Resumo - Introdução a Teoria da Computação

Flávio Avad Briz

7 de novembro de 2012

Sumário

Capítulo 1

Linguagens Regulares

1.1 Autômatos Finitos

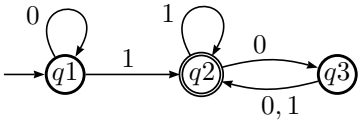


Figura 1.1: Exemplo de um *diagrama de estado* de um AFD chamado M_1

1.1.1 Definição Formal

Um **autômato finito** é uma 5-nupla $(Q, \Sigma, \delta, q_0, F)$ onde:

1. Q é um conjunto de **estados** finito;
2. Σ é um conjunto finito chamado de **alfabeto**;
3. $\delta : Q \otimes \Sigma \to Q$ é a **função de transição**
4. $q_0 \in Q$ é o **estado inicial**
5. $F \subseteq Q$ é o **conjunto dos estados de aceitação**

Exemplo: Descrição formal de $M_1 = (Q, \Sigma, \delta, q_0, F)$ onde

1. $Q = \{q1, q2, q3\}$;
2. $\Sigma = \{0, 1\}$;
3. $\delta : Q \otimes \Sigma \to Q$

	0	1
q1	q1	q2
q2	q3	q2
q3	q2	q2

4. $q_0 = \{q1\}$;
5. $F = \{q2\}$;

1.1.2 Linguagem de Máquina

Seja A o conjunto de todas as cadeias que a máquina M dizemos que A é a **linguagem da máquina** M, formalmente $L(M) = A$.

Quer dizer que M **reconhece**/**aceita** A

1.2 Definição Formal de Computação

Seja $M = \{Q, \Sigma, \delta, q_0, F\}$ um autômato finito e suponha $w = w_1 w_2 w_3 ... w_n$ seja uma cadeia onde cada $w_i \in \Sigma$. Então M **aceita** w se existe uma sequência de estados $r_0, r_1, r_2, ..., r_n$ em Q com três condições:

1. $r_0 = q_0$;

2. $\delta(r_i, w_{i+1}) = r_{i+1}$ para $i = 1, 2, ..., n-1$;

3. $r_i \in F$;

Dizemos que M reconhece a linguagem A
se $A = \{w | M \ \ aceita \ \ w\}$

1.2.1 Linguagem Regular

Uma linguagem é chamada de uma ***linguagem regular*** se algum autômato finito a reconhece.

1.3 Projetando Autômatos Finitos

Desejamos contruir um autômato que reconheça a linguagem que consiste de todas as cadeias com um número ímpar de *1*, supondo que o alfabeto seja $\{0, 1\}$ e os estados possíveis $Q = \{qPar, qImpar\}$.

1. Primeiramente desenhe os estados

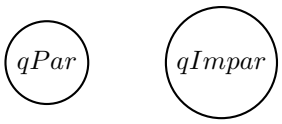


Figura 1.2: Os dois estados.

2. Em seguida desenhe as **transições de estado**.

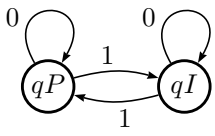


Figura 1.3: Transições dizendo como as possibilidades se rearranjam

3. Adicione o estado **inicial** e de **aceitação**.

1.4 As Operações Regulares

Sejam A e B linguagens. Definimos as operações de **união**,**concatenação** e **estrela** da seguinte forma.

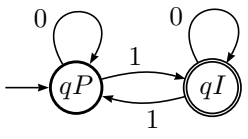


Figura 1.4: Estados inicial e de aceitação adicionados

- **União:** $A \cup B = \{x | x \in A \text{ ou } x \in B\}$;
- **Concatenação:** $A \circ B = \{xy | x \in A \text{ e } y \in B\}$;
- **Estrela:** $A^* = \{x_1 x_2 ... x_k | k \geq 0 \text{ e cada } x_i \in A\}$;

As operações **união** e **concatenação** são **binárias**, já a operação **estrela** é **unária**.

Na operação *estrela* não podemos esquecer que ε está sempre presente.

1.5 Não-Determinismo

Em uma máquina ***não-determinística*** podem existir várias escolhas para o próximo estado em qualquer ponto.

Autômatos finitos não-determinísticos são uma genarização de determinísticos, portanto todo AFD também é um AFN.

A computação de um AFN consiste em gerar diversas cópias da máquina quando existir uma transição não-determinística, e verificar se em algumas dessas máquinas chega à algum estado de aceitação. Em caso de positivo de alguma dessas máquinas o último estado seja o de aceitação, então a cadeia é reconhecida pelo AFN.

