símbolo a ser impresso, nesta ordem.

Entrada: a entrada para a UTM é uma fita de duas trilhas. A primeira delas contém a descrição da máquina de Turing a ser simulada, o 'String' e o código da sequência dos 'mappings'. A segunda trilha tem duas marcas, uma sôbre o princípio da máquina de Turing a ser simulada e outra sôbre o princípio do 'String'. Estas marcas indicam a posição da cabeça de leitura sôbre a configuração e do símbolo que está sendo examinado.

Para exemplificação, suponhamos uma máquina de Turing muito simples para reconhecer 'String' em $\{0^n1\}$ e 'String'de entrada w = 0001:

TM = $(\{q_1,q_2,q_3,q_4\}, \{0,1,B\}, \{0,1\}, \delta, \{q_1\}, \{q_4\})$, onde δ:

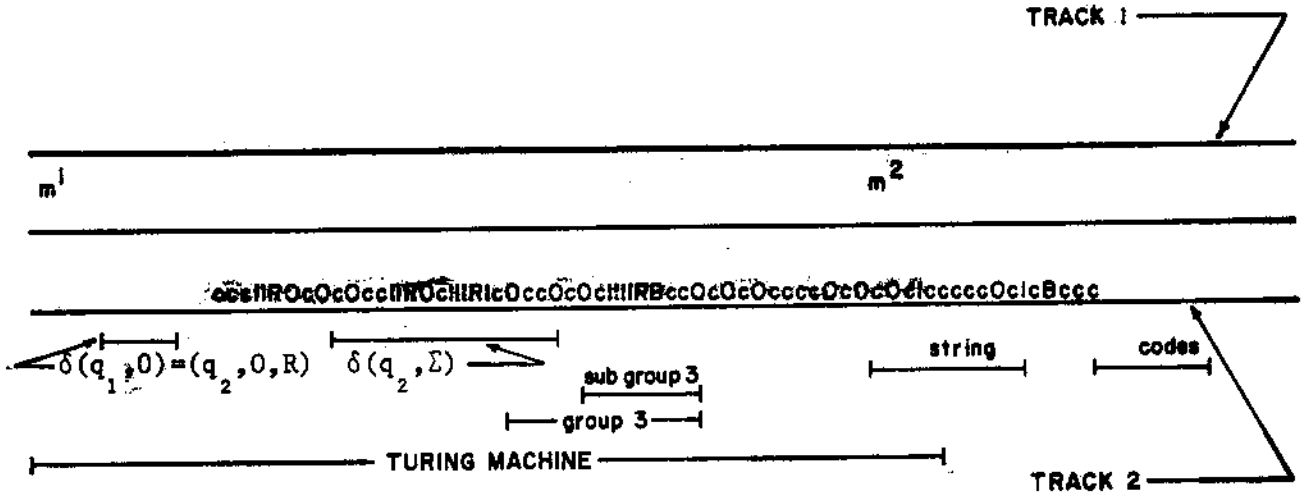
$\delta(q_1,0) = (q_2,0,R)$ - se a máquina de Turing encontrar zero no estado inicial ela entra no estado q_2 e imprime zero. Não foi definido $\delta(q_1,1)$ e $\delta(q_1,B)$.

$\delta(q_2,0) = (q_2,0,R)$ - se a máquina está no estado q_2 e encontra zero ela permanece no estado q_2 e imprime zero.

$\delta(q_2,1) = (q_3,1,R)$ - se a máquina encontra um (1) no estado q_2 ela passa ao estado q_3 e imprime um (1). Não foi definido $\delta(q_2,B)$

$\delta(q_3,B) = (q_4,B,R)$ - se a máquina encontra B no estado q_3 ela passa ao estado q_4 e imprime B. Não foi definido $\delta(q_3,0)$ e $S(q_3,1)$.

