

# Linguagens Formais e Autômatos

*Notas de Aula*

---

Valdigleis S. Costa

**Universidade Federal do Rio Grande do Norte – UFRN**

**Centro de Ciências Exatas e da Terra – CCET**

**Departamento de Informática e Matemática Aplicada – DIMAP**

9 de setembro de 2025

*Copyright* © 2019-2025 Linus van Pelt

Este texto NÃO possui qualquer tipo de vínculo editorial, e não possui fins lucrativos.

Página pessoal do autor <https://linus.pagina>

Este material é licenciado sob a Licença Atribuição-NãoComercial-CompartilhaIgual 3.0 Não Adaptada (CC BY-NC-SA 4.0). Você pode obter uma cópia da licença acessando a página:

<https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode.pt>

ou enviando uma carta para Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

---

Este tomo foi escrito com base em uma coleção de notas de aulas do autor, o mesmo foi redigido usando um *template* desenvolvido pelo próprio autor. Este texto foi escrito com o conjunto de macros L<sup>A</sup>T<sub>E</sub>X (em sua versão 2) e compilado usando as ferramentas LuaL<sup>A</sup>T<sub>E</sub>X e BibT<sub>E</sub>X, tais ferramentas fornecidas pelas distribuições T<sub>E</sub>XLive e MacT<sub>E</sub>X, respectivamente nos sistemas operacionais *Unix-like*: **Debian** e no **Mac OS X**, para edição foram usados os *softwares* livres de edição textual **Vim** (versão 0.10.1), além disso, o sistema de controle de versão adotado é o **Git** (versão 2.34.1).

*Release* compilado em 9 de setembro de 2025 (883 minutos após a meia-noite).

# Sumário

---

## I Fundamentos

<b>1</b>	<b>Sobre Autômatos e Linguagens</b>	<b>3</b>
1.1	Introdução	3
1.2	Sobre Autômatos Finitos	4
1.3	Noções Fundamentais	4
1.4	Sobre Gramática Formais	9
1.5	Questionário	11

## II Linguagens Regulares

<b>2</b>	<b>Autômatos Finitos e suas Linguagens</b>	<b>15</b>
2.1	Autômato Finito Determinístico	15
2.2	Autômatos Finitos Não-determinísticos	20
2.3	$\lambda$ -Autômatos Finitos Não-determinísticos	28
2.4	Teorema Myhill-Nerode e o Autômato Mínimo	34
2.5	Algoritmos de Minimização de AFD	37
2.6	Questionário	43
<b>3</b>	<b>Linguagens Regulares: Modelos Alternativos</b>	<b>47</b>
3.1	Gramática Regulares	47
	<b>Referências Bibliográficas</b>	<b>50</b>



# Parte I

## Fundamentos



# Sobre Autômatos e Linguagens

---

*“Ciência é uma equação diferencial.  
Religião é a condição de contorno.”*

*Alan M. Turing.*

## 1.1 Introdução

O conceito de linguagem formal é estabelecido como sendo um conjunto (possivelmente infinito) de palavras (ou sentenças) definidas sobre um dado alfabeto. Cada palavra em uma linguagem formal é simplesmente uma sequência finita de símbolos presente no alfabeto em questão. As palavras de uma linguagem formal, em alguns cenários também pode ser chamadas de termos, ou ainda, de fórmulas [22].

A sintaxe de qualquer linguagem formal é especificada por um conjunto finito de regras bem definidas, tal conjunto recebe o nome de gramática. A gramática é de fato o mecanismo que determinam a estrutura das palavras que podem existir dentro da linguagem, o que faz com que não exista nenhum grau de liberdade na forma (o por isso a nomenclatura **formal**) das palavras[7], ou seja, todas as palavras devem seguir uma forma rigorosa.

No que diz respeito a palavras em uma linguagem formal, elas (como já foi dito) são apenas sequência de símbolos sem qualquer significado. Para atribuir um significado as palavras de uma linguagem formal, isto é, para atribuir semântica a linguagem formal, é obrigatoriamente necessário definir externamente<sup>1</sup> uma estrutura avaliativa a semântica da linguagem, e de fato, isso é exatamente o que ocorrer durante a construção de compiladores e interpretadores para as linguagens de programação (detalhes em [2, 11]), essa estrutura externa funciona como um universo avaliativo, no qual as palavras da linguagem pode ser interpretadas (ter seu significado exposto).

<sup>1</sup> No sentido de que a interpretação das palavras é feita fora de sua gramática geradora.

Neste documento o foco será descrever uma teoria para computabilidade nos padrões apresentados por Turing [32], ou seja, uma visão de computabilidade por máquinas de computação, os chamados autômatos finitos [2, 17]. Além desse objetivo, essa parte do documento também irá formar uma pequena teoria para as linguagens formais usando modelos de representação, computação e geração, para as diferentes classes de linguagens. O que é importante para o leitor interessado em aprender sobre construção de compiladores e linguagens de programação, assim neste primeiro capítulo serão apresentados alguns conceitos básicos necessários nos capítulos seguintes.

## 1.2 Sobre Autômatos Finitos

Como dito em [12, 13] uma definição informal do conceito de autômato finito (ou máquina de estado finita) e que tais dispositivos podem ser vistos como sendo máquinas com dois componentes fundamentais:

- Um conjunto finito de memórias<sup>2</sup>, estas sendo subdivididas em células, cada uma das quais capaz de comportar um único símbolo por vez.
- Uma unidade de controle<sup>3</sup> que administra o estado atual do autômato e é responsável por executar as instruções (programa) da máquina.

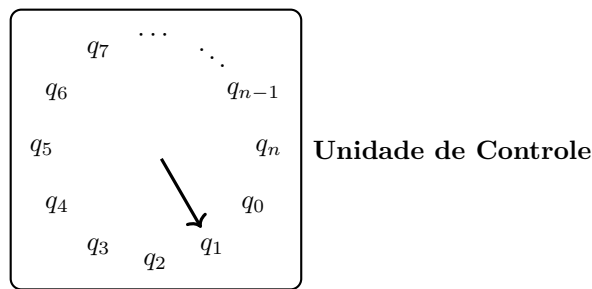


Figura 1.1: Representação informal do conceito de autômato finito com uma única memória retirado de [13].

Com respeito as memórias é comum assumir a existência de um **dispositivo de leitura e (ou) escrita**<sup>1</sup> que é capaz de acessar uma única célula por vez, e assim pode lê e (ou) escrever na célula. A depender do tipo de autômato podem existir vários dispositivos de leitura/escrita ou apenas um [8].

A(s) memória(s) de um autômato finito serve(m) para guarda dados (os símbolos) usados durante o funcionamento do autômato. O funcionamento de um autômato por sua vez, pode ser descrito em tempo discreto [12, 13], assim sendo, em qualquer momento no tempo  $t$ , a **unidade de controle** do autômato estará sempre em algum **estado** interno possível e a(s) **unidade(s) de leitura/escrita** tem acesso a alguma(s) **célula(s)** da(s) memória(s).

Formalmente pode-se dizer como apontado em [13], que a teoria dos autômatos finitos, ou simplesmente teoria dos autômatos, teve seu desenvolvimento inicial entre os anos de 1940 e 1960 sendo este início os trabalhos de McCulloch e Pitts [23], Kleene [18], Mealy [24], Moore [26], Rabin e Scott [29, 30].

## 1.3 Noções Fundamentais

Neste primeiro momento para o estudo dos autômatos finitos serão apresentados alguns conceitos fundamentais de extrema importância para o desenvolvimento das próximas seções e capítulos.

**Definição 1.1**

(Alfabetos e Palavras) [12] Qualquer conjunto finito e não vazio  $\Sigma$  será chamado de alfabeto. Qualquer sequência finita de símbolos na forma  $a_1 \cdots a_n$  com  $a_i \in \Sigma$  para todo  $1 \leq i \leq n$  será chamada de palavra sobre o alfabeto  $\Sigma$ .

<sup>1</sup>Também é usado a nomenclatura cabeçote [12, 13].



**Exemplo 1.3.1** Os conjuntos  $\{0, 1, 2, 3\}$ ,  $\{a, b, c\}$ ,  $\{\heartsuit, \spadesuit, \diamondsuit, \clubsuit\}$  e  $\{n \in \mathbb{N} \mid n \leq 25\}$  são todos alfabetos, os conjuntos  $\mathbb{N}$  e  $\mathbb{R}$  não são alfabetos.

**Exemplo 1.3.2** Dado o alfabeto  $\Sigma = \{0, 1, 2, 3\}$  tem-se que as sequências 0123, 102345, 1 e 0000 são todas palavras sobre  $\Sigma$ .

**Definição 1.2** (Comprimento das palavras) Seja  $w$  uma palavra qualquer sobre um certo alfabeto  $\Sigma$ , o comprimento<sup>a</sup> de  $w$ , denotado por  $|w|$ , corresponde ao número de símbolos existentes em  $w$ .

<sup>a</sup>Por conta desta notação em alguns texto é usado o termo módulo em vez de comprimento.

**Exemplo 1.3.3** Dado o alfabeto  $\Sigma = \{a, b, c, d\}$  e as palavras  $abcd, aacbd, c$  e  $ddaacc$  tem-se que:  $|abcd| = 4$ ,  $|aa| = 2$ ,  $|c| = 1$  e  $|ddaacc| = 6$ .

Como muito bem explicado em [8, 17, 19], pode-se definir uma série de operações sobre palavras, sendo a primeira delas a noção de concatenação.

**Definição 1.3** (Concatenação de palavras) Sejam  $w_1 = a_1 \cdots a_m$  e  $w_2 = b_1 \cdots b_n$  duas palavras quaisquer, tem-se que a concatenação de  $w_1$  e  $w_2$ , denotado por  $w_1 w_2$ , corresponde a uma sequência iniciada com os símbolos que forma  $w_1$  imediatamente seguido dos símbolos que forma  $w_2$ , ou seja,  $w_1 w_2 = a_1 \cdots a_m b_1 \cdots b_n$ .

É importante notar que a concatenação apenas combina duas palavras em uma nova palavra, sendo que, não existe qualquer tipo de exigência sobre os alfabeto sobre os quais as palavras usadas na concatenação estão definidas, ou seja, **podem ser alfabetos distintos**.

**Exemplo 1.3.4** Dado duas palavras  $w_1 = abra$  e  $w_2 = cadabra$  tem-se que  $w_1 w_2 = abracadabra$  e  $w_2 w_1 = cadabraabra$ .

Note que o Exemplo 1.3.4 estabelece que a operação de concatenação entre duas palavras não é comutativa, isto é, a ordem com que as palavras aparecem na concatenação é responsável pela forma da palavra resultante da concatenação.

**Teorema 1** (Associatividade da Concatenação) Para quaisquer  $w_1, w_2$  e  $w_3$  tem-se que  $(w_1 w_2) w_3 = w_1 (w_2 w_3)$ .

**Prova** Dado três palavras quaisquer  $w_1 = a_1 \cdots a_i, w_2 = b_1 \cdots b_j$  e  $w_3 = c_1 \cdots c_k$  tem-se que,

$$\begin{aligned} (w_1 w_2) w_3 &= (a_1 \cdots a_i b_1 \cdots b_j) c_1 \cdots c_k \\ &= a_1 \cdots a_i b_1 \cdots b_j c_1 \cdots c_k \\ &= a_1 \cdots a_i (b_1 \cdots b_j c_1 \cdots c_k) \\ &= w_1 (w_2 w_3) \end{aligned}$$

o que conclui a prova.  $\square$

Sobre qualquer alfabeto  $\Sigma$  sempre é definida uma palavra especial chamada **palavra vazia** [17, 19], essa palavra especial não possui nenhum símbolo, e em geral é usado o símbolo  $\lambda$  para denotar a palavra vazia [8, 12]. Como mencionado em [8, 13] sobre a palavra vazia é importante destacar que:

$$w\lambda = \lambda w = w \quad (1.1)$$

$$|\lambda| = 0 \quad (1.2)$$

Isto é, a palavra vazia é neutra para a operação de concatenação, além disso, a mesma apresenta comprimento nulo.

**Definição 1.4** (Potência das palavras) Seja  $w$  uma palavra sobre um alfabeto  $\Sigma$  a potência de  $w$  é definida recursivamente para todo  $n \in \mathbb{N}$  como sendo:

$$w^0 = \lambda \quad (1.3)$$

$$w^{n+1} = ww^n \quad (1.4)$$

**Exemplo 1.3.5** | Sejam  $w_1 = ab, w_2 = bac$  e  $w_3 = cbb$  palavras sobre  $\Sigma = \{a, b, c\}$  tem-se que:

(a)  $w_1^3 = w_1 w_1^2 = w_1 w_1 w_1^1 = w_1 w_1 w_1 w_1^0 = w_1 w_1 w_1 \lambda = ababab.$

(b)  $w_2^2 = w_2 w_2^1 = w_2 w_2 w_2^0 = w_2 w_2 \lambda = w_2 w_2 = bacbac.$

**Exemplo 1.3.6** | Seja  $u = 01$  e  $v = 231$  tem-se que:

$$uv^3 = uvv^2 = uvvv^1 = uvvv\lambda = uvvv = 01231231231$$

e também

$$u^2v = uu^1v = uu\lambda v = uuv = 0101231$$

**Lema 1** | Para toda palavra  $w$  e todo  $m, n \in \mathbb{N}$  tem-se que:

(i)  $(w^m)^n = w^{mn}.$

(ii)  $w^m w^n = w^{m+n}.$

**Prova** | Direto das Definições 1.3 e 1.4, e portanto, ficará como exercício ao leitor.  $\square$

Outro importante conceito existente sobre a ideia de palavra é a noção de palavra inversa (ou reversa) formalmente definida como se segue.

**Definição 1.5** (Palavra Inversa) [12] Seja  $w = a_1 \cdots a_n$  uma palavra qualquer, a palavra inversa de  $w$  denotada por  $w^r$ , é tal que  $w^r = a_n \cdots a_1$ .

**Exemplo 1.3.7** | Dado as palavras  $u = aba, v = 011101$  e  $w = 3021$  tem-se que  $u^r = aba, v^r = 101110$  e  $w^r = 1203$ .