1.5.1 Definição Formal de um AFN

Um **autômato finito não-determinístico** é uma 5-nupla $(Q, \Sigma, \delta, q_0, F)$ onde:

1. Q é um conjunto de **estados** finito;
2. Σ é um conjunto finito chamado de **alfabeto**;
3. $P(Q)$ é a coleção de todos os subconjuntos de Q, chamado de **conjunto das partes**;
4. $\delta : Q \otimes \Sigma \to P(Q)$ é a **função de transição**
5. $q_0 \in Q$ é o **estado inicial**
6. $F \subseteq Q$ é o **conjunto dos estados de aceitação**

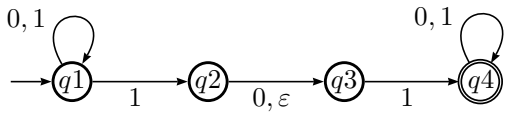


Figura 1.5: Diagrama de estados de N_1

Exemplo: Autômato N_1
Descrição formal de $N_1 = (Q, \Sigma, \delta, q_0, F)$ onde

1. $Q = \{q1, q2, q3, q4\}$;
2. $\Sigma = \{0, 1\}$;
3. $\delta : Q \otimes \Sigma \rightarrow P(Q)$

	0	1	ε
q1	{q1}	{q1,q2}	Ø
q2	{q3}	Ø	{q3}
q3	Ø	{q4}	Ø
q3	{q4}	{q4}	Ø

4. $q_0 = \{q1\}$;
5. $F = \{q4\}$;

1.6 Equivalência entre AFNs e AFDs

Todo autômato finito **não-determinístico** tem um autômato finito **determinístico** equivalente.

Corolário 1 *Uma linguagem é regular se e somente se um AFN a reconhece.*

1.6.1 Exemplo de uma conversão de um AFD para um AFN

Vamos tentar encontrar um AFD D equivalente ao AFN a seguir:

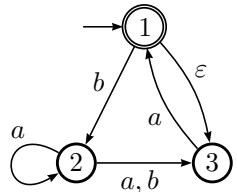


Figura 1.6: Estados inicial e de aceitação adicionados

1. Primeiramente determinamos os estados de D, que vão ser iguais a $Q = P(1, 2, 3)$, portanto:

$$\{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$$

2. Determinamos os estados inicial e de aceitação. O estado inicial é igual ao próprio estado inicial do AFD, mais os outros estados atingíveis a partir dele por ε, ou seja $q_0 = 1, 3$.

Os estados de aceitação possíveis são aqueles que contêm o estado de aceitação do AFN, portanto $F = \{\{1\}, \{1,2\}, \{1,3\}, \{1,2,3\}\}$

3. Adicionamos as transições. Para fazer isso precisamos analizar quais transições ocorerriam para cada estado separadamente e depois realizamos a união desses resultados, por exemplo no caso do estado 1,2,3 recebesse a, o estado resultando seria $\delta^{'}(1,2,3,a) = \delta(1,a) \bigcup \delta(2,a)\delta(3,a) = \emptyset \bigcup \{2,3\} \bigcup \{1\} = \{1,2,3\}$

	a	b	ε
$\emptyset$	$\emptyset$	$\emptyset$	
{1}	$\emptyset$	{2}	
{2}	{2,3}	{3}	
{3}	{1,3}	$\emptyset$	$\emptyset$
{1,2}	{2,3}	{2,3}	
{1,3}	{1,3}		
{2,3}	{1,2,3}	{3}	
{1,2,3}	{1,2,3}	{2,3}	

1.7 Fecho sob as Operações Regulares

A classe das **linguagem regulares** é fechada sobre qualquer *operação regular*.

1.8 Expressões Regulares

Assim como na aritimética podemos montar expressões utilizando os operandores de soma(+) e mutiplicação(**x**), também podemos montar *expressões regulares* utilizando *operações regulares*.

A criação de operações regulares consiste em criar uma equivvalência entre a expressão e o conjunto representado por ela. O alfabeto $\Sigma = \{0,1\}$ poderia ser escrito como $0 \bigcup 1$

1.8.1 Definição Formal de uma Expressão Regular

R é uma expressão regular se R for:

1. a para algum $a \in \Sigma$,

2. ε,

3. $\emptyset$,

4. $(R_1 \bigcup R_2)$ onde R_1 e R_2 são expressões regulares,

5. $(R_1 \circ R_2)$ onde R_1 e R_2 são expressões regulares,

6. (R_1^*) onde R_1 é uma expressão regular,

As espressões regulares ε e $\emptyset$ são **DIFERENTES**, ε representa uma linguagem contendo somente a cadeia vazia ε e $\emptyset$ representa uma linguagem que não contém nenhuma cadeia.