O aspecto da codificação para esta máquina de Turing será a seguinte:



5.2 - Como a UTM funciona:

A máquina universal de Turing obedece às seguintes regras:

1) Toma o número de ordem do símbolo sob m_2 e movimenta m_1, sôbre o subgrupo de mesma ordem.

2) Se o estado no subgrupo sob m_1, é zero, a UTM para.

3) Caso contrário, substitui o símbolo sob m_2 pelo símbolo sob m_1.

4) Movimenta m_2 para a esquerda ou direita e m_1 sôbre um novo grupo.

A fim de executar êstes passos a UTM usa um acumulador. A cabeça de leitura lê os 'mappings', os símbolos e a codificação, bem como imprime os símbolos na fita. Os movimentos das marcas e os testes são feitos pelo contrôle finito.

O algorítmo repete, essencialmente, êstes passos e a resposta de UTM estará no 'String' e/ou na marca m_2. A UTM para sempre que a máquina de Turing simulada entra num estado indefinido.

A UTM tem subrotinas internas para exefutar diversas funções. Assim, dividimos a UTM em parte principal e subrotinas, para melhor compreensão.

5.3 - O ALGORÍTMO

Usaremos o nome de 'string' ACUM como acumulador.

INICIALIZAÇÃO

A1) Coloque o estado inicial em ACUM.

A2) Movimente m_1 sôbre o início do grupo apontado por ACUM.

A3) Movimente m_2 sôbre o primeiro símbolo.

EM QUALQUER PONTO

B1) Tome o número de ordem do símbolo sob m_1 e coloque êste número em ACUM.

B2) Movimente m_1 sôbre o subgrupo de número de ordem dado por ACUM. Se m_1 está sôbre zero e m_2 sôbre CCCCC, imprima o 'String', a marca m_2 e pare.

B3) Execute no 'String' a substituição dada pelo subgrupo.

B4) Tome o código de movimento da cabeça de leitura (esquerda/direita) no subgrupo e movimente m_2 de acôrdo com o código.

B5) Tome o estado no subgrupo e coloque-o em ACUM. Movimente m_1 para o princípio da máquina de Turing e daí para o princípio do grupo de número de ordem dado por ACUM. Vá para B1.

6 - UM RECONHECEDOR PARA 'STRINGS' GERADOS POR GRAMÁTICAS LL(k)

Para introduzir as gramáticas LL(k) usaremos a seguinte definição:

— "Há uma classe de gramáticas livres do contexto que podem ser examinadas determinísticamente no sentido do símbolo inicial para os demais, sem voltar a examinar símbolos já examinados e tomando uma quantidade fixa de símbolos (do 'String') a frente. Estas gramáticas são denominadas LL(k) onde k é a quantidade de símbolos que se toma a frente".

Informalmente e em têrmos de exame, uma gramática LL(k) é do tipo livre do contexto e é tal que cada palavra pertencente à sua linguagem e cada produção nas suas derivações podem ser identificadas sem ambiguidade por meio de inspeção da palavra. Esta inspeção é feita desde o princípio da palavra até o seu k-ésimo símbolo a partir do início da produção. Deste modo, a determinação da produção que deve ser usada para derivar um símbolo não terminal durante o exame é dada pela porção que está sendo examinada do 'String' de entrada mais os seus k próximos símbolos.

6.1 O ALGORÍTMO PARA SIMULAR O DPDA

O reconhecedor de 'strings' gerados por uma gramática tipo LL(k) é o autômata determinístico PUSH DOWN (DPDA). Êste autômata é definido como um sistema $(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$, onde as δ transformações indicam que há sòmente um movimento para qualquer δ e o restante do sistema é definido do mesmo modo que no PDA não determinístico.

O DPDA que reconhece o conjunto de 'Strings' gerados por uma gramática do tipo LL(K) $G = (V_N, V_T, P, S)$ é -

Q - conjunto de todos os 'Strings' de comprimento k e gerados através de derivações em P e $\vdash^k$.