Além das palavras, pode-se também formalizar uma série de operações sobre a própria noção de alfabeto. Em primeiro lugar, uma vez que, alfabetos são conjuntos, obviamente todas operações usuais de união, interseção, complemento, diferença e diferença simétrica também são válidas sobre alfabetos. Além dessas operações, também esta definida a operação de potência e os fechos positivo e de Kleene sobre alfabetos.

**Definição 1.6** (Potência de um alfabeto) [8] Seja  $\Sigma$  um alfabeto a potência de  $\Sigma$  é definida recursivamente para todo  $n \in \mathbb{N}$  como:

$$\Sigma^0 = \{\lambda\} \quad (1.5)$$

$$\Sigma^{n+1} = \{aw \mid a \in \Sigma, w \in \Sigma^n\} \quad (1.6)$$

**Exemplo 1.3.8** | Dado  $\Sigma = \{a, b\}$  tem-se que  $\Sigma^3 = \{aaa, aab, aba, baa, abb, bab, bba, bbb\}$  e  $\Sigma^1 = \{a, b\}$

**Exemplo 1.3.9** | Seja  $\Sigma = \{0, 1, 2\}$  tem-se que  $\Sigma^2 = \{00, 01, 02, 10, 11, 12, 20, 21, 22\}$  e  $\Sigma^0 = \{\lambda\}.$

O leitor mais atencioso e maduro matematicamente pode notar que para qualquer que seja  $n \in \mathbb{N}$  o conjunto potência tem a propriedade de que todo  $w \in \Sigma^n$  é tal que  $|w| = n$ , além disso, é claro que todo  $\Sigma^n$  é sempre finito, uma vez que, o conjunto  $\Sigma^n$  pode ser visto como sendo nada mais do que, um conjunto de arranjos com repetição.

**Definição 1.7** (Fecho Positivo e de Kleene) Seja  $\Sigma$  um alfabeto o fecho positivo e o fecho de Kleene de  $\Sigma$ , denotados respectivamente por  $\Sigma^+$  e  $\Sigma^*$ , correspondem aos conjuntos:

$$\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i \text{ e } \Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i.$$

Obviamente como dito em [8], o fecho positivo pode ser reescrito em função do fecho de Kleene usando a operação de diferença de conjunto, isto é, o fecho positivo corresponde a seguinte identidade,  $\Sigma^+ = \Sigma^* - \{\lambda\}$ . Sobre o fecho de Kleene como destacado em [13] o mesmo corresponde ao monoide livremente gerado pelo conjunto  $\Sigma$  munida da operação de concatenação.

**Definição 1.8** (Prefixos e Sufixos) Uma palavra  $u \in \Sigma^*$  é um prefixo de outra palavra  $w \in \Sigma^*$ , denotado por  $u \preceq_p w$ , sempre que  $w = uv$ , com  $v \in \Sigma^*$ . Por outro lado, uma palavra  $u$  é um sufixo de outra palavra  $w$ , denotado por  $u \preceq_s w$ , sempre que  $w = vu$ .

**Exemplo 1.3.10** Seja  $w = abracadabra$  tem-se que as palavras  $ab$  e  $abrac$  são prefixos de  $w$ , por outro lado  $cadabra$  e  $bra$  são sufixos de  $w$ , e a palavra  $abra$  é prefixo e também sufixo. Já a palavra  $cada$  não é prefixo e nem sufixo de  $w$ .

**Definição 1.9** (Conjunto dos Prefixos e Sufixos) Seja  $w \in \Sigma^*$  o conjunto de todos os prefixos de  $w$  corresponde ao conjunto:

$$PRE(w) = \{w' \in \Sigma^* \mid w' \preceq_p w\} \quad (1.7)$$

e o conjunto de todos os sufixos de  $w$  corresponde ao conjunto:

$$SUF(w) = \{w' \in \Sigma^* \mid w' \preceq_s w\} \quad (1.8)$$

**Exemplo 1.3.11** Seja  $w = univasf$  tem-se que:

$$PRE(w) = \{\lambda, u, un, uni, univ, univa, univas, univasf\}$$

e

$$SUF(w) = \{\lambda, f, sf, asf, vasf, ivasf, nivasf, univasf\}$$

**Exemplo 1.3.12** A seguir é apresentado alguns exemplos de palavras e seus conjuntos de prefixos e sufixos.

- (a) Se  $w = ab$ , então  $PRE(w) = \{\lambda, a, ab\}$  e  $SUF(w) = \{\lambda, b, ab\}$ .
- (b) Se  $w = 001$ , então  $PRE(w) = \{\lambda, 0, 00, 001\}$  e  $SUF(w) = \{\lambda, 1, 01, 001\}$ .
- (c) Se  $w = \lambda$ , então  $PRE(w) = \{\lambda\}$  e  $SUF(w) = \{\lambda\}$
- (d) Se  $w = a$ , então  $PRE(w) = \{\lambda, a\}$  e  $SUF(w) = \{\lambda, a\}$ .

Com respeito a cardinalidade dos conjuntos de prefixos e sufixos, os mesmo apresentam as propriedades descritas pelo teorema a seguir.

**Teorema 2** Para qualquer que seja  $w \in \Sigma^*$  as seguintes asserções são verdadeiras.

- (i)  $\#PRE(w) = |w| + 1$ .
- (ii)  $\#PRE(w) = \#SUF(w)$ .
- (iii)  $\#(PRE(w) \cap SUF(w)) > 1$ .

**Prova** | Dado uma palavra  $w$  tem-se que:

- (i) Sem perda de generalidade assumindo que  $w = a_1 \cdots a_n$  logo  $w \in \Sigma^n$  (o caso quando  $w = \lambda$  é trivial e não será demonstrado aqui) logo  $|w| = n$  para algum  $n \in \mathbb{N}$ , assim existem exatamente  $n$  palavras da forma  $a_1 \cdots a_i$  com  $1 \leq i \leq n$  tal que  $a_1 \cdots a_i \preceq_p w$ , portanto, para todo  $1 \leq i \leq n$  tem-se que  $a_1 \cdots a_i \in PRE(w)$ , além disso, é claro que  $w = \lambda w$ , e portanto,  $\lambda \in PRE(w)$ , consequentemente,  $\#PRE(w) = n + 1 = |w| + 1$ .
- (ii) Análoga ao item anterior.
- (iii) Trivial, pois basta notar que  $\lambda, w \in (PRE(w) \cap SUF(w))$ , e portanto, tem-se claramente que  $\#(PRE(w) \cap SUF(w)) > 1$ .  $\square$

**Corolário 1** | Toda palavra tem pelo menos um prefixo e um sufixo.

**Prova** | Direto do item (iii) do Teorema 2.  $\square$

Seguindo com este documento pode-se finalmente formalizar o pilar fundamental (a ideia de linguagem) necessário para desenvolver o estudo da computabilidade neste e nos próximos capítulos.

**Definição 1.10** | (Linguagem) Dado um alfabeto  $\Sigma$ , qualquer subconjunto  $L \subseteq \Sigma^*$  será chamado de linguagem.

**Exemplo 1.3.13** | Seja  $\Sigma = \{0, 1\}$  tem-se que os conjuntos a seguir são todos linguagens sobre  $\Sigma$ .

- (a)  $\Sigma^*$ .
- (b)  $\{0^n b^n \mid n \in \mathbb{N}\}$ .
- (c)  $\{\lambda, 0, 1\}$ .
- (d)  $\Sigma^{22}$ .
- (e)  $\emptyset$ .

Similarmente ao que ocorre com os alfabetos, as linguagens por serem conjuntos “herdam” as operações básicas da teoria dos conjuntos [20, 21, 1], isto é, estão definidas sobre as linguagens as operações de união, interseção, complemento, diferença e diferença simétrica. E como par aos alfabetos novas operações são definidas.

**Definição 1.11** | (Concatenação de Linguagens) Sejam  $L_1$  e  $L_2$  duas linguagens, a concatenação de  $L_1$  com  $L_2$ , denotado por  $L_1 L_2$ , corresponde a seguinte linguagem:

$$L_1 L_2 = \{xy \in (\Sigma_1 \cup \Sigma_2)^* \mid x \in L_1, y \in L_2\} \quad (1.9)$$

**Exemplo 1.3.14** | Dado as três linguagens  $L_1 = \{\lambda, ab, bba\}$ ,  $L_2 = \{0^{2n}1 \mid n \in \mathbb{N}\}$  e  $L_3 = \{a^p \mid p \text{ é um número primo}\}$  tem-se que:

- (a)  $L_1 L_2 = \{w \mid w = 0^{2n}1 \text{ ou } w = ab0^{2n}1 \text{ ou } w = bba0^{2n}1 \text{ com } n \in \mathbb{N}\}$ .
- (b)  $L_3 L_1 = \{w \mid w = a^p \text{ ou } w = a^{p+1}b \text{ ou } a^p bba \text{ onde } p \text{ é um número primo}\}$ .
- (c)  $L_2 L_3 = \{0^{2n}1a^p \mid n \in \mathbb{N}, p \text{ é um número primo}\}$ .

**Definição 1.12** | (Linguagem Reversa) Seja  $L$  uma linguagem, a linguagem inversa de  $L$ , denotada por  $L^r$ , corresponde ao conjunto  $\{w^r \mid w \in L\}$ .

**Exemplo 1.3.15** | Considerando as linguagens  $L_1, L_2$  e  $L_3$  do Exemplo 1.3.14 tem-se que:

- (a)  $L_1^r = \{\lambda, ba, abb\}$ .

- (b)  $L_2^r = \{10^{2n} \mid n \in \mathbb{N}\}$ .
- (c)  $L_3^r = \{a^p \mid n \in \mathbb{N}, p \text{ é um número primo}\}$ .

O leitor mais atento pode perceber que a propriedade involutiva da operação reversa sobre palavras é “herdada” para a reversão sobre linguagens, isto é, para qualquer linguagem  $L$  tem-se que  $(L^r)^r = L$ .

**Definição 1.13** (Linguagem Potência) Seja  $L$  uma linguagem, a linguagem potência de  $L$ , denotada por  $L^n$ , é definida recursivamente para todo  $n \in \mathbb{N}$  como:

$$L^0 = \{\lambda\} \quad (1.10)$$

$$L^{n+1} = LL^n \quad (1.11)$$

Utilizando o conceito de linguagem potência a seguir é apresentado a formalização para os fechos positivo e de Kleene sobre linguagens.

**Definição 1.14** (Fecho positivo e Fecho de Kleene de Linguagens) Seja  $L$  uma linguagem, o fecho positivo  $(L^+)$  e o fecho de Kleene  $(L^*)$  de  $L$  são dados pelas equações a seguir.

$$L^+ = \bigcup_{i=1}^{\infty} L^i \quad (1.12)$$

$$L^* = \bigcup_{i=0}^{\infty} L^i \quad (1.13)$$

Por fim, esta seção irá apresentar a noção de linguagem dos prefixos e sufixos.

**Definição 1.15** (Linguagem de Prefixos e Sufixos) Seja  $L$  uma linguagem, a linguagem dos prefixos e dos sufixos de  $L$ , respectivamente  $PRE(L)$  e  $SUF(L)$ , são exatamente os seguintes conjuntos:

$$PRE(L) = \{w' \in \Sigma^* \mid w' \preceq_p w, w \in L\}$$

$$SUF(L) = \{w' \in \Sigma^* \mid w' \preceq_s w, w \in L\}$$

Nos próximos capítulos deste documento irão ser apresentadas as formalizações da ideia de linguagens formais na visão “mecânica” de Turing [32]. Entretanto, em vez de apresentar de forma direta os conceitos ligados as máquinas de Turing e as computações por elas realizadas, este documento opta por fazer um estudo seguindo a ideia dos livros texto de linguagens formais [8, 19, 25], assim sendo, aqui serão apresentadas as linguagens formais da mais simples para a mais complexas seguindo a hierarquia de Chomsky [10], ou seja, serão aqui estudadas as linguagens formais na seguintes ordem: regulares, livres do contexto, recursivas e recursivamente enumeráveis.

## 1.4 Sobre Gramática Formais

Agora que foram introduzidos os conceitos fundamentais para a teoria das linguagens formais pode-se formalizar o conceito de estrutura geradora ou gramática formal, o leitor mais atento e com maior conhecimento sobre lógica de primeira ordem e teoria da prova [3, 9] pode notar que gramáticas formais são na verdade outro nome para sistemas de reescrita [4].

**Definição 1.16** (Gramática formal) Uma gramática formal é uma estrutura da forma  $G = \langle V, \Sigma, S, P \rangle$  onde  $V$  é um conjunto não vazio de símbolos chamados variáveis tal que  $V \cap \Sigma = \emptyset$ ,  $\Sigma$  é um alfabeto,  $S \in V$  é uma variável destacada chamada de **variável inicial** e  $P$  é um conjunto de regras de reescrita<sup>a</sup> da forma  $w \rightarrow w'$  onde

$w \in (V \cup \Sigma)^+$  e  $w' \in (V \cup \Sigma)^*$ .

<sup>a</sup>Também é comum encontrar na literatura a nomenclatura regras de produção [8, 19].



Atenção

Na escrita do conjunto  $P$  sempre que  $w \rightarrow w_1$  e  $w \rightarrow w_2$  com  $w_1 \neq w_2$ , é escrito simplesmente  $w \rightarrow w_1 \mid w_2$ , em vez de escrever as duas regras separadas.

Exemplo 1.4.1

A estrutura  $G = \langle \{A, B\}, \{a\}, A, P \rangle$  em que  $P$  é formado pelas regras  $A \rightarrow aABa \mid B$  e  $B \rightarrow \lambda$  é uma gramática formal.

Qualquer gramática então pode ser visto com um sistema para a geração de palavras através de um mecanismo chamado derivação descrito a seguir.

Definição 1.17

(Derivação de palavras) Dado uma gramática  $G = \langle V, \Sigma, S, P \rangle$ , a palavra  $XwY$  deriva a palavra  $Xw'Y$  na gramática  $G$ , denotado por  $XwY \vdash_G Xw'Y$ , sempre que existe uma regra forma  $w \rightarrow w' \in P$ .