Exemplos:

1. 0^*10^*=$\{w|w$ contém um único $1\}$
2. $\Sigma^*1\Sigma^*$=$\{w|w$ tem pelo menos um símbolo $1\}$
3. $\Sigma^*001\Sigma^*$=$\{w|w$ tem pelo menos uma cadeia 001 como subcadeia$\}$
4. $1^*(01^*)^*$=$\{w|$ todo 0 em w é seguido por pelo menos um símbolo $1\}$
5. $\Sigma\Sigma^*$ =$\{w|w$ é uma cadeia de comprimento par$\}$
6. $\Sigma\Sigma\Sigma^*$ =$\{w|w$ é uma cadeia de comprimento múltiplo de 3$\}$

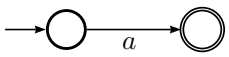
1.8.2 Equivalência com Autômatos Finitos

Qualquer expressão regular pode ser reconhecida por um autômato finito que reconhece a linguagem regular e vice-versa.

Teorema 1 *Uma linguagem é regular se e somente se alguma expressão regular reconhece ela.*

Prova Conversão de R em um AFN, considerando os seis casos da definição:

1. $R = a$ para algum a em Σ. Então $L(R) = \{a\}$ e o seguinte AFN reconhece *L(R)*:

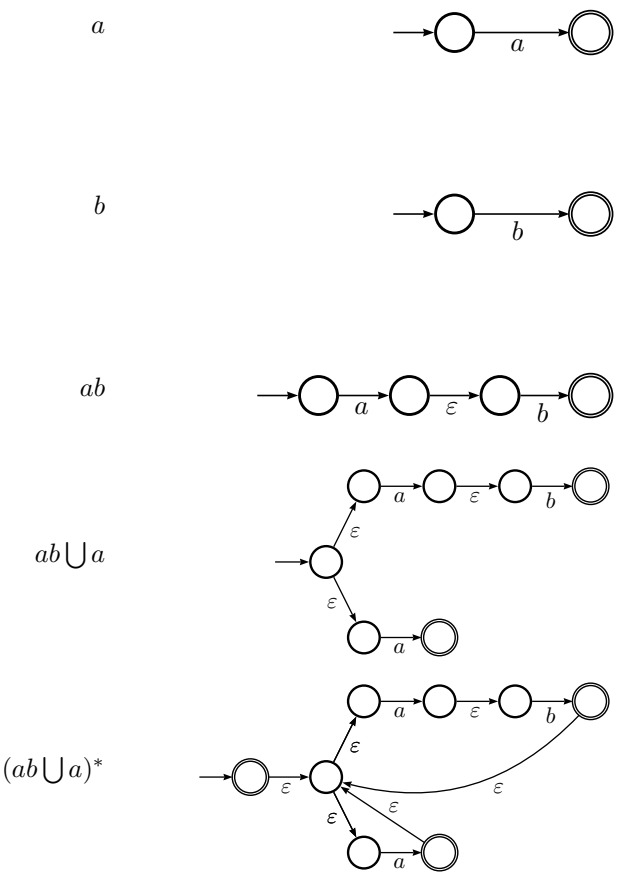


2. $R = \varepsilon$. Então $L(R) = \{\varepsilon\}$ e o seguinte AFN reconhece *L(R)*:

3. $R = \emptyset$. Então $L(R) = \emptyset$ e o seguinte AFN reconhece *L(R)*:

4. $R = R_1 \bigcup R_2$
5. $R = R_1 \circ R_2$
6. $R = R_1^*$

Exemplo: $(ab \bigcup a)^*$



Lema 1 *Se uma linguagem é regular, então ela é descrita por uma expressão regular.*

Prova Dividimos esse procedimento em duas partes, usando um novo tipo de autômato finito chamado autômato finito não-determinístico generalizado, AFNG. Primeiro, mostramos como converter AFDs em AFNGs, e depois AFNGs em expressões regulares.