Σ - $\{V_T\} \cup \{\dashv\}$

Γ - $\{V_T\} \cup \{V_N\}$

q_0- são os k primeiros símbolos do 'String' de entrada

Z_0- é o símbolo inicial S

F - $\dashv^k$

onde $\dashv$ (marca final) é um terminal que não existe em V_T.

Em têrmos de implementação do algorítmo não estaremos interessados nos 'mappings'. Para trabalhar na pilha usaremos pares da forma $\{A,F^k(A,\tau)\}$ que podem ser obtidos diretamente das produções.

O algorítmo tem duas partes principais, a inicialização e a operação do DPDA. Na inicialização lemos a gramática e o 'String' de entrada (OMEGA), além de gerar todos os 'Strings' de comprimento k a partir das produções. Êstes 'Strings' são gerados e armazenados de forma que o lado esquerdo da regra aponta para o seu lado direito e para todos os 'Strings' gerados a partir dela.

A geração dos 'Strings' é feita tomando-se o lado direito da regra e escrevendo-se todas as suas possíveis derivações. A seguir substituimos as variáveis (caminhando da esquerda para a direita) pelos terminais de acôrdo com as regras até que o comprimento do 'String' seja k. Se o comprimento do 'String' assim formado fôr menor que k, adicionamos marcas finais($\dashv$) até que o comprimento k seja atingido. Entre os strings colocamos um delimitador qualquer.

A operação do DPDA tem três partes principais -

1) Operações na pilha - colocamos e removemos pares do tipo $\{A,F^k(A,\tau)\}$ onde $F^k(A,\tau)$ são os 'Strings' de comprimento k que começam por A e seguem ao 'String' τ. Ao remover um par do tôpo da pilha é necessário analisá-lo.

2) Operações se A é um não terminal - testamos se o estado pertence ao $\{L(A)F^k(A,\tau)\}$, obtendo a regra através de A e formando novos pares de acôrdo com a regra.

3) Operações se A é um terminal - testamos se A é igual ao símbolo examinado, trocamos o estado e movemos a cabeça de leitura do DPDA.

6.2 - O ALGORÍTMO

Usaremos os seguintes nomes:

VARK - valor de k

VARIABLE - conjunto de variáveis

TERMINAL - conjunto de terminais

VARA, VARB, LEVELA,LEVELB - nomes de 'Strings' de trabalho

COUNTERA,COUNTERB - contadores

RULEA, RULEB - nomes de 'Strings' para as regras

VARR - apontados de posição para a cabeça de leitura

OMEGA - nome do 'String' de entrada

STATE - contém o estado atual da máquina

INICIALIZAÇÃO

A1) Leia o 'String' a ser testado. Se a gramática já foi lida vá para C1.

A2) Leia a gramática e VARK.

PRODUÇÃO DOS 'STRINGS' DE COMPRIMENTO K A PARTIR DAS REGRAS

B1) Retire um símbolo de VARIABLE, coloque-o em VARA e faça COUNTERA ← 1. Se falhar vá para C1.

B2) Concatene VARA e COUNTERA em RULEA. Da regra apontada por RULEA tome VARK símbolos e o delimitador, colocando este grupo em LEVELA. Se RULEA for vazio vá para B1.

B3) Retire o primeiro grupo de LEVELA e decomponha-o em VARX1, VARX2,..... VARXN. Se falhar e LEVELB for vazio, faça COUNTERA ← COUNTERA + 1 e vá para B2. Se LEVELB não for vazio, faça LEVELA ← LEVELB e vá para B3.

B4) Tome uma decomposição ($VARX_i$) e se êla pertence a TERMINAL, vá para B4. Se não houver mais decomposição vá para B3. Em qualquer outro caso faça COUNTERB ← 1.

B5) Concatene a decomposição e COUNTERB e coloque a regra apontada por esta decomposição em RULEB. Concatene todas as decomposições, exceto $VARX_i$, com RULEB e retire os primeios VARK símbolos desta concatenação. Se RULEB for vazio, vá para B4.