Exemplo 1.4.2

Dado a gramática do Exemplo 1.4.1 tem-se que  $aABa \vdash_G aaABaBa$ , pois existe em  $P$  a regra  $A \rightarrow aABa$ .

Rigorosamente  $\vdash_G$  na verdade é uma relação entre  $(V \cup \Sigma)^+$  e  $(V \cup \Sigma)^*$ , e assim  $\vdash_G^*$  denota o fecho transitivo e reflexivo de  $\vdash_G$ , além disso, sempre que não causar confusão é comum eliminar a escrita do rótulo da gramática, ou seja, são escritos respectivamente  $\vdash^*$  e  $\vdash$  em vez de  $\vdash_G^*$  e  $\vdash_G$ .

Exemplo 1.4.3

Considerando ainda a gramática exibida no Exemplo 1.4.1 tem-se que  $aABa \vdash^* aaaABaBaBa$ , uma vez que,  $aABa \vdash aaABaBa \vdash aaaABaBaBa$ .

Exemplo 1.4.4

A estrutura  $G = \langle \{A, B, S\}, \{0, 1\}, S, P \rangle$  em que  $P$  é formado pelas regras  $S \rightarrow 11A$ ,  $A \rightarrow B0$  e  $B \rightarrow 000$  é uma gramática formal e assim  $11A \vdash^* 110000$ , pois tem-se que,  $11A \vdash^* 11B0 \vdash^* 110000$ .

Como dito em [8], dado uma gramática formal  $G$  sempre que houver uma sequência de derivações  $w_1 \vdash w_2 \vdash \dots \vdash w_n$  acontecer, as palavras  $w_1, w_2, \dots, w_n$  são chamadas de formas sentenciais, ou simplesmente, sentenças. Assim uma derivação nada mais é do que uma sequência finita de formas sentenciais.

Definição 1.18

(Igualdade de Derivações) Dado uma gramática  $G = \langle V, \Sigma, S, P \rangle$  e duas derivações  $S \vdash w_1 \vdash^* w_n$  e  $S \vdash w'_1 \vdash^* w'_n$  sobre  $G$ , será dito que estas derivações são iguais sempre que  $w_i = w'_i$  para todo  $1 \leq i \leq n$ .

Desde que  $\vdash^*$  é de fato uma relação pode-se facilmente que a igualdade entre derivações nada mais é do que a igualdade entre tuplas ordenadas.

Definição 1.19

(Linguagem de uma gramática) Dado uma gramática  $G = \langle V, \Sigma, S, P \rangle$  a linguagem gerada por  $G$ , denotada por  $\mathcal{L}(G)$ , corresponde ao conjunto formado por todas as palavras sobre  $\Sigma$  que são deriváveis a partir do variável inicial da gramática, ou seja,  $\mathcal{L}(G) = \{w \in \Sigma^* \mid S \vdash^* w\}$ .

Exemplo 1.4.5

Não é difícil verificar que a gramática do Exemplo 1.4.1 gera a linguagem  $\{w \in \{a\}^* \mid |w| = 2k, k \in \mathbb{N}\}$ .

Agora o leitor pode ter notado que como gramática formais possuem um conjunto finito de regras, as linguagens por elas geradas nada mais são do que conjuntos indutivamente gerados.

## 1.5 Questionário

**Questão 1.1** Dado o alfabeto  $\Sigma = \{a, b, c\}$  e as palavras  $u = aabcab, v = bbccabac$  e  $w = ccbabbaaca$  determine:

- (a). A palavra  $uv^r$ .
- (b). A palavra  $(w^r)^2u$ .
- (c). A palavra  $((u^r)^2v^0)^rv$ .
- (d). A palavra  $uu^2v^rw$ .
- (e). A palavra  $((wuv)^r)^2u$ .
- (f). Determine o valor numérico da expressão  $|w^3| + 2|v^2u| - |u|$ .
- (g). Determine o valor numérico da expressão  $2|w^r| - |uv|$ .
- (h). Determine o valor numérico da expressão  $|w^raaw| - |w|$ .
- (i). Determine o valor numérico da expressão  $|uv^r| - 4$ .
- (j). Determine o valor numérico da expressão  $\frac{|(w^r)^2u|}{2} - \frac{|u|}{6}$ .

**Questão 1.2** Demonstre que, se  $u$  é um prefixo de  $v$ , então  $|u| \leq |v|$ .

**Questão 1.3** Demonstre para quaisquer palavras  $u$  e  $v$  e para todo  $n \in \mathbb{N}$  as asserções a seguir.

- (a).  $|u^n| = n|u|$ .
- (b).  $|(uv)^r| = |vu|$ .
- (c). Se  $|u| = n$ , então  $n \leq |uv|$ .

**Questão 1.4** Considere a linguagem  $L = \{\lambda, abb, a, abba\}$  e determine:

- (a).  $L^r - \{\lambda, a\}$ .
- (b).  $L^3$ .
- (c).  $PRE(L)$ .
- (d).  $SUF(L^2)$ .
- (e).  $w$  tal que  $|w| = \max(\{|w'| \mid w' \in L^3\})$ .

**Questão 1.5** Prove que para qualquer linguagem  $L$  e quaisquer  $m, n \in \mathbb{N}$  as seguintes asserções.

- (a).  $(L^m)^n = L^{mn}$ .
- (b).  $L^m L^n = L^{m+n}$ .
- (c).  $(L^r)^n = (L^n)^r$ .
- (d).  $\overline{L^r} = \overline{L^r}$ .
- (e).  $PRE(L) = (SUF(L^r))^r$ .

**Questão 1.6** Dado duas linguagens quaisquer  $L_1$  e  $L_2$  demonstre ou apresente um contra-exemplo para os seguintes enunciados:

- (a). Se  $L_1 \cap L_2 \neq \emptyset$ , então  $PRE(L_1) \cap PRE(L_2) = \emptyset$ .
- (b). Se  $L_1 \subseteq L_2$ , então  $SUF(L_1) \cap SUF(L_2) = \emptyset$ .
- (c). Se  $L_1 \subseteq L_2$ , então  $L_1^r \subseteq L_2^r$ .
- (d). Se  $L_1 \subseteq L_2$ , então para todo  $L$  tem-se que  $LL_1 \subseteq LL_2$ .

**Questão 1.7** Demonstre ou refute o predicado  $(\forall L \subseteq \Sigma^*)(\forall n \in \mathbb{N})[\overline{L}^n = \overline{L^n}]$ .

**Questão 1.8** Esboce uma linguagem  $L$  não trivial<sup>a</sup>, para que a igualdade:

$$PRE(L) = (SUF(L))^r$$

seja verdadeira.

---

<sup>a</sup>Trivial aqui diz respeito a uma linguagem que não seja o próprio alfabeto ou o conjunto  $\{\lambda\}$ .



**Parte II**

**Linguagens Regulares**



# Autômatos Finitos e suas Linguagens

*“Eu acredito que às vezes são as pessoas que ninguém espera nada que fazem as coisas que ninguém consegue imaginar.”*

*Alan M. Turing.*

Como explicado em diversas obras tais como [8, 17, 19, 25], os autômatos finitos podem ser separados em dois tipos bem definidos, a saber, Autômato Finito Determinístico (AFD) e Autômato Finito Não-determinístico (AFN).

## 2.1 Autômato Finito Determinístico

Agora este documento inicia o estudo dos AFD apresentando sua forma algébrica equacional.

### Definição 2.1

(Autômato Finito Determinístico) Um AFD é uma estrutura  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  onde:  $Q$  é um conjunto finito de estados,  $\Sigma$  é um alfabeto,  $\delta : Q \times \Sigma \rightarrow Q$  é uma função total (chamada função de transição),  $q_0 \in Q$  é um estado destacado (chamado estado inicial) e  $F \subseteq Q$  é o conjunto de estados finais<sup>a</sup>.

<sup>a</sup>Em algumas referências também é usado o termo conjunto de estados de aceitação [14].

### Exemplo 2.1.1

A estrutura  $A = \langle \{q_0, q_1\}, \{a\}, \delta, q_0, \{q_1\} \rangle$  onde a função de transição é definida por:  $\delta(q_0, a) = q_1$  e  $\delta(q_1, a) = q_0$ , é um AFD.

### Exemplo 2.1.2

A estrutura  $B = \langle \{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_0\} \rangle$  onde a função de transição é definida por:  
não é um AFD, pois  $\delta(q_2, b)$  não está definido, e portanto,  $\delta$  não é uma função total furando assim a definição de AFD.

Não é um fato claro por que usamos a letra  $Q$  para representar o conjunto de estados, e a letra minúscula  $q$  para simbolizar a um estado genérico, mas isso agora é o padrão notacional firmemente estabelecido nas principais obras da área [8, 17, 19].

Embora o estudo formal dos autômatos finitos tenha começado muito antes, sua formulação moderna foi estabelecida pelo artigo<sup>1</sup> do ano de 1959 escrito por Michael Rabin e Dana Scott [30]. Em tal artigo Rabin e Scott chamaram o conjunto de estados

<sup>1</sup>Esse artigo levou Michael Rabin e Dana Scott a ganharem o Prêmio Turing.

de  $S$ , usaram a letra minúscula  $s$  para um estado genérico e chamaram o estado inicial de  $s_0$ .

Por outro lado, no artigo de 1936, que é o trabalho seminal da teoria da computação, Turing usou  $q_1, q_2, \dots, q_R$  para se referir aos estados (ou “m-configurações”) de uma máquina de Turing genérica, não existe uma evidência forte que explique o motivo da escolha do  $q$ . Então o leitor não deve se preocupar se hora esse texto usar  $q$  e depois  $s$  para representar os estados, pois isso é apenas um conceito de notação e estética textual.



Atenção

As siglas AFD e AFN serão usadas tanto para designar o singular quanto o plural, ficando a distinção a critério das sentenças envolvendo tais singlas.

A função de transição ( $\delta$ ) pode ser interpretada semanticamente como sendo o programa que o autômato executa, assim uma aplicação qualquer de  $\delta$  é uma instrução do programa do autômato, por exemplo, a aplicação  $\delta(q, x) = p$  significa que, o AFD muda do estado atual  $q$  para o estado  $p$  quando o mecanismo de leitura lê o símbolo  $x$  na memória.

Uma representação comum para os AFD é baseada no uso de grafos de transição [13]. Em um grafo de transição os vértices irão ser representados por círculos, que neste caso são usados para representar os estados do autômato, isto é, os círculos representam os elementos de  $Q$ . Cada aresta  $(q_i, q_j)$  são rotuladas por  $x$  representando assim a transição da forma  $\delta(q_i, x) = q_j$ . Por fim, os estados finais, isto é, cada  $q \in F$  será representado por vértices desenhados como um círculo duplo em vez de um círculo simples e o estado inicial é marcado com uma seta.

**Exemplo 2.1.3** A representação por grafo de transição do AFD descrito no Exemplo 2.1.1 corresponde a figura a seguir.

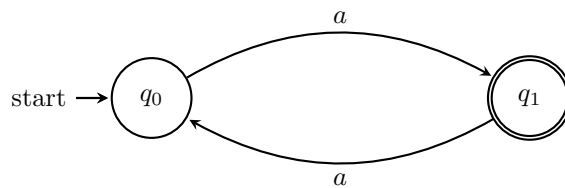


Figura 2.1: Representação visual do AFD no Exemplo 2.1.1.

**Exemplo 2.1.4** O AFD  $S = \langle \{s_0, s_1, s_2\}, \{0, 1\}, \delta, s_0, \emptyset \rangle$  onde a função de transição é definida como sendo:  $\delta(s_0, 0) = s_1, \delta(s_1, 0) = s_2, \delta(s_2, 0) = s_1, \delta(s_0, 1) = s_2, \delta(s_1, 1) = s_1$  e  $\delta(s_2, 1) = s_1$ , é um AFD e pode ser representado pela Figura 2.2 a seguir.

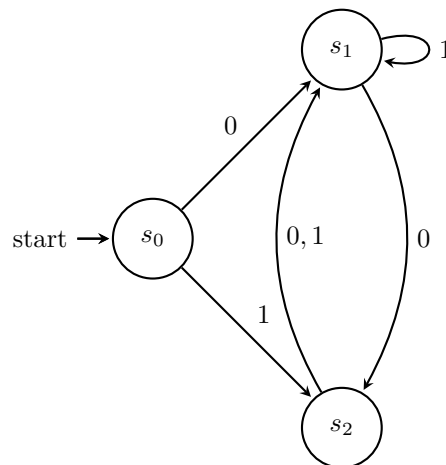


Figura 2.2: Representação visual do AFD  $S$  do Exemplo 2.1.4.

Pode-se agora então estender a função de transição, para que o autômato possa vir a processar palavras, em vez de apenas símbolos individuais.

**Definição 2.2** (Função de Transição Estendida) Seja  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  um AFD a função  $\delta$  é estendida para uma função  $\widehat{\delta} : Q \times \Sigma^* \rightarrow Q$  usando recursividade como se segue.

$$\widehat{\delta}(q, \lambda) = q \quad (2.1)$$

$$\widehat{\delta}(q, wa) = \delta(\widehat{\delta}(q, w), a) \quad (2.2)$$

onde  $q \in Q, a \in \Sigma$  e  $w \in \Sigma^*$ .

A partir da definição de função de transição estendida é definida a noção de computação para os AFD, tal conceito é formalizado a seguir.

**Definição 2.3** (Computação em AFD) Seja  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  um AFD e seja  $w \in \Sigma^*$  uma computação de  $w$  em  $A$  corresponde a aplicação  $\widehat{\delta}(q_0, w)$ .

Note que a definição de computação em AFD pode ser interpretada como sendo a resposta ao seguinte questionamento: “Em que estado o autômato (ou a máquina) estará após iniciar o processamento no estado inicial e ter lido todos os símbolos da palavra de entrada  $w$ ?”