Por conveniência, requeremos que os AFNGs tenham sempre um formato especial que atenda às seguintes condições:

- O estado inicial tem setas de transição saindo para todos os outros estados, mas nenhuma seta chegando de qualquer outro estado.
- Existe apenas um estado de aceitação, e ele tem setas chegando de todos os outros estados, mas nenhuma seta saindo para qualquer outro estado. Além disso, o estado de aceitação não é o mesmo que o estado inicial.

- Com exceção dos estados inicial e de aceitação, uma seta sai de cada estado para todos os outros e também de cada estado para ele mesmo.

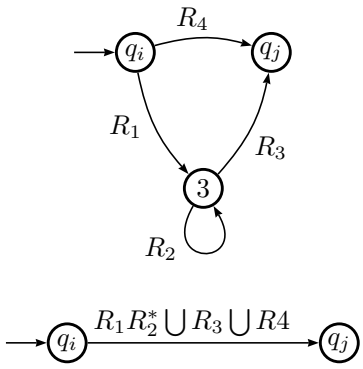


Figura 1.7: AFNG com um estado a menos

1.8.3 Definição Formal de um AFNG

Um **autômato finito não-determinístico generalizado** é uma 5-nupla $(Q, \Sigma, \delta, q_{inicio}, q_{aceita})$ onde:

1. Q é um conjunto de **estados** finito;
2. Σ é um conjunto finito chamado de **alfabeto**;
3. $\delta : (Q - \{q_{aceita}\}) \otimes (Q - \{q_{inicio}\}) \rightarrow R$ é a **função de transição**
4. $q_{inicio} \in Q$ é o **estado inicial**
5. $q_{aceita} Q$ é o **estado de aceitação**

1.9 Lema do bombeamento

Implementar...

Capítulo 2

Linguagens livres-do-contexto

É uma maneira muito mais poderosa de descrever linguagens, sendo primeiramente utilizadas no estudo de linguagens humanas.

Uma aplicação importante de gramáticas livres-do-contexto ocorre na especificação e compilação de linguagens de programação.

A colecao de linguagens associadas com gramáticas *livres-do-contexto* são denominadas ***linguagens livres-do-contexto***. São denominadas as máquinas que reconhecem linguagens livres-do-contexto de ***autômatos-com-pilha***.

2.1 Gramáticas livres-do-contexto

Exemplo de uma gramáticas livres-do-contexto chamada G_1:

$A \rightarrow 0A1$
$A \rightarrow B$
$B \rightarrow \#$

A mesma gramática pode ser abreviada e representada por:

$A \rightarrow 0A1|B$
$B \rightarrow \#$

Uma gramática consiste de ***regras de substituição*** , ou ***produções*** . Cada regra é representada por uma linha da gramática, os símbolos em letra maiúscula são chamados de ***variáveis*** , e os ***terminais*** são análogos ao alfabeto de entrada e são geralmente representados por letras minúsculas.

A ***variável inicial*** é representada pelo primeiro símbolo da primeira regra.

Portanto a gramática G_1 é representada pelas variáveis A e B, onde A é a variável inicial e possui terminais *0,1 e #*.

Como usar a gramática:

- **1.** Escreva a variável inicial.
- **2.** Encontre uma variável e substitual ela pelo lado direito da regra.
- **3.** Repita o processo 2 até que não reste nenhuma variável.

A sequência de substituições é chamada de ***derivação*** . Um exemplo de uma possível derivação 000#111 em G_1 é

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

Uma forma de representar mais visualmente a derivação é através da ***árvore sintática***

O conjunto de todas as cadeias geradas a partir da variável inicial é chamada de ***linguagem de uma gramática*** $(L(G_i))$.

A liguagem da gramática G_1 é $\{0^n\#1^n|n > 0\}$.

Toda linguagem gerada por alguma gramática livre-de-contexto é chamada de ***linguagem livre-de-contexto*** (LLC).

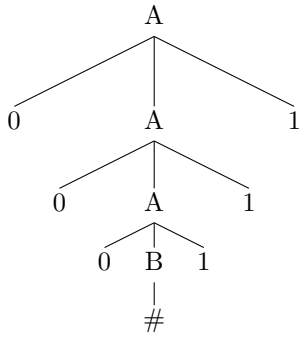


Figura 2.1: Arvore sintática para 000#111 na gramática G_1

2.1.1 Definição Formal de uma Gramática Livre-de-Contexto

Uma **gramática livre-de-contexto** (GLC) é uma 4-upla (V,Σ,R,S) onde:

- **1.** V é um conjunto finito denonimado de as **variáveis**;
- **2.** Σ é um conjunto finito, disjunto de V , denonimado de os **terminais**;
- **3.** R é um conjunto finito de **regras**, com cada regra sendo uma variável e uma cadeia de variáveis e terminais;
- **4.** $S \in V$ é a **variável inicial**;

Se u, v e w são cadeias de variáveis e terminais, e $A \to w$ é uma regra da gramática, dizemos que uAv **origina** uwv, escrito $uAv \to uwv$. Digamos que u **deriva** v, escrito $u \overset{*}{\Rightarrow} v$, se $u = v$ ou se uma sequencia $u_1, u_2, \ldots, u_k$ existe para $k \geq 0$ e

$$u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k$$

A **linguagem da gramática** é $\{w \in \Sigma^* | S \overset{*}{\Rightarrow} w\}$.

2.1.2 Projetando Gramáticas Livres-de-Contexto

- **1.** Muitas LLCs pode ser consideradas como a união de outras LLCs mais simples, portanto para gerar uma GLC para um LLC é aconselhável dividir o problema em partes menores e combinar as regras obtidas para os LLCs mais simples em um nova regra $S \Rightarrow S_1|S_2|\ldots|S_n$, onde as variáveis S_i são as variáveis iniciais para as gramáticas individuais.

  Exemplo: Obtenha a gramática para $\{0^n1^n|n \geq 0\} \cup \{1^n0^n|n \geq 0\}$

  - Primeiramente obtemos a gramática de $\{0^n1^n|n \geq 0\}$
    $S_1 \to 0S_11|\varepsilon$
  - Depois obtemos a gramática de $\{1^n0^n|n \geq 0\}$
    $S_2 \to 1S_20|\varepsilon$

– Para unir as duas gramáticas basta utilizar $S \rightarrow S_1|S_2$, a gramática final fica:

$S \rightarrow S_1|S_2$
$S_1 \rightarrow 0S_11|\varepsilon$
$S_2 \rightarrow 1S_20|\varepsilon$

- **2.** Caso a linguagem seja regular, pode ser construído primeiramente o AFD e convertê-lo em um GLC da seguinte maneira:

  Crie uma variável R_i para cada estado q_i;

  Adicione a regra $R_i \rightarrow aR_j$, se $\delta(q_i, a) = q_j$;

  Adicione a regra $R_i \rightarrow \varepsilon$ se q_i for um estado de aceitação;

  Faça R_0 a variável inicial do GLC se q_0 do estado inicial do AFD;

- **3.** No caso em que a linguagem apresentar subcadeias *ligadas*, de forma que uma máquina que reconheça essa linguagem teria de armazenar uma quantidade ilimitada de informação sobre cada uma das subcadeias como ocorre em $\{0^n1^n|n \geq 0\}$, onde o número de 0s tem q ser igual ao número de 1s. Voce pode construir uma GLC para lidar com essa situação usando uma regra da forma $R \rightarrow uRv$, que gera cadeias nas quais a parte contendo os u's corresponde a parte contendo os v's.

- **4.** No caso mais complexo as cadeias podem conter certas estruturas que aparecem recursivamente como parte de outras(ou da mesma) estrutura. Para resolver o problema colocamos o símbolo da variável que gera a estrutura na regra, a onde pode ocorrer recursão.

2.1.3 Ambiguidade

Ocorre quando uma cadeia pode ter mais de uma árvore sintática. Se uma gramática gera a mesma cadeia de várias maneiras diferentes, dizemos a que a cadeia é *derivada ambiguamente* nessa gramatica. Se uma gramática gera alguma cadeia ambiguamente dizemos que a gramática e *ambígua*.

Para verificar se existe ambiguidade é analizada a **derivação mais a esquerda**, se a cada passo a variável remanescente for igual à que deve ser substituída, pode existir ambiguidade.

Definição 1 *Uma cadeia w é derivada **ambiguamente** na gramática livre-do-contexto G se ela tem duas ou mais derivações mais a esquerda diferentes. A gramática G é **ambígua** se ela gera alguma cadeia ambiguamente.*

Para uma gramática ambígua podem existir outras gramáticas não ambíguas que geram a mesma linguagem.

Algumas LLCs podem ser geradas somente por gramáticas ambíguas, essas são chamadas de **inerentemente ambíguas**.

2.1.4 Forma Normal de Chomsky

Uma gramática livre-do-contexto está na **forma normal de Chomsky** se toda regra é da forma:

$$A \to BC$$
$$A \to a$$

Onde a e qualquer terminal e A, B e C são quaisquer variáeis exceto que B e C não podem ser a variável inicial. Adicionalmente, permitimos a regra $S \to \varepsilon$, onde S e a variável inicial.