B6) Se todos os VARK elementos do 'String' formado pertencem a TERMINAL, coloque-os no conjunto apontado por VARA ($VARA) desde que êstes elementos já não pertençam a este conjunto. Se o comprimento do 'String' formado é menor que VARK complete-o com marcas finais (—|). Se os VARK elementos já pertencem ao conjunto, aumente COUNTERB de uma unidade e vá para B5.

B7) Se o 'String' formado (os VARK elementos) tem pelo menos um elemento pertencente a VARIABLE e o 'String' não faz parte de LEVELA, coloque-o em LEVELB. Em caso contrário aumente COUNTERB de uma unidade e vá para B5.

INICIALIZAÇÃO DO DPDA

C1) Coloque o símbolo inicial no tôpo da pilha, faça VAR ← 0, concatene OMEGA

com VARK marcas finais em OMEGA e tome como estado os primeiros VARK símbolos de OMEGA, colocando-os em STATE.

REMOÇÃO E ANÁLISE DO TÔPO DA PILHA

D1) Retire o tôpo da pilha e coloque o primeiro símbolo em VARA e o restante em VARB. Se VARA pertence a TERMINAL e se o símbolo de ordem VARR + 1 é igual a VARA faça VARR ← VARR + 1; coloque os símbolos (de OMEGA) de ordem (VARR + 2),....,(VARR + VARK + 1) em STATE e vá para D1. Se o símbolo de ordem VARR + 1 for diferente de VARA, rejeite OMEGA.

D2) Se a pilha for vazia e STATE igual a VARK marcas finais aceite OMEGA. Se for diferente de VARK marcas finais, rejeite-o.

ARMAZENAMENTO NO TÔPO DA PILHA

E1) Concatene cada um dos 'Strings' de VARB com cada um dos 'Strings' apontados por VARA. Quando encontrar uma concatenação igual a STATE vá para E2. Caso contrário rejeite OMEGA.

E2) Tome a regra apontada por VARA, decomponha-a em VARX1, VARX2,...,VARXN elementos e faça VARGN ← VARB, onde N é o número de ordem de decomposição(o N de VARXN).

E3) Tome todos os 'Strings' apontados por VARA e concatene cada um deles com VARGN. Coloque estas contatenações em VARG(N-1).

E4) Coloque no tôpo da pilha o par VARXN e VARGN, decremente N de unidade e volte a E3 se N ≠ 1. Em caso contrário vá para D1.

B I B L I O G R A F I A

(1) FORMAL LANGUAGE AND THEIR RELATION TO AUTOMATA, HOPCROFT AND ULLMAN. ADDISON WESLEY, 1969

(2) ON THE USE OF PUSH DOWN AUTOMATA FOR THE PARSING OF SENTENCES ON CONTEXT FREE LANGUAGE, R. L. DE CARVALHO

SERIES: MONOGRAPHS IN COMPUTER SCIENCE AND COMPUTER APPLICATIONS

CENTRO TECNICO CIENTÍFICO - PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

(3) PROPERTIES OF DETERMINISTIC TOP DOWN GRAMMARS

D. J. ROSENKRANTZ and R.E. STEARNS

INFORMATION AND CONTROL VOL 17,1970 pgs 226-250

(4) SNOBOL 3 PROGRAMMING LANGUAGE

D. J. FARBER, R.E. GRISWOLD and I.P. POLONSKY

THE BELL SYSTEM TECHNICAL JOURNAL, JULY-AUGUST 1966

(5) MANUAL DE SNOBOL 3

A.T. DE MEDEIROS e G. SCHWACHHEIM

NOTAS TÉCNICAS - CENTRO BRASILEIRO DE PESQUISAS FÍSICAS - VOL.II Nº 2

(6) PROGRAMMING SYSTEMS AND LANGUAGE

SAUL ROSEN, EDITOR

McGRAW HILL, 1967