**Exemplo 2.1.5** Considere o AFD do Exemplo 2.1.1 e a palavra de entrada  $aaaa$  tem-se que a computação desta palavra corresponde a:

$$\begin{aligned} \widehat{\delta}(q_0, aaaa) &= \delta(\widehat{\delta}(q_0, aaa), a) \\ &= \delta(\delta(\widehat{\delta}(q_0, aa), a), a) \\ &= \delta(\delta(\delta(\widehat{\delta}(q_0, a), a), a), a) \\ &= \delta(\delta(\delta(\delta(\widehat{\delta}(q_0, \lambda), a), a), a), a) \\ &= \delta(\delta(\delta(\delta(q_0, a), a), a), a) \\ &= \delta(\delta(\delta(q_1, a), a), a) \\ &= \delta(\delta(q_0, a), a) \\ &= \delta(q_1, a) \\ &= q_0 \end{aligned}$$

**Exemplo 2.1.6** Considere o AFD do Exemplo 2.1.4 e a palavra de entrada 0101 tem-se que a computação desta palavra corresponde a:

$$\begin{aligned} \widehat{\delta}(s_0, 0101) &= \delta(\widehat{\delta}(s_0, 010), 1) \\ &= \delta(\delta(\widehat{\delta}(s_0, 01), 0), 1) \\ &= \delta(\delta(\delta(\widehat{\delta}(s_0, 0), 1), 0), 1) \\ &= \delta(\delta(\delta(\delta(\widehat{\delta}(s_0, \lambda), 0), 1), 0), 1) \\ &= \delta(\delta(\delta(\delta(s_0, 0), 1), 0), 1) \\ &= \delta(\delta(\delta(s_1, 1), 0), 1) \\ &= \delta(\delta(s_1, 0), 1) \\ &= \delta(s_2, 1) \\ &= s_1 \end{aligned}$$

De pose da definição de computação pode-se formalizar o conceito de reconhecimento (ou aceitação) de palavras nos AFD.

**Definição 2.4** (Reconhecimento de palavras em AFD) [8] Sejam  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  um AFD e seja  $w \in \Sigma^*$ . A palavra  $w$  é dita aceita (reconhece ou computada) por  $A$  sempre que  $\hat{\delta}(q_0, w) \in F$  e é rejeitada por  $A$  em qualquer outro caso.

É fácil perceber que  $\hat{\delta}(q_0, w) \in F$  com  $w = a_1 a_2 \cdots a_n$  se, e somente se, existir uma sequência finita de estados  $(q_i)_{i \in I}$  tal que,

$$\delta(q_0, a_1) = q_{i_1}, \delta(q_{i_1}, a_2) = q_{i_2}, \dots, \delta(q_{i_{n-1}}, a_n) = q_{i_n}$$

com  $q_n \in F$ , sendo uma sequência de números naturais e  $i_1, i_2, i_{n-1}, i_n \in I$ . O leitor pode notar que em particular tem-se que  $\hat{\delta}(q_0, \lambda) \in F$  se, e somente se,  $q_0 \in F$ .

**Exemplo 2.1.7** Considerando os Exemplos 2.1.5 e 2.1.6 tem-se que a palavra  $aaaa$  não é aceita pelo AFD do Exemplo 2.1.5, uma vez que,  $q_0 \notin F$ . Já a palavra  $0101$  também não é aceita pelo AFD do Exemplo 2.1.6, uma vez que,  $s_1 \notin F$ , de fato o leitor atento pode notar que o AFD do Exemplo 2.1.6 não aceita qualquer palavra de entrada pois  $F = \emptyset$ .

**Exemplo 2.1.8** Considere o AFD representado pelo grafo de transições abaixo:

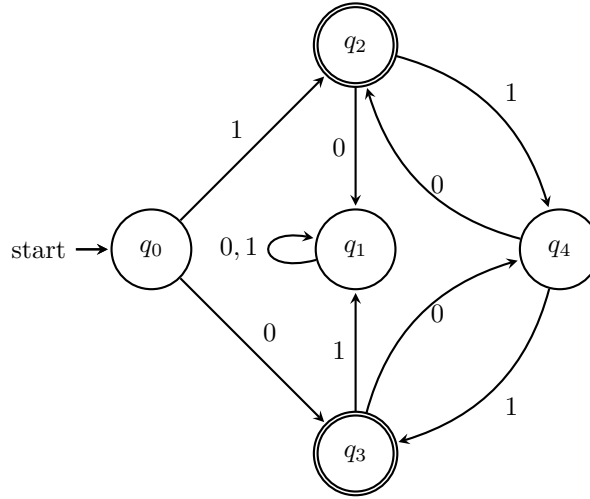


Figura 2.3: Um AFD com dois estados finais.

Por indução sobre o tamanho das palavras é fácil mostrar que este AFD reconhece palavras das forma  $1(10)^n$  e  $1(10)^n$  com  $n \in \mathbb{N}$ .

Tendo definido precisamente as noções de AFD e de computação em AFD, agora é possível definir formalmente a ideia de linguagem reconhecida (ou computada) por um AFD.

**Definição 2.5** (Linguagem de um AFD) Seja  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  um AFD a linguagem reconhecida (ou computada) por  $A$ , denotada por  $\mathcal{L}(A)$ , corresponde ao conjunto de todas as palavras aceitas por  $A$ , formalmente tem-se que:

$$\mathcal{L}(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\} \quad (2.3)$$

Utilizando a definição acima o leitor deve ser capaz de perceber que se um AFD reconhece uma linguagem  $L \subseteq \Sigma^*$ , então ele para em estados finais apenas para as palavras  $w \in L$ . Em outra palavra para mostrar que uma linguagem  $L$  é a linguagem de um AFD  $A$ , deve-se provar que  $L = \mathcal{L}(A)$ , ou seja, deve-se provar que  $w \in L \iff w \in \mathcal{L}(A)$ , em geral quando  $L$  é infinito tal prova é por indução.

Exemplo 2.1.9

A seguir você encontrará a prova de que a linguagem  $L = \{bba^{2n} \mid n \in \mathbb{N}\}$  é reconhecida pelo AFD  $A_1$  na Figura 2.4 a seguir.

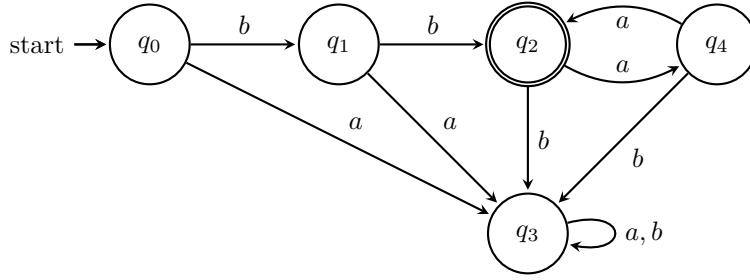


Figura 2.4: AFD  $A_1$  que reconhece a linguagem  $\{bba^{2n} \mid n \in \mathbb{N}\}$ .

Prova

( $\Rightarrow$ ) Suponha que  $w \in L$  assim  $w = bba^{2n}$  e por indução sobre o tamanho das palavras tem-se que,

**(B)**ase: Quando  $n = 0$  vale que  $w = bba^{2 \cdot 0}$  e usando a definição do AFD tem-se que,

$$\hat{\delta}(q_0, bba^{2 \cdot 0}) = \hat{\delta}(q_0, bb) = \delta(\hat{\delta}(q_0, b), b) = \delta(\delta(\hat{\delta}(q_0, \lambda), b), b) = q_2$$

como  $q_2 \in F$  tem-se que  $bb \in \mathcal{L}(A_1)$ .

**(H)**ipótese indutiva: Suponha que para todo  $n \in \mathbb{N}$  tem-se que  $\hat{\delta}(q_0, bba^{2n}) \in F$ , ou seja,  $\hat{\delta}(q_0, bba^{2n}) = q_2$ .

**(P)**asso indutivo: Dado  $w = bba^{2(n+1)}$  tem-se que

$$\begin{aligned}
 \hat{\delta}(q_0, bba^{2(n+1)}) &= \hat{\delta}(q_0, bba^{2n+2}) \\
 &= \hat{\delta}(q_0, bba^{2n}aa) \\
 &= \delta(\hat{\delta}(q_0, bba^{2n}), a), a) \\
 &\stackrel{\text{(HI)}}{=} \delta(\delta(q_2, a), a) \\
 &= \delta(q_3, a) \\
 &= q_2
 \end{aligned}$$

Logo, por **(B)**, **(H)** e **(P)** tem-se que  $\hat{\delta}(q_0, bba^{2n}) \in \mathcal{L}(A_1)$  para qualquer que seja  $n \in \mathbb{N}$ .

( $\Leftarrow$ ) Suponha que  $w \in \mathcal{L}(A_1)$ , assim pela definição do AFD  $A_1$  tem-se que  $\hat{\delta}(q_0, w) = q_2$ , entretanto, pela definição de  $\delta$  (ver Figura 2.4) tem-se que  $q_2$  só é acessado pelas transições  $\delta(q_1, b)$  e  $\delta(q_4, a)$ , ou seja,  $w = w_1a$  ou  $w = w_2b$  com  $w_1, w_2 \in \Sigma^*$ . Agora analisando cada possibilidade em separado tem-se que:

- Para realizar o acesso via  $q_1$  é necessário obviamente chegar em  $q_1$  e isso só é possível a partir da transição  $\delta(q_0, b)$ , logo o acesso a  $q_2$  via  $q_1$  só é permitido para palavras com o prefixo  $bb$ , agora como toda palavra é prefixo de si mesmo isso já garante que  $bb \in \mathcal{L}(A_1)$ .
- Já o acesso via  $q_4$  só é permitido pela transição  $\delta(q_2, a)$  e como visto no caso anterior tem-se que o estado  $q_2$  só pode ser acessado por palavras com prefixo  $bb$ , note porém, que as transições  $\delta(q_2, a) = q_4$  e  $\delta(q_4, a) = q_2$  formam um *loop* e assim pode-se concluir que o acesso a  $q_2$  via  $q_4$  obrigatoriamente é realizado por palavras da forma  $bba^{2n}$  com  $n \geq 1$ .

Note que a palavra  $bb$  pode ser escrita como sendo  $bba^0$ , portanto, pelas duas análises anteriores pode-se concluir que se  $\hat{\delta}(q_0, w) = q_2$ , então  $w = bba^{2n}$  com  $n \in \mathbb{N}$ , e portanto,  $w \in L$ , completando assim a prova.  $\square$

Exemplo 2.1.10 | O AFD  $A$  do Exemplo 2.1.1 reconhece a linguagem  $L = \{a^{2n+1} \mid n \in \mathbb{N}\}$ .

Prova  $(\Rightarrow)$  Suponha que  $w \in L$  assim  $w = a^{2n+1}$ , agora por indução sobre o tamanho das palavras tem-se que,

(B)ase: Quando  $n = 0$  vale a igualdade  $w = a^{2 \cdot 0 + 1}$ , agora usando a definição do AFD  $A$  tem-se que,

$$\widehat{\delta}(q_0, a^{2 \cdot 0 + 1}) = \widehat{\delta}(q_0, a^1) = \delta(\widehat{\delta}(q_0, \lambda), a) = \delta(q_0, a) = q_1$$

e como  $q_1 \in F$  tem-se que  $a^{2 \cdot 0 + 1} \in \mathcal{L}(A)$ , ou seja,  $w \in \mathcal{L}(A)$ .

(H)ipótese indutiva: Suponha que para todo  $n \in \mathbb{N}$  tem-se que  $\widehat{\delta}(q_0, a^{2n+1}) \in F$ , ou seja,  $\widehat{\delta}(q_0, a^{2n+1}) = q_1$ .

(P)asso indutivo: Dado  $w = a^{2(n+1)+1}$  tem-se que,

$$\begin{aligned} \widehat{\delta}(q_0, a^{2(n+1)+1}) &= \widehat{\delta}(q_0, a^{2n+1+2}) \\ &= \widehat{\delta}(q_0, a^{2n+1}aa) \\ &= \delta(\widehat{\delta}(q_0, a^{2n+1}), a), a) \\ &\stackrel{(HI)}{=} \delta(\delta(q_1, a), a) \\ &= \delta(q_0, a) \\ &= q_1 \end{aligned}$$

Logo, por (B), (H) e (P) tem-se que  $\widehat{\delta}(q_0, a^{2n+1}) \in \mathcal{L}(A_1)$  para qualquer que seja  $n \in \mathbb{N}$ .

$(\Leftarrow)$  A volta fica como exercício argumentativo ao leitor.  $\square$

Pode-se agora formalizar a primeira das classes de linguagens sendo esta a classe das linguagens regulares, tal classe foi primeiramente definida por Kleene em seu trabalho [18], entretanto, em tal ocasião tais linguagens foram chamadas de eventos regulares, como será visto é momentos futuros nesse manuscrito a classe das linguagens regulares é aquela que possui o menor nível complexidade computacional.

**Definição 2.6** (Linguagens Regulares) Uma linguagem  $L$  qualquer é dita ser regular se, e somente se, existe um AFD  $A$  tal que  $L = \mathcal{L}(A)$ . A classe de todas as linguagens regulares é denotada por  $\mathcal{L}_{Reg}$ .

## 2.2 Autômatos Finitos Não-determinísticos

Como explicado por Peter Linz em [19], um autômato finito não-determinístico, ou simplesmente AFN, é um autômato que se diferencia dos AFD apenas no quesito da função de transição. A diferença consiste no fato de que, enquanto a imagem da função de transição em um AFD é sempre um estado, nos AFN a imagem da função de transição é um subconjunto de estados, em um sentido moderno da teoria dos autômatos, um AFN seria uma máquina que algumas transições geraria uma superposição de estados [13]. Formalmente um AFN é como se segue.

**Definição 2.7** (Autômato Finito Não-determinístico) Um AFN é uma estrutura  $A = \langle Q, \Sigma, \delta_N, q_0, F \rangle$  onde:  $Q, \Sigma, q_0$  e  $F$  são da mesma forma que na Definição 2.1, já  $\delta_N : Q \times \Sigma \rightarrow \wp(Q)$  é uma função total (chamada função de transição não determinística).