Teorema 2 *Qualquer linguagem livre-do-contexto é gerada por uma gramática livre-do-contexto na forma normal de Chomsky.*

Ideia da Prova: Primeiramente adicionamos uma nova variável inicial, em seguida eliminamos todas as **regras** ε da forma $A \to \varepsilon$, depois são eliminadas as **regras unitárias** da forma $A \to B$ e finalmente convertemos as regras remanescentes de forma apropriada.

Prova:

- **1.** Adicionamos uma nova variável inicial S_0 e a regra $S_0 \to S$, onde S era a variável inicial original.
- **2.** Remoção das **regras** ε, onde as variáveis A, que não sejam a variável inicial. Para cada regra que contenha A do lado direito, *adicionamos uma nova regra com essa ocorrência apagada*, por exemplo para $R \to uAv$ adicionaríamos:

  $R \to uv$

  No caso $R \to uAvAw$ adicionaríamos:

  $R \to uvw$

  $R \to uAvw$

  $R \to uvAw$

  Caso a regra seja $R \to A$ adicionamos:

  $R \to \varepsilon$, somente no caso de não tenha sido removido anteriormente.
- **3.** Remoção das **regras unitárias**. Removemos $A \to B$, então sempre que existir um $B \to u$, adicionamos $A \to u$ a menos que tenha sido uma regra unitária removida anteriormente.
- **4.** Substituir cada regra $A \to u_1u_2 \ldots u_k$, onde $k \geq 3$ e cada u_i é uma variável ou símbolo terminal com as regras $A \to u_1A_1$,$A \to u_2A_2$,...,$A \to u_kA_k$. Os $A_i's$ são novas variáveis.

  Se $k = 2$, substituir qualquer u_i por uma nova variável U_i e adicionar a regra $U_i \to u_i$.

Exemplo: A GLC original G a ser transformada para a forma normal é:

$S \to ASA|aB$

$A \to B|S|\varepsilon$

$B \to b|\varepsilon$

Todas as mudanças de adição de novos símbolos realizadas estão em **azul** e a variáveis ou terminal à serem removidas em **cinza**.

- **1.** Adicionamos primeiramente a nova variável inicial S_0:

  $\mathbf{S_0 \to S}$

  $S \to ASA|aB$

  $A \to B|S$

  $B \to b|\varepsilon$

- **2.** Agora é necessário remover as regras ε com destaque em **cinza**, nesse caso a variável a ser analizada é B :

  $S_0 \to S$

  $S \to ASA|aB|\mathbf{a}$

  $A \to B|S|\varepsilon$

  $B \to b|\varepsilon$

  $S_0 \to S$

  $S \to ASA|aB|a|\mathbf{AS}|\mathbf{SA}|\mathbf{S}$

  $A \to B|S|\varepsilon$

  $B \to b$

- **3.** Removendo as regras unitárias (somente B e S nesse caso) temos:

  $S_0 \to \mathbf{S}|\mathbf{ASA}|\mathbf{aB}|\mathbf{a}|\mathbf{AS}|\mathbf{SA}$

  $S \to ASA|aB|a|AS|SA|\mathbf{S}$

  $A \to B|\mathbf{S}|\mathbf{ASA}|\mathbf{aB}|\mathbf{a}|\mathbf{AS}|\mathbf{SA}$

  $B \to b$

  $S_0 \to ASA|aB|a|AS|SA$

  $S \to ASA|aB|a|AS|SA$

  $A \to \mathbf{B}|\mathbf{b}|ASA|aB|a|AS|SA$

  $B \to b$

- **4.** Converter regras remanescentes substituindo AS por A_1 e AS por A_1 e a por U

  $S_0 \to S|\mathbf{A_1}A|\mathbf{U}B|a|AS|SA$

  $S \to \mathbf{A_1}A|\mathbf{U}B|a|AS|SA$

  $A \to b|\mathbf{A_1}A|\mathbf{U}B|a|AS|SA$

  $B \to b$

  $\mathbf{A_1 \to AS}$

  $\mathbf{U \to a}$

2.2 Autômato com Pilha

São como AFDs, mas com um componente extra chamado **pilha**. Um autômato com pilha :

- Reconhecer algumas linguagens não-regulares.
- É equivalente a gramáticas de livre-contexto.
- determinístico e não-determinístico não são equivalentes em poder.
- não-determinístico pode reconhecer certas linguagens que o determinístico não pode.

2.2.1 Definição Formal de um Autômato com Pilha

Um autômato com pilha e uma 6-upla (Q, Σ, Γ, δ, q_0 , F), onde (Q, Σ, Γ, δ, q_0 e F são todos conjuntos finitos, e:

- **1.** Q é o conjunto de estados;
- **2.** Σ é o alfabeto de entrada;
- **3.** Γ é o alfabeto de pilha;
- **4.** $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow P(Q \times \Gamma_\varepsilon)$ é a função de transição;
- **5.** $q_0 \in Q$ é o estado inicial;
- **6.** $F \subseteq Q$ é o conjunto de estados de aceitação;

Um autômato com pilha $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ satisfaz as seguintes suposições:

- **1.** $r_0 = q_0$ e $s_0 = \varepsilon$. Essa condição significa que M inicia apropriadamente, no estado inicial e com uma pilha vazia.
- **2.** Para $i = 0, \ldots, m-1$ temos $(r_i + 1, b) \in \delta(r_i, w_{i+1}, a)$, onde $s_i = at$ e $s_{i+1} = bt$ para algum $a, b \in \Gamma_\varepsilon$ e $t \in \Gamma^*$. Essa condicao afirma que M se
- **3.** $r_m \in F$. Essa condição afirma que um estado de aceitacao ocorre ao final da entrada.

Exemplo: O que se segue e a descricao formal do AP que reconhece a linguagem $0^n 1^n | n \geq 0$. Suponha que M_1 seja $(Q, \Sigma, \Gamma, \delta, q_1, F)$, onde:

- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, \$\}$
- $F = \{q_1, q_4\}$
- δ é dado pela tabela:

Input:	0			1			ε		
Pilha:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

M_1 também pode ser representado por um diagrama de estados,e sua pilha é representada da seguinte maneira: "$a, b \to c$" significa que ao ler um símbolo a da entrada, pode ser substituído no topo da pilha b por um símbolo c.

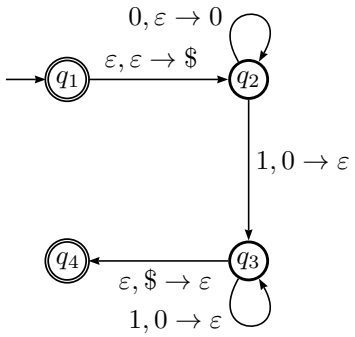


Figura 2.2: Diagrama de estados para o AP M_1 que reconhece $\{0^n 1^n | n \geq 0\}$

2.3 Equivalência com gramáticas livres-do-contexto

Teorema 3 *Uma linguagem é livre do contexto se e somente se algum autômato com pilha a reconhece.*

Lema 2 (converter GLC em um AP equivalente) *Se uma linguagem é livre do contexto, então algum autômato com pilha a reconhece.*

Ideia da prova Sabemos que existe para a *LLC* uma *GLC*, G que a gera. Mostramos como converter G em um autômato a pilha equivalente AP. É um pouco difícil de imaginar o funcionamento do autômato, mas criando ele de forma não-determinística, fico muito mais fácil, pois sempre que empilhamos, desempillhamos em seguida na transição não-determinística.

Prova Na construção do autômato $P = (Q, \Sigma, \Gamma, \delta, q_1, F)$, utilizamos a seguinte notação de transição $(r, u) \in \delta(q, a, s)$ para dizer que quando q é o estado atual, a próximo símbolo a ser lido, s o símbolo no topo da pilha, que irá ocorrer uma transição para o estado r e uma substituíção de s por u (*pode conter mais de uma váriavel ou terminal, para abreviação*) no topo da pilha.

Os estados de P são $\{q_{inicio}, q_{loop}, q_{aceita}\} \bigcup E$, onde E são os estados que precisamos implementar as abreviações. O estado inicial é q_{inicio}, e o único estado de aceitação é q_{aceita}.

A funcao de transição é definida da seguinte forma.