**Exemplo 2.2.1** A estrutura  $A = \langle \{q_0, q_1, q_2\}, \{a, b\}, \delta_N, q_0, \{q_0, q_1\} \rangle$  onde a função  $\delta$  é descrita pela Tabela 2.1 a seguir é um AFN.

$Q \backslash \Sigma$	$a$	$b$
$q_0$	$\{q_1\}$	$\{q_0\}$
$q_1$	$\{q_2\}$	$\{q_0, q_2\}$
$q_2$	$\{q_2\}$	$\{q_1\}$

Tabela 2.1: Tabela de transição para a função  $\delta_N$  do AFN no Exemplo 2.2.1.

Quanto a representação visual de um AFN usando grafos de transição é construída exatamente da mesma forma que a representação de um AFD, a única diferença é o fato de poder existir múltiplas arestas rotuladas por um símbolo  $a \in \Sigma$  saindo de um vértice  $q_i$  e chegando em diferentes vértices  $q_j \in X$  onde  $X \subseteq Q$  e  $\delta_N(q_i, a) = X$ .

**Exemplo 2.2.2** O grafo de transição representado na Figura 2.5 a seguir é uma representação para o AFN do Exemplo 2.2.1.

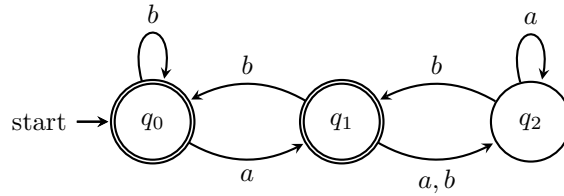


Figura 2.5: Grafo de transição do AFN do Exemplo 2.2.1.

**Atenção** As transições da forma  $\delta_N(q, a) = \emptyset$ , para algum  $q \in Q$ , não são representadas no grafo de transição de um AFN.

Como para o caso determinístico a função de transição, neste caso  $\delta_N$ , pode ser estendida para uma função  $\widehat{\delta}_N$  usando recursividade como se segue.

**Definição 2.8** (Transição não-determinística estendida) Seja  $A = \langle Q, \Sigma, \delta_N, q_0, F \rangle$  um AFN, a função de transição estendida é uma função  $\delta_N : Q \times \Sigma^* \rightarrow \wp(Q)$  definida pela seguinte recursão.

$$\widehat{\delta}_N(q, \lambda) = \{q\} \quad (2.4)$$

$$\widehat{\delta}_N(q, wa) = \bigcup_{q' \in \widehat{\delta}_N(q, w)} \delta_N(q', a) \quad (2.5)$$

Como para os AFD a noção de computação em qualquer AFN consiste simplesmente da aplicação da função  $\widehat{\delta}_N$  sobre alguma palavra  $w \in \Sigma^*$  e um estado  $q$ .

**Exemplo 2.2.3** Considerando o AFN ilustrado na Figura 2.5 e a palavra “abb” tem-se que,

$$\widehat{\delta}_N(q_0, abb) = \bigcup_{q' \in \widehat{\delta}_N(q_0, ab)} \delta_N(q', b) \quad (2.6)$$

mas tem-se que,

$$\widehat{\delta}_N(q_0, ab) = \bigcup_{q'' \in \widehat{\delta}_N(q_0, a)} \delta_N(q'', b) \quad (2.7)$$

e

$$\begin{aligned}
 \widehat{\delta}_N(q_0, a) &= \bigcup_{q''' \in \widehat{\delta}_N(q_0, \lambda)} \delta_N(q''', a) \\
 &= \bigcup_{q''' \in \{q_0\}} \delta_N(q''', a) \\
 &= \delta_N(q_0, a) \\
 &= \{q_1\}
 \end{aligned} \tag{2.8}$$

substituindo a Equação (2.8) na Equação (2.7) tem-se que,

$$\widehat{\delta}_N(q_0, ab) = \{q_0, q_2\} \tag{2.9}$$

e finalmente substituindo a Equação (2.9) na Equação (2.6) tem-se que,

$$\widehat{\delta}_N(q_0, aba) = \{q_0, q_1\} \tag{2.10}$$

 ou seja, a computação da palavra “abb” pelo AFN da Figura 2.5 termina no conjunto de estados  $\{q_0, q_1\}$ .

Pelo exemplo anterior o leitor mais atento pode ter notado que diferente do caso determinístico, a computação em um AFN não é linear, no sentido de que não existe um único caminho de computação<sup>1</sup>, em vez disso, a computação em um AFN pode ser vista como uma árvore  $n$ -ária em que a união dos estados em cada nível da árvore representa a superposição de estados assumida pela unidade de controle do autômato a cada símbolo consumido da palavra  $w$ , ou seja, cada nível da árvore é gerado pelo “consumo” de um símbolo na memória do AFN, o exemplo a seguir ilustra bem essa ideia de árvore de computação.

<sup>1</sup> Como explicado em [13] um caminho de computação é uma sequência finita de estados assumidos pela unidade central do autômato durante o processamento de uma palavra de entrada.

Exemplo 2.2.4

Considerando o AFN ilustrado na Figura 2.5 e a palavra “abab” tem-se que o processo de computação para tal palavra poder ser representado pela árvore da Figura 2.6 a seguir.

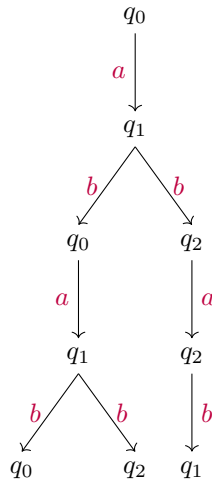


Figura 2.6: Árvore de computação da palavra “abab” no AFN da Figura 2.5.

Outra forma de visualizar a evolução de um AFN (ou AFD) durante o processamento de uma palavra  $w \in \Sigma^*$  é usando a ideia de descrição instantânea (ou descrição de momento).

Pode-se agora apresentar a noção de aceitação (reconhecimento ou computação) de palavras nos AFD.

**Definição 2.9** (Reconhecimento de palavras em AFN) Sejam  $A = \langle Q, \Sigma, \delta_N, q_0, F \rangle$  um AFN e seja  $w \in \Sigma^*$ . A palavra  $w$  é dita aceita (reconhece ou computada) por  $A$  sempre que  $\widehat{\delta}_N(q_0, w) \cap F \neq \emptyset$  e é rejeitada por  $A$  em qualquer outro caso.

Note que a Definição 2.9 pode ser informalmente interpretada da seguinte forma, uma palavra é aceita por um AFN  $A$  se existe pelo menos um caminho de computação para  $w$  que termine em um estado final, isto é, pelo menos uma das folhas na árvore de computação deve ser um estado  $q \in F$ , neste caso  $w$  é aceita por  $A$ .

**Exemplo 2.2.5** Considerando o AFN representado pela Figura 2.7 a seguir e as palavras “aabbba” e “aabb” tem-se que:

$$\widehat{\delta}_N(q_0, aabbbba) = \{q_1, q_3\}$$

e

$$\widehat{\delta}_N(q_0, aabb) = \{q_2, q_3\}$$

logo a palavra “aabbba” é aceita por tal AFN. Por outro lado, a palavra “aabb” não é aceita pelo AFN.

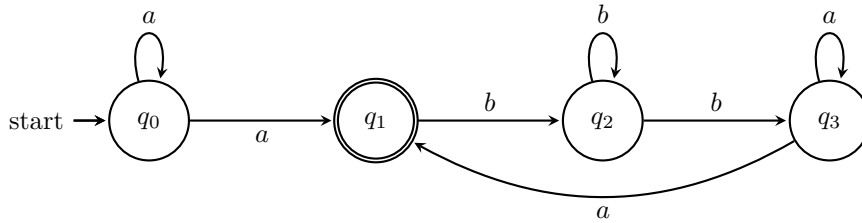


Figura 2.7: Grafo de transição de um AFN.

Usando a definição apresentada anteriormente de palavra aceita pode-se finalmente introduzir formalmente a noção de linguagem aceita (computada ou reconhecida) pelos AFN.

**Definição 2.10** (Linguagem de um AFN) Seja  $A = \langle Q, \Sigma, \delta_N, q_0, F \rangle$  um AFN a linguagem reconhecida (ou computada) por  $A$ , denotada por  $\mathcal{L}(A)$ , corresponde ao conjunto de todas as palavras aceitas por  $A$ , formalmente tem-se que:

$$\mathcal{L}(A) = \{w \in \Sigma^* \mid \widehat{\delta}_N(q_0, w) \cap F \neq \emptyset\} \quad (2.11)$$

De forma similar ao que ocorre com os AFD, para mostrar que uma linguagem  $L$  é aceita por algum AFN  $A$  deve-se provar a igualdade  $L = \mathcal{L}(A)$ , ou seja, deve-se provar que  $w \in L \iff w \in \mathcal{L}(A)$ .

**Exemplo 2.2.6** A linguagem  $L = \{a^i(ba)^j \mid i \geq 1, j \geq 0\}$  é aceita pelo AFN  $A$  representado pelo grafo de transição da Figura 2.8 a seguir.

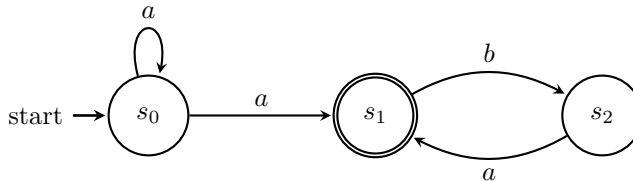


Figura 2.8: Grafo de transição de um AFN.

**Prova** ( $\Rightarrow$ ) Suponha que  $w \in L$ , portanto,  $w = a^m(ba)^n$ , e agora por indução dupla sobre o par  $(m, n)$  tem-se que:

(B)ase: Quando com  $m = 1$  e  $n = 0$  vale a igualdade  $w = a^1(ba)^0 = a$ , agora

usando a definição de  $\delta_N$  do AFN  $A$  como representado na Figura 2.8 tem-se que,

$$\widehat{\delta_N}(s_0, a) = \bigcup_{s' \in \widehat{\delta}(s_0, \lambda)} \delta_N(s', a) = \delta_N(s_0, a) = \{s_0, s_1\}$$

uma vez que,  $s_1 \in F$  tem-se que  $\widehat{\delta_N}(s_0, a) \cap F \neq \emptyset$ , e portanto,  $w \in \mathcal{L}(A)$ . Agora suponha que para  $w = a^1(ba)^n$  com  $n \geq 0$  tem-se que  $\widehat{\delta_N}(s_0, a^1(ba)^n) \cap F \neq \emptyset$ . Assim dado  $a^1(ba)^{n+1}$  por definição tem-se que:

$$\begin{aligned} \widehat{\delta_N}(s_0, a^1(ba)^{n+1}) &= \widehat{\delta_N}(s_0, a^1(ba)^n ba) \\ &= \bigcup_{s' \in \widehat{\delta_N}(s_0, a^1(ba)^n b)} \delta_N(s', a) \end{aligned} \quad (2.12)$$

agora fazendo,

$$K = \bigcup_{s'' \in \widehat{\delta_N}(s_0, a^1(ba)^n)} \delta_N(s'', b) \quad (2.13)$$

e reescrevendo a Equação (2.12) usando a Equação (2.13) tem-se que,

$$\widehat{\delta_N}(s_0, a^1(ba)^{n+1}) = \bigcup_{s' \in K} \delta_N(s', a) \quad (2.14)$$

entretanto, por hipótese tem-se que  $\widehat{\delta_N}(s_0, a^1(ba)^n) \cap F \neq \emptyset$ , consequentemente, tem-se que  $s_1 \in \widehat{\delta_N}(s_0, a^1(ba)^n)$  dessa forma pela Equação (2.13) é claro que  $\delta_N(s_1, b) \subseteq K$ . Mas  $\delta_N(s_1, b) = \{s_2\}$  logo pela Equação (2.14) tem-se que  $\delta_N(s_2, a) \subseteq \widehat{\delta_N}(s_0, a^1(ba)^{n+1})$ , desde que  $\delta_N(s_2, a) = \{s_1\}$ , tem-se  $s_1 \in \widehat{\delta_N}(s_0, a^1(ba)^{n+1})$ , portanto,  $\widehat{\delta_N}(s_0, a^1(ba)^{n+1}) \cap F \neq \emptyset$ , consequentemente  $a^1(ba)^{n+1} \in \mathcal{L}(A)$ .

**(H)** hipótese indutiva: Assuma que  $\widehat{\delta_N}(s_0, a^m(ba)^n) \cap F \neq \emptyset$  com  $n \geq 0$ .

**(P)** asso indutivo: Primeiro seja  $w \in L$  de forma que  $w = a^{m+1}(ba)^0$  logo pela hipótese indutiva segue que,

$$\widehat{\delta_N}(s_0, a^{m+1}(ba)^0) \cap F \neq \emptyset$$

consequentemente,  $a^{m+1}(ba)^0 \in \mathcal{L}(A)$ . Além disso, sendo  $w \in L$  tal que  $w = a^{m+1}(ba)^n$ , usando a definição de  $\widehat{\delta_N}$  tem-se para  $a^{m+1}(ba)^{n+1}$  que,

$$\begin{aligned} \widehat{\delta_N}(s_0, a^{m+1}(ba)^{n+1}) &= \widehat{\delta_N}(s_0, a^{m+1}(ba)^n ba) \\ &= \bigcup_{s' \in \widehat{\delta_N}(s_0, a^{m+1}(ba)^n b)} \delta_N(s', a) \end{aligned} \quad (2.15)$$

agora desenvolvendo o termo  $\widehat{\delta_N}(s_0, a^{m+1}(ba)^n b)$  tem-se

$$\widehat{\delta_N}(s_0, a^{m+1}(ba)^n b) = \bigcup_{s'' \in \widehat{\delta_N}(s_0, a^{m+1}(ba)^n)} \delta_N(s'', b)$$

por **(H)** tem-se que  $\widehat{\delta_N}(s_0, a^{m+1}(ba)^n) \cap F = \emptyset$ , consequentemente, tem-se  $s_1 \in \widehat{\delta_N}(s_0, a^{m+1}(ba)^n)$ , e dessa forma é claro que a relação de inclusão,  $\delta_N(s_1, b) \subseteq \widehat{\delta_N}(s_0, a^{m+1}(ba)^n)$  acontece. Agora, uma vez que,  $\delta_N(s_1, b) = \{s_2\}$ , tem-se que  $\{s_2\} \subseteq \widehat{\delta_N}(s_0, a^{m+1}(ba)^n)$ , assim pela Equação (2.15) segue que  $\delta_N(s_2, a) \subseteq \widehat{\delta_N}(s_0, a^{m+1}(ba)^{n+1})$ , mas por definição  $\delta_N(s_2, a) = \{s_1\}$ , portanto, tem-se que  $\{s_1\} \subseteq \widehat{\delta_N}(s_0, a^{m+1}(ba)^{n+1})$ , logo  $\widehat{\delta_N}(s_0, a^{m+1}(ba)^{n+1}) \cap F \neq \emptyset$  e assim  $a^{m+1}(ba)^{n+1} \in \mathcal{L}(A)$ .

Consequentemente por **(B)**, **(H)** e **(P)** segue que  $\widehat{\delta}_N(s_0, a^m(ba)^n) \cap F \neq \emptyset$  para todo  $m \geq 1$  e  $n \in \mathbb{N}$ .

( $\Leftarrow$ ) Suponha que  $w \in \mathcal{L}(A)$  assim  $s_1 \in \widehat{\delta}_N(s_0, w)$ , note porém que  $s_1$  só é acessível a partir de duas transições:

- (1)  $\delta_N(s_0, a)$  e
- (2)  $\delta_N(s_2, a)$ .

Note que devido ao *loop* fornecido pelo fato de que  $s_0 \in \delta_N(s_0, a)$  a transição (1) pode ser executada  $m$  vezes com  $m \geq 1$ , em que para cada execução um novo ramo com o estado  $s_1$  é gerado na árvore de computação de  $A$ , entretanto, executar  $m$  vezes a transição  $\delta_N(s_0, a)$  implica em executar a computação  $\widehat{\delta}_N(s_0, a^m)$ , pelo fato<sup>a</sup> de que  $s_1 \in \widehat{\delta}_N(s_0, a^m)$  tem-se que  $a^m \in \mathcal{L}(A)$ , e uma vez que  $a^m = a^m(ba)^0$  tem-se que a primeira forma de  $\widehat{\delta}_N(s_0, w) \cap F \neq \emptyset$  é que  $w = a^m(ba)^0$  e assim  $w \in L$ . Por outro lado, para acessar  $s_1$  via a transição (2) é necessário antes chegar a um ramo de computação em que o estado  $s_2$  seja uma folha, mas pela definição de  $A$  isso só é possível se a transição  $\delta_N(s_1, b)$  for usada, note entretanto, que as transições  $\delta_N(s_1, b) = \{s_2\}$  e  $\delta_N(s_2, a) = \{s_1\}$  também geram um *loop* que pode ser executado  $n$  vezes com  $n \geq 0$ , mas executar esse *loop*  $n$  vezes corresponde a executar  $\widehat{\delta}_N(s_1, (ba)^n)$ , e como dito anteriormente,  $s_1$  só é acessível pela definição de  $A$  usando a computação  $\widehat{\delta}_N(s_0, a^m)$ , portanto, para que  $s_1 \in \widehat{\delta}_N(s_0, w) \cap F$ , obrigatoriamente,  $w = a^m(ba)^n$  com  $m \geq 1, n \geq 0$ , e portanto,  $w \in L$ .  $\square$

<sup>a</sup>Fica para o leitor a tarefa de provar que para todo  $m \geq 1$  tem-se que  $s_1 \in \widehat{\delta}_N(s_0, a^m)$ .

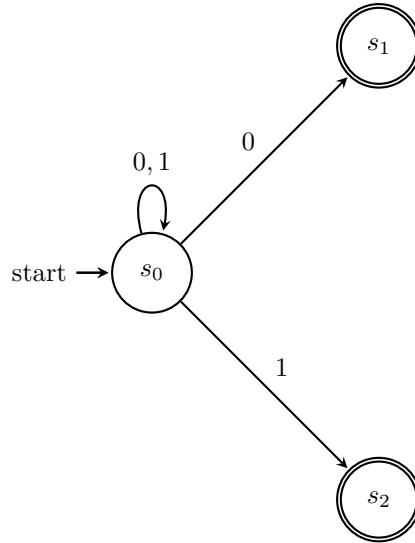


Figura 2.9: Grafo de transição de um AFN  $S$ .

**Exemplo 2.2.7** | O AFN  $S$  representado no grafo de transição exposto na Figura 2.9 a seguir reconhece a linguagem  $L = \{uv \mid u \in \{0, 1\}^*, v \in \{0, 1\}\}$ .

**Prova** | ( $\Rightarrow$ ) A ida fica a cargo do leitor. ( $\Leftarrow$ ) Suponha que  $w \in \mathcal{L}(A)$  assim por definição  $\widehat{\delta}_N(s_0, w) \cap \{s_1, s_2\} \neq \emptyset$ , agora pela definição de  $\delta_N$  é claro que toda árvore de computação de  $A$  apresenta a propriedade de sempre conter um dos estados  $s_1$  ou  $s_2$ , mas nunca os dois simultaneamente<sup>2</sup>. Além disso, o fato de

$$s_0 \in \widehat{\delta}_N(s_0, a)$$

para todo  $a \in \{0, 1\}$ , garante que qualquer palavra não a vazia  $u$  sobre o alfabeto  $\{0, 1\}$  pode ser gerada, por fim, no último passo de computação é claro que  $s_1$  ou

<sup>2</sup> A prova desta propriedade fica como exercício ao leitor.

$s_2$  será uma folha da árvore, entretanto,  $s_1$  só será tal folha no caso da palavra terminar em 0 caso contrário a folha será  $s_2$ , e portanto, todo  $w \in \mathcal{L}(A)$  tem a forma  $uv$  com  $u \in \{0,1\}^*$  e  $v \in \{0,1\}$ , consequentemente  $w \in L$ .  $\square$

De forma ingênua o leitor pode vir a imaginar que a possibilidade da unidade de controle de um AFN poder assumir mais de um estado interno simultaneamente, faz com que os AFN sejam mais poderosos que os AFD, entretanto, como será exibido pelos resultados a seguir, isso não ocorre, de fato, como dito [8, 19] apesar de tornar mais fácil a tarefa de construir um autômato quem reconheça uma linguagem  $L$ , o não-determinismo não aumenta nem nada o poder de computação dos autômatos finitos.

**Teorema 3** (Transformação AFD - AFN) Se  $L = \mathcal{L}(A)$  para algum AFD  $A$ , então existe um AFN  $A'$  tal que  $L = \mathcal{L}(A')$ .

**Prova** A prova é trivial, uma vez que, todo AFD  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  pode ser convertido em um AFN  $A' = \langle Q, \Sigma, \delta_N, q_0, F \rangle$  apenas realizando as transformações das transições  $\delta(q_i, a) = q_j$  nas transições não-determinísticas  $\delta_N(q_i, a) = \{q_j\}$  e mantendo todo o resto da estrutura igual.  $\square$

O próximo resultado estabelece a contraparte do Teorema 3, isto é, tal resultado mostrará que sempre é possível obter um AFD que pode “simular”. O termo simular aqui, diz respeito a ideia de que cada aplicação de uma função de transição não-determinística pode ser representada de forma precisa por uma aplicação de uma função de transição determinística, para detalhes consulte [17, 25].

**Teorema 4** (Transformação AFN - AFD) Se  $L = \mathcal{L}(A)$  para algum AFN  $A$ , então existe um AFD  $A'$  tal que  $L = \mathcal{L}(A')$ .

**Prova** Suponha que  $L = \mathcal{L}(A)$  para algum AFN  $A = \langle Q, \Sigma, \delta_N, q_0, F \rangle$ , agora é construído um autômato  $A' = \langle \wp(Q), \Sigma, \delta, \{q_0\}, F' \rangle$  onde para todo  $X \in \wp(Q)$  e  $a \in \Sigma$  tem-se

$$\delta(X, a) = \bigcup_{q \in X} \delta_N(q, a) \quad (2.16)$$

claramente este autômato é realmente determinístico, e para todo  $X \in \wp(Q)$  tem-se que  $X \in F'$  se, e somente se,  $X \cap F \neq \emptyset$ . Agora será mostrado por indução sobre o tamanho de  $w \in \Sigma^*$  que:

$$\widehat{\delta}(\{q_0\}, w) = \widehat{\delta}_N(q_0, w)$$

**(B)ase:** Quando  $|w| = 0$  isto é  $w = \lambda$  tem-se trivialmente pela definição das funções de transição estendidas que  $\widehat{\delta}(\{q_0\}, \lambda) = \widehat{\delta}_N(q_0, \lambda)$ .

**(H)ipótese indutiva:** Suponha que para todo  $w \in \Sigma^*$  com  $|w| \geq 0$  tem-se que  $\widehat{\delta}(\{q_0\}, w) = \widehat{\delta}_N(q_0, w)$ .

**(P)asso indutivo:** Dado  $w = ua$  com  $u \in \Sigma^*$ ,  $|u| \geq 0$  e  $a \in \Sigma$  tem-se que,

$$\begin{aligned} \widehat{\delta}(\{q_0\}, w) &= \widehat{\delta}(\{q_0\}, ua) \\ &= \widehat{\delta}(\widehat{\delta}(\{q_0\}, u), a) \\ &\stackrel{\text{(HI)}}{=} \widehat{\delta}(\widehat{\delta}_N(q_0, u), a) \\ &\stackrel{\text{Eq. (2.16)}}{=} \bigcup_{q \in \widehat{\delta}_N(q_0, u)} \delta_N(q, a) \\ &= \widehat{\delta}_N(q_0, ua) \\ &= \widehat{\delta}_N(q_0, w) \end{aligned}$$

Portanto, pode-se concluir por **(B)**, **(H)** e **(P)** que  $w \in \mathcal{L}(A)$  se, e somente se,  $w \in \mathcal{L}(A')$ , ou seja,  $L = \mathcal{L}(A')$  o que completa a prova.  $\square$

Observe que o método de construção usado na prova do Teorema 4 cria um AFD cujo número de estado cresce em razão de uma potência de 2 quando comparado com o quantitativo de estados do AFN original. Como consequência deste resultado segue o seguinte corolário.

**Corolário 2** Uma linguagem  $L$  é regular se, e somente se, existe um AFN  $A$  tal que  $L = \mathcal{L}(A)$ .

*Prova*  $(\Rightarrow)$  Assuma que  $L$  é regular, assim por definição existe um AFD  $A'$  tal que  $L = \mathcal{L}(A')$ , entretanto, pelo Teorema 3 existe um AFN  $A$  tal que  $L = \mathcal{L}(A)$ .  $(\Leftarrow)$  Suponha que  $L = \mathcal{L}(A)$  para algum AFN  $A$ , agora pelo Teorema 4 existe um AFD  $A'$  tal que  $L = \mathcal{L}(A')$ , e portanto, por definição  $L$  é regular.  $\square$

É importante destacar que o método de construção do AFD usado na prova do Teorema 4, conhecido como método de construção das partes introduzido por Rabin e Scott em [30], tem a característica de poder vir a produzir durante sua execução alguns estados inacessíveis<sup>3</sup> no AFD resultante.

Outro ponto sobre o método de construção das partes é que em alguns cenários pode ser tornar impraticável, pois se o AFN de entrada possuir  $n$  estados, o AFD resultante do método terá  $2^n$  estados, ou seja, o crescimento no número de estados do AFD resultante do método cresce proposicional a uma potência de 2, o que rapidamente gera um número exponencialmente grande de estados.

A seguir o leitor será apresentado a uma melhoria no algoritmo de construção das partes, no sentido de que, a execução de tal algoritmo não produz estados inacessíveis no AFD de saída, o algoritmo a seguir é um pseudo-código baseado na versão textual apresentada no livro de Bedregal *et al.* [8]. A melhoria no Algoritmo 1 consiste do fato dele não considerar simplesmente o conjunto  $\wp(Q)$  no AFD de saída, em vez disso, ele constrói iterativamente um conjunto de estados  $Q' \subseteq \wp(Q)$ , que no pior caso tem-se que  $Q' = \wp(Q)$ .

<sup>3</sup> Um estado  $q$  em um AFD é dito inacessível se não existe um  $w \in \Sigma^*$  tal que  $\hat{\delta}(q_0, w) = q$ . Vale também ressaltar como destaque em [8, 17] que estados inacessíveis não aumentam o poder de computação nos AFD.

---

**Algoritmo 1:** Algoritmo para converter AFN em AFD sem estados inacessíveis.

---

**Entrada:** Um AFN  $A = \langle Q, \Sigma, \delta_N, q_0, F \rangle$

**Saída:** Um AFD  $A' = \langle Q', \Sigma, \delta, \{q_0\}, F' \rangle$

---

1 **início**

2     Inicialize os conjuntos  $Q_u$  e  $Q'$  com um estado rotulado por  $\{q_0\}$

3     Inicialize o conjunto  $F'$  como sendo vazio

4     **repita**

5         Selecione um estado  $X \in Q_u$

6         **para cada**  $a \in \Sigma$  **faça**

7             Determine o conjunto  $Y = \bigcup_{q \in X} \delta_N(q, a)$

8             **se**  $Y \notin Q'$  **então**

9                 Adicione o estado rotulado por  $Y$  em  $Q'$  e em  $Q_u$

10             **fim**

11             Defina a transição  $\delta(X, a) = Y$

12         **fim**

13         Remova  $X$  de  $Q_u$

14     **até**  $Q_u = \emptyset$ ;

15     **para cada**  $X \in Q'$  **faça**

16         **se**  $X \cap F \neq \emptyset$  **então**

17             Adicione  $X$  ao conjunto  $F'$

18         **fim**

19     **fim**

20     **retorna**  $A' = \langle Q', \Sigma, \delta, \{q_0\}, F' \rangle$

21 **fim**

---

**Exemplo 2.2.8** Usando o Algoritmo 1 tendo o AFN representado pelo grafo de transição da Figura 2.7 como entrada será obtido o AFD  $M = \langle \{s_0, s_1, s_2, s_3, s_4, \emptyset\}, \{a, b\}, \delta, s_0, F' \rangle$ , onde tem-se os estados equivalentes aos seguintes conjuntos,

$$\begin{aligned} s_0 &= \{q_0\} \\ s_1 &= \{q_0, q_1\} \\ s_2 &= \{q_2\} \\ s_3 &= \{q_2, q_3\} \\ s_4 &= \{q_1, q_3\} \end{aligned}$$

tem-se que  $F' = \{s_1, s_4\}$  e a função de transição  $\delta$  é como se segue.