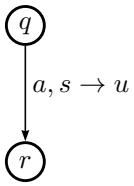


Figura 2.3: Exemplo da transição $\delta(q, a, s) = \{(r, u)\}$

- **1.** Começamos inicializando a pilha para conter os símbolos \$ e S, portanto temos inicialmente $\delta(q_{inicio}, \varepsilon, \varepsilon) = \{(q_{loop}, S\$)\}$.
- **2.a** No caso de contermos no topo da pilha uma *variável* A por exemplo, adicionamos a transição $\delta(q_{loop}, \varepsilon, A) = \{\delta(q_{loop}, w)|$ onde $A \to w$ é uma regra de $R.\}$.
- **2.b** No caso de contermos no topo da pilha um símbolo *terminal* a por exemplo, adicionamos a transição $\delta(q_{loop}, \varepsilon, a) = \{\delta(q_{loop}, \varepsilon)\}$.
- **2.c** No caso de contermos no topo da pilha o símbolo \$ que serve de marcador para indicar que a pilha está vazia, adicionamos $\delta(q_{loop}, \varepsilon, \$) = \{\delta(q_{aceita}, \varepsilon)\}$.

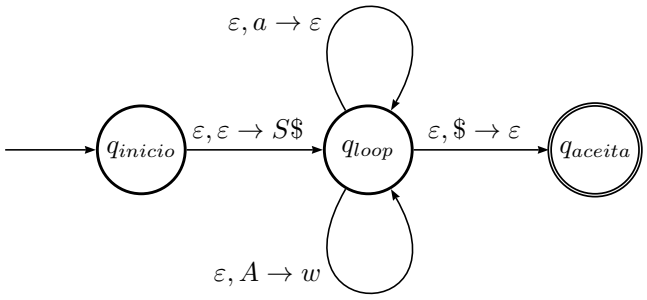


Figura 2.4: Exemplo da transição $\delta(q, a, s) = \{(r, u)\}$

Lema 3 (converter GLC em um AP equivalente) *Se um autômato com pilha reconhece alguma linguagem, então ela é livre-do-contexto.*

Ideia da prova Substituir o intervalo entre um empilhamento (no estado p) e um desempillhamento (no estado q), em que seja respeitada a condição de que a pilha deve ser a mesma em ambos os estados considerados, por uma váriavel (A_{pq}). Tentamos dessa maneira gerar todas as cadeias de uma gramática G a partir de um autômato à pilha P.

Os casos que podem ocorrer sobre a leitura de uma cadeia x são que ou o símbolo empilhado seja o mesmo a ser desempillhado no final, ou não. Obrigatoriamente a pilha inicia e termina vazia. Caso ocorra de o símbolo desempillhado no estado final q seja o mesmo empilhado no início p, adicionamos a regra $A_{pq} \to aA_{rs}b$, onde a é a entrada lida no primeiro movimento, b a entrada lida no último movimento, r é o estado seguinte a p, e s o estado anterior a q. No caso de o símbolo desempillhado seja diferente do empilhado inicialmente adicionamos a regra $A_{pq} \to A_{pr}A_{rq}$.

Prova Sendo $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{aceita}\})$, vamos construir G. A váriavel inicial de G é $A_{q_{inicio}q_{aceita}}$ e suas regras são:

- Para cada $p, q, r, s \in Q$, $t \in \Gamma$ e $a, b \in \Sigma_\varepsilon$ se $\delta(p, a, \varepsilon) = \{(r, t)\}$ e $\delta(q, b, t) = \{(q, \varepsilon)\}$, adicionar a regra $A_{pq} \to aA_{rs}b$ em G.
- Para cada $p, q, r \in Q$, adicionar a regra $A_{pq} \to A_{pr}A_{rs}$ em G.
- Para cada $p \in Q$, adicionar a regra $A_{pp} \to \varepsilon$ em G.

2.4 Lema do bombeamento para linguagens livres-de-contexto

Se A é uma linguagem livre-de-contexto, então existe um número p (comprimento do bombeamento) onde, se s é uma cadeia qualquer em A de comprimento pelo menos p, s pode ser dividido em cinco partes $s = uvxyz$ satisfazendo as condições:

- paca cada $i \geq 0, uv^ixy^iz \in A$,
- $|vy| > 0$,
- $|vxy| \leq p$,

Capítulo 3

A tese de Church-Turing

3.1 Máquinas de Turing

Uma máquina de Turing pode fazer tudo o que um computador real pode fazer. O modelo da máquina de Turing usa uma fita infinita como sua memória ilimitada. Ela tem uma cabeça de fita que pode ler e escrever símbolos e mover-se sobre a fita. Inicialmente, a fita contem apenas a cadeia de entrada e está em branco em todo o restante. A máquina continua a computar até que ela decida produzir uma saída. As saídas aceite e rejeite são obtidas entrando em estados designados de *aceitação* e de *rejeição*. Se não entrar num estado de aceitacao ou de rejeição, ela continuará a computar para sempre.