$Q \backslash \Sigma$	$a$	$b$
$s_0$	$s_1$	$\emptyset$
$s_1$	$s_1$	$s_2$
$s_2$	$\emptyset$	$s_3$
$s_3$	$s_4$	$s_3$
$s_4$	$s_4$	$s_2$
$\emptyset$	$\emptyset$	$\emptyset$

Tabela 2.2: Tabela da função de transição do AFD  $M$  obtido a partir do uso do Algoritmo 1 no AFN da Figura 2.7.

## 2.3 $\lambda$ -Autômatos Finitos Não-determinísticos

Os  $\lambda$ -Autômatos Finitos Não-determinísticos, ou simplesmente,  $\lambda$ -AFN são como dito em [25], uma generalização do modelo de AFN que foi introduzido na seção anterior e que são permitidas transições entre estados diferentes usando (ou consumindo) a palavra vazia, tais transições recebem o nome de  $\lambda$ -transições, a seguir tais autômatos serão apresentados formalmente.

**Definição 2.11** ( $\lambda$ -Autômatos Finitos Não-determinísticos) Um  $\lambda$ -AFN é uma estrutura  $A = \langle Q, \Sigma, \underline{\delta}_N, q_0, F \rangle$  onde:  $Q, \Sigma, q_0$  e  $F$  são da mesma forma que na Definição 2.1, já  $\underline{\delta}_N : Q \times (\Sigma \cup \{\lambda\}) \rightarrow \wp(Q)$  é uma função total (chamada  $\lambda$ -função de transição não determinística).

A representação usando grafos de transição dos  $\lambda$ -AFN é similar a representação dos AFN da seção anterior, a única diferença é que podem haver transições rotuladas pelo símbolo  $\lambda$ , isto é, podem existir no grafo arestas entre vértices que são rotuladas por  $\lambda$ , e o mesmo vale para a representação das árvores de computação.

**Exemplo 2.3.1** A estrutura  $A = \langle \{q_0, q_1, q_2\}, \{0, 1\}, \underline{\delta}_N, q_0, \{q_0\} \rangle$  com  $\underline{\delta}_N$  sendo especificada pela Tabela 2.3 a seguir é um  $\lambda$ -AFN.

$Q \backslash \Sigma \cup \{\lambda\}$	0	1	$\lambda$
$q_0$	$\emptyset$	$\emptyset$	$\{q_1\}$
$q_1$	$\{q_1\}$	$\{q_2\}$	$\{q_2\}$
$q_2$	$\{q_2\}$	$\{q_2\}$	$\{q_0, q_2\}$

Tabela 2.3: Tabela de transição para a função  $\underline{\delta}_N$  do AFN no Exemplo 2.3.1.

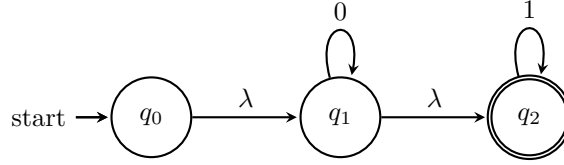
**Exemplo 2.3.2** O  $\lambda$ -AFN  $A = \langle \{q_0, q_1, q_2\}, \{0, 1\}, \underline{\delta}_N, q_0, \{q_2\} \rangle$  com  $\underline{\delta}_N$  sendo especificada pela Tabela 2.4 a seguir é um  $\lambda$ -AFN.



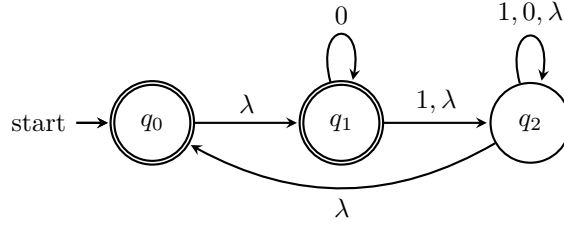
$Q \backslash \Sigma \cup \{\lambda\}$	0	1	$\lambda$
$q_0$	$\emptyset$	$\emptyset$	$\{q_1\}$
$q_1$	$\{q_1\}$	$\emptyset$	$\{q_2\}$
$q_2$	$\emptyset$	$\{q_2\}$	$\emptyset$

 Tabela 2.4: Tabela de transição para a função  $\delta_N$  do AFN no Exemplo 2.3.2.

e representado pela figura 2.10 a seguir.


 Figura 2.10: Grafo de transição do  $\lambda$ -AFN do Exemplo 2.3.2.

**Exemplo 2.3.3** O grafo de transição representado na Figura 2.11 a seguir é uma representação para o  $\lambda$ -AFN do Exemplo 2.3.1.


 Figura 2.11: Grafo de transição do  $\lambda$ -AFN do Exemplo 2.3.1.

Uma interpretação para as transições da forma  $\delta_N(q, \lambda) = X$  é que a unidade de controle do autômato consegue mudar seu estado interno  $q$  para um subconjunto de estados  $X$  sem precisar acessar a memória.

Note porém que a definição da função de transição  $\delta_N$  garante que as transições em um  $\lambda$ -AFN acontecem apenas em duas situações, a primeira em relação símbolos individuais do alfabeto  $\Sigma$  e a segunda com relação a palavra vazia, assim não existe uma forma de computar uma palavra  $w$  de forma que  $|w| > 1$ . A saída para contorna esse fato é estender a função de transição do autômato, similarmente ao que é feito para os AFD e AFN, para isso entretanto, é necessária algumas definições adicionais.

**Definição 2.12** (Função  $\delta_\lambda$ ) Seja  $A = \langle Q, \Sigma, \delta_N, q_0, F \rangle$  um  $\lambda$ -AFN, então a função  $\delta_\lambda : Q \rightarrow \wp(Q)$  é definida como,

$$\delta_\lambda(q) = \bigcup_{i=0}^n \lambda\text{-fecho}^i(q) \quad (2.17)$$

onde  $n = \#Q - 1$  e

$$\lambda\text{-fecho}^0(q) = \{q\} \quad (2.18)$$

$$\lambda\text{-fecho}^{i+1}(q) = \bigcup_{q' \in \lambda\text{-fecho}^i(q)} \delta_N(q', \lambda) \quad (2.19)$$

**Exemplo 2.3.4** Considere o  $\lambda$ -AFN da Figura 2.10 tem-se para o estado  $q_1$  que,

$$\begin{aligned} \delta_\lambda(q_1) &= \bigcup_{i=0}^2 \lambda\text{-fecho}^i(q_1) \\ &= \lambda\text{-fecho}^2(q_1) \cup \lambda\text{-fecho}^1(q_1) \cup \lambda\text{-fecho}^0(q_1) \end{aligned} \quad (2.20)$$

sabe-se que,

$$\lambda\text{-fecho}^0(q_1) = \{q_1\}$$

e desenvolvendo  $\lambda\text{-fecho}^2(q_1)$  tem-se que,

$$\lambda\text{-fecho}^2(q_1) = \bigcup_{q' \in \lambda\text{-fecho}^1(q_1)} \underline{\delta_N}(q', \lambda)$$

mas,

$$\begin{aligned} \lambda\text{-fecho}^1(q_1) &= \bigcup_{q' \in \lambda\text{-fecho}^0(q_1)} \underline{\delta_N}(q', \lambda) \\ &= \bigcup_{q' \in \{q_1\}} \underline{\delta_N}(q', \lambda) \\ &= \{q_2\} \end{aligned}$$

assim,

$$\lambda\text{-fecho}^2(q_1) = \bigcup_{q' \in \{q_2\}} \underline{\delta_N}(q', \lambda) = \underline{\delta_N}(q_2, \lambda) = \emptyset$$

substituindo tais resultados na Equação 2.20 tem-se que,

$$\delta_\lambda(q_1) = \emptyset \cup \{q_2\} \cup \{q_1\} = \{q_1, q_2\}$$

Uma interpretação semântica para a função  $\delta_\lambda$  é que ela representa a resposta ao questionamento: “Estando no estado  $q$  e executando  $n$   $\lambda$ -transições qual subconjunto de estados a unidade central do autômato irá assumir?”.

**Definição 2.13**

(Função  $\widehat{\delta_\lambda}$ ) Seja  $A = \langle Q, \Sigma, \underline{\delta_N}, q_0, F \rangle$  um  $\lambda$ -AFN, então a função  $\widehat{\delta_\lambda} : \wp(Q) \rightarrow \wp(Q)$  é definida como,

$$\widehat{\delta_\lambda}(X) = \bigcup_{q \in X} \delta_\lambda(q) \quad (2.21)$$

*Exemplo 2.3.5*

Considere o  $\lambda$ -AFN da Figura 2.11 tem-se para o conjunto  $\{q_1, q_2\}$  que,

$$\begin{aligned} \widehat{\delta_\lambda}(\{q_1, q_2\}) &= \bigcup_{q \in \{q_1, q_2\}} \delta_\lambda(q) \\ &= \delta_\lambda(q_1) \cup \delta_\lambda(q_2) \\ &= \{q_0, q_1, q_2\} \cup \{q_0, q_2, q_1\} \\ &= \{q_0, q_2, q_1\} \end{aligned}$$

Agora usando as definições de  $\delta_\lambda$  e  $\widehat{\delta_\lambda}$  pode-se apresentar a extensão da função de transição dos  $\lambda$ -AFN.

**Definição 2.14**

( $\lambda$ -Transição não-determinística estendida) Dado um  $\lambda$ -AFN  $A = \langle Q, \Sigma, \underline{\delta_N}, q_0, F \rangle$ , a função  $\underline{\delta_N}$  é estendido para a função  $\widehat{\underline{\delta_N}} : Q \times \Sigma^* \rightarrow \wp(Q)$  definida pela seguinte recursão:

$$\widehat{\underline{\delta_N}}(q, \lambda) = \delta_\lambda(q) \quad (2.22)$$

$$\widehat{\underline{\delta_N}}(q, wa) = \bigcup_{q' \in \widehat{\underline{\delta_N}}(q, w)} \widehat{\delta_\lambda}(\underline{\delta_N}(q', a)) \quad (2.23)$$

*Exemplo 2.3.6*

Considere o  $\lambda$ -AFN da Figura 2.11 tem-se a seguinte computação para a palavra

“10”:

$$\begin{aligned}
 \widehat{\delta_N}(q_0, 10) &= \bigcup_{q' \in \widehat{\delta_N}(q_0, 1)} \widehat{\delta_\lambda}(\delta_N(q', 0)) \\
 &= \bigcup_{q' \in \widehat{\delta_N}(q_0, 1)} \widehat{\delta_\lambda}(\delta_N(q', 0)) \quad (2.24)
 \end{aligned}$$

mas,

$$\begin{aligned}
 \widehat{\delta_N}(q_0, 1) &= \bigcup_{q' \in \widehat{\delta_N}(q_0, \lambda)} \widehat{\delta_\lambda}(\delta_N(q', 1)) \\
 &= \bigcup_{q' \in \delta_\lambda(q_0)} \widehat{\delta_\lambda}(\delta_N(q', 1)) \\
 &= \bigcup_{q' \in \{q_0, q_1, q_2\}} \widehat{\delta_\lambda}(\delta_N(q', 1)) \quad (2.25) \\
 &= \widehat{\delta_\lambda}(\{q_1, q_2\}) \\
 &= \{q_0, q_1, q_2\}
 \end{aligned}$$

substituindo o valor da Equação (2.25) na Equação (2.24) tem-se que,

$$\begin{aligned}
 \widehat{\delta_N}(q_0, 10) &= \bigcup_{q' \in \{q_0, q_1, q_2\}} \widehat{\delta_\lambda}(\delta_N(q', 0)) \\
 &= \{q_0, q_1, q_2\}
 \end{aligned}$$

Assim como para o caso dos AFN uma palavra qualquer  $w \in \Sigma^*$  será dita aceita por um  $\lambda$ -AFN quando a computação da palavra  $w$  para em pelo menos um estado final, ou seja,  $w$  é reconhecida pelo  $\lambda$ -AFN sempre que  $\widehat{\delta_N}(q_0, w) \cap F \neq \emptyset$ , e assim pode-se definir formalmente a noção de linguagem para os  $\lambda$ -AFN.

**Definição 2.15**

(Linguagem de um  $\lambda$ -AFN) Seja  $A = \langle Q, \Sigma, \delta_N, q_0, F \rangle$  um  $\lambda$ -AFN a linguagem aceita por  $A$ , denotado por  $\mathcal{L}(A)$ , corresponde ao seguinte conjunto  $\mathcal{L}(A) = \{w \in \Sigma^* \mid \widehat{\delta_N}(q_0, w) \cap F \neq \emptyset\}$ .

Os aspectos relacionados a mostrar que um  $\lambda$ -AFN reconhece uma linguagem  $L$  são similares ao mesmo aspectos com respeito aos AFN.

**Exemplo 2.3.7**

O  $\lambda$ -AFN representado pelo grafo de transição da Figura 2.12 a seguir reconhece a linguagem  $L = \{w \in \{1, 2, 3\}^* \mid w = 1^i 2^j 3^k \text{ com } i, j, k \in \mathbb{N}\}$ .

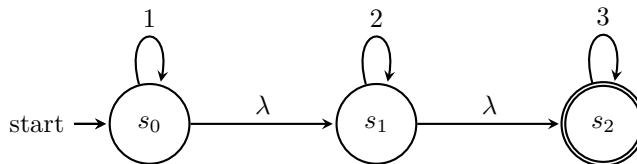
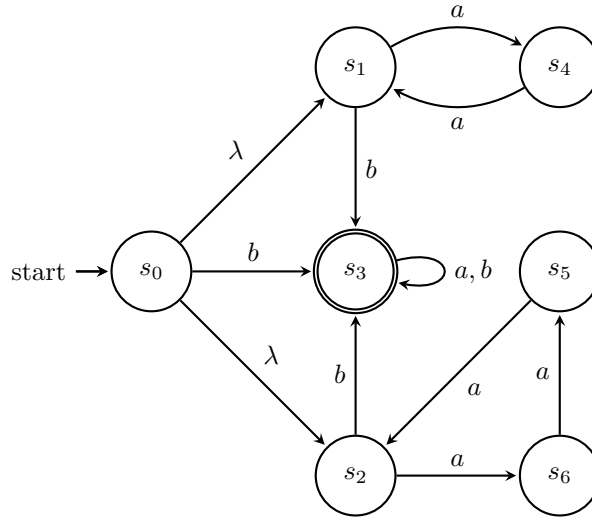


Figura 2.12: Grafo de transição do  $\lambda$ -AFN do Exemplo 2.3.7.


 Figura 2.13: Grafo de transição do  $\lambda$ -AFN do Exemplo 2.3.8.

**Exemplo 2.3.8** | O  $\lambda$ -AFN representado pelo grafo de transição esboçado pela Figura 2.13, aceita a linguagem  $L = \{uv \mid u \in \{a\}^*, |u|_a = 2k \text{ ou } |u|_a = 3k, v = bx, x \in \{a, b\}^*, k \in \mathbb{N}\}$ .

**Teorema 5** (Transformação  $\lambda$ -AFN-AFD) Se  $L = \mathcal{L}(A)$  para algum  $\lambda$ -AFN  $A$ , então existe um AFD  $A'$  tal que  $L = \mathcal{L}(A')$ .

**Prova** Suponha que  $L = \mathcal{L}(A)$  para algum  $\lambda$ -AFN  $A = \langle Q, \Sigma, \delta_N, q_0, F \rangle$  agora defina o seguinte o autômato  $A' = \langle \wp(Q), \Sigma, \delta, \delta_\lambda(q_0), F' \rangle$  onde para todo  $X \in \wp(Q)$  tem-se que  $X \in F'$  se, e somente se,  $X \cap F \neq \emptyset$ , e além disso, para todo  $X \in \wp(Q)$  e  $a \in \Sigma$  tem-se:

$$\delta(X, a) = \bigcup_{q \in X} \widehat{\delta_\lambda}(\delta_N(q, a)) \quad (2.26)$$

por essa construção obviamente esse autômato é um AFD<sup>a</sup>. Agora será mostrado por indução sobre o tamanho de  $w \in \Sigma^*$  que:

$$\widehat{\delta}(\delta_\lambda(q_0), w) = \widehat{\delta_N}(q_0, w)$$

**(B)ase:** Quando  $|w| = 0$  isto é  $w = \lambda$  tem-se trivialmente pela definição das funções de transição estendidas que,

$$\begin{aligned} \widehat{\delta}(\delta_\lambda(q_0), \lambda) &= \delta_\lambda(q_0) \\ &= \widehat{\delta_N}(q_0, \lambda) \end{aligned}$$

**(H)ipótese indutiva:** Suponha que para todo  $w \in \Sigma^*$  com  $|w| \geq 0$  tem-se que  $\widehat{\delta}(\delta_\lambda(q_0), w) = \widehat{\delta_N}(q_0, w)$ .

**(P)asso indutivo:** Dado  $w = ua$  com  $u \in \Sigma^*$ ,  $|u| \geq 0$  e  $a \in \Sigma$  tem-se que,

$$\begin{aligned} \widehat{\delta}(\delta_\lambda(q_0), w) &= \widehat{\delta}(\delta_\lambda(q_0), ua) \\ &= \delta(\widehat{\delta}(\delta_\lambda(q_0), u), a) \\ &\stackrel{\text{(HI)}}{=} \delta(\widehat{\delta_N}(q_0, u), a) \\ &\stackrel{\text{Eq. (2.26)}}{=} \bigcup_{q \in \widehat{\delta_N}(q_0, u)} \widehat{\delta_\lambda}(\delta_N(q, a)) \\ &= \widehat{\delta_N}(q_0, ua) \\ &= \widehat{\delta_N}(q_0, w) \end{aligned}$$

Agora por **(B)**, **(H)** e **(P)** pode-se efetivamente enunciar que  $w \in \mathcal{L}(A)$  se, e somente se,  $w \in \mathcal{L}(A')$ , ou seja,  $L = \mathcal{L}(A')$  o que completa a prova.  $\square$

<sup>a</sup>A prova desse fato fica como exercício ao leitor.

**Teorema 6** (Transformação AFD- $\lambda$ -AFN) Se  $L = \mathcal{L}(A)$  para algum AFD  $A$ , então existe um  $\lambda$ -AFN  $A'$  tal que  $L = \mathcal{L}(A')$ .

Prova | Trivial e ficará como exercício ao leitor.  $\square$

**Corolário 3** Uma linguagem  $L$  é regular se, e somente se, existe um  $\lambda$ -AFN  $A$  tal que  $L = \mathcal{L}(A)$ .

Prova |  $(\Rightarrow)$  Assuma que  $L$  é regular, assim por definição existe um AFD  $A'$  tal que  $L = \mathcal{L}(A')$ , entretanto, pelo Teorema 6 existe um  $\lambda$ -AFN  $A$  tal que  $L = \mathcal{L}(A)$ .  $(\Leftarrow)$  Suponha que  $L = \mathcal{L}(A)$  para algum  $\lambda$ -AFN  $A$ , agora pelo Teorema 5 existe um AFD  $A'$  tal que  $L = \mathcal{L}(A')$ , e portanto, por definição  $L$  é regular.  $\square$

Assim como para o caso da transformação de AFN em AFD, o processo de usar a construção do conjunto das partes no Teorema 5 possui a desvantagem de gera estados inacessíveis. Mas como discutido em [5, 6, 17, 19], algumas simples modificações no Algoritmo 2 fazem com que o novo algoritmo gerado seja capaz de remover as  $\lambda$ -transições e não sejam produzidos estados inacessíveis a seguir é apresentado este novo algoritmo.

---

**Algoritmo 2:** Algoritmo para remoção de  $\lambda$ -transições de um  $\lambda$ -AFN.

---

**Entrada:** Um  $\lambda$ -AFN  $A = \langle Q, \Sigma, \delta_N, q_0, F \rangle$

**Saída:** Um AFD  $A' = \langle Q', \Sigma, \delta, \delta_\lambda(q_0), F' \rangle$

```

1 início
2   Inicialize os conjuntos  $Q_u$  e  $Q'$  com o estado rotulado por  $\delta_\lambda(q_0)$ 
3   Inicialize o conjunto  $F'$  como sendo vazio
4   repita
5     Selecione um estado  $X \in Q_u$ 
6     para cada  $a \in \Sigma$  faça
7       Determine o conjunto  $Y = \widehat{\delta}_\lambda \left( \bigcup_{q \in X} \delta_N(q, a) \right)$ 
8       se  $Y \notin Q'$  então
9         Adicione um estado rotulado por  $Y$  em  $Q'$  e em  $Q_u$ 
10      fim
11      Defina a transição  $\delta(X, a) = Y$ 
12    fim
13    Remova  $X$  de  $Q_u$ 
14  até  $Q_u = \emptyset$ ;
15  para cada  $X \in Q'$  faça
16    se  $X \cap F \neq \emptyset$  então
17      Adicione  $X$  ao conjunto  $F'$ 
18    fim
19  fim
20  retorna  $A' = \langle Q', \Sigma, \delta, \delta_\lambda(q_0), F' \rangle$ 
21 fim

```

---

*Exemplo 2.3.9* | Aplicando o Algoritmo 2 ao  $\lambda$ -AFN do Exemplo 2.3.8 é obtido como saída o AFD:

$$D = \langle \{A_0, A_1, A_2, A_3, A_4, A_5, A_6, A_7, \emptyset\}, \{a, b\}, \delta, \{s_0, s_1, s_2\}, F' \rangle$$

onde tem-se que:

$$\begin{aligned} A_0 &= \{s_0, s_1, s_2\} \\ A_1 &= \{s_4, s_6\} \\ A_2 &= \{s_3\} \\ A_3 &= \{s_1, s_5\} \\ A_4 &= \{s_2, s_4\} \\ A_5 &= \{s_1, s_6\} \\ A_6 &= \{s_4, s_5\} \\ A_7 &= \{s_1, s_2\} \end{aligned}$$

sendo  $F' = \{A_2\}$  e  $\delta$  é descrito pela Tabela a seguir

$\begin{matrix} \Sigma \\ Q \end{matrix}$	$a$	$b$
$A_0$	$A_1$	$A_2$
$A_1$	$A_3$	$\emptyset$
$A_2$	$A_2$	$A_2$
$A_3$	$A_4$	$A_2$
$A_4$	$A_5$	$A_2$
$A_5$	$A_6$	$A_2$
$A_6$	$A_7$	$\emptyset$
$A_7$	$A_1$	$A_2$
$\emptyset$	$\emptyset$	$\emptyset$

Tabela 2.5: Tabela da função de transição do AFD obtido a partir da aplicação do Algoritmo 2 ao  $\lambda$ -AFN do Exemplo 2.3.8.

Nestas últimas seções foram usados os símbolos  $\delta$ ,  $\delta_N$  e  $\underline{\delta}_N$  para denotar as funções de transições dos AFD, AFN e  $\lambda$ -AFN respectivamente, entretanto, o motivo disto foi por questões puramente didáticos, para ajudar o entendimento na conversão entre os tipos de autômatos, mas é comum encontrar na literatura (ver [8, 17, 19]) que independente do tipo de autômato, sua função de transição será sempre representada apenas por  $\delta$ .

## 2.4 Teorema Myhill-Nerode e o Autômato Mínimo

Até agora este manuscrito se preocupou com a tarefa de saber se uma linguagem pode ou não ser reconhecida por um autômato finito, seja ele determinístico ou não-determinístico. Nesta seção será apresentada ao leitor a questão de eficiência no reconhecimento de linguagens em relação aos autômatos finitos, aqui será mostrado que o problema de encontrar um menor AFD que reconhece uma linguagem  $L$  é decidível.

Na teoria dos autômatos quando se usa a palavra “menor”, se está querendo dizer simplesmente aquele com o menor número possível de estados, ou seja, o AFD mínimo. Mais adiante será aqui provado, que esse AFD mínimo é único a menos de isomorfismo, ou seja, se dois AFD reconhecem a mesma linguagem, cada um tendo o menor número possível de estados, então eles são isomórficos. Isso significa que cada linguagem regular está associada com um AFD mínimo.

Este resultado da existência de um AFD mínimo recebe o nome de **Teorema Myhill-Nerode**, em homenagem aos matemáticos John Myhill<sup>4</sup> (1923-1987) e Anil Nerode (1932-), que o provaram na Universidade de Chicago em 1958 no artigo [28], de forma geral tal resultado fornece as condições suficientes e necessárias para que uma linguagem  $L$  seja regular, para construir tal resultado antes é necessário considerar algumas definições básicas e alguns resultados auxiliares.

<sup>4</sup> O professor Myhill também é conhecido por seu Teorema de isomorfismo[27], que pode ser visto como um análogo dentro da teoria da computabilidade ao teorema de Cantor-Bernstein-Schroeder e pelo famoso pelo Teorema de Rice-Myhill-Shapiro, mais comumente conhecido como Teorema de Rice [8, 31].

**Definição 2.16**

(A família  $\mathcal{H}_L$ ) Seja  $L$  uma linguagem qualquer<sup>a</sup> sobre o alfabeto  $\Sigma$ , para qualquer palavra  $w$  é definido o conjunto  $L_w = \{x \mid wx \in L\}$ . A família  $\{L_w \mid w \in \Sigma^*\}$  construída sobre  $L$  será denotada por  $\mathcal{H}_L$ , ou seja,  $\mathcal{H}_L = \{L_w \mid w \in \Sigma^*\}$

<sup>a</sup>Não necessariamente regular.

Com respeito aos conjuntos  $L_w$  o leitor mais atento pode notar que  $L_\lambda = L$ , além disso, os conjuntos  $L_w$  também apresentam a seguinte propriedade básica.

**Proposição 1**

Dado  $L \subseteq \Sigma^*$ . Se  $L_w = L_{w'}$ , então  $L_{wa} = L_{w'a}$  para todo  $a \in \Sigma$ .

**Prova**

Suponha que  $L_w = L_{w'}$ , assim para todo  $a \in \Sigma^*$  tem-se que:

$$\begin{aligned} x \in L_{wa} &\iff wax \in L \\ &\iff ax \in L_w \\ &\stackrel{Hip.}{\iff} ax \in L_{w'} \\ &\iff x \in L_{w'a} \end{aligned}$$

concluindo a prova.  $\square$

Um fato importante sobre AFD que será usado a seguir e que não foi mencionado diretamente até agora é o exposto pelo resultado a seguir.

**Proposição 2**

Se  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  é um AFD, então  $\widehat{\delta}(q_0, uv) = \widehat{\delta}(\widehat{\delta}(q_0, u), v)$  para todo  $u, v \in \Sigma^*$ .

**Prova**

A prova é por indução sobre o tamanho da palavra  $uv$  e ficará como exercício ao leitor.  $\square$

O lema a seguir mostra que  $\#\mathcal{H}_L$  é na verdade um limite inferior para o número de estados em um AFD.

**Lema 2**

Se  $L = \mathcal{L}(A)$  para algum AFD  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ , então  $\#\mathcal{H}_L \leq \#Q$ .

**Prova**

Suponha que  $L = \mathcal{L}(A)$  para algum AFD  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ , agora para todo  $q \in Q$  defina um novo AFD  $A_q$  igual ao anterior em todos os aspectos menos no estado inicial pois este será o estado  $q$ , ou seja,  $A_q = \langle Q, \Sigma, \delta, q, F \rangle$ . Agora para toda palavra  $w \in \Sigma^*$  suponha que  $\widehat{\delta}(q_0, w) = q$ , por definição note que,

$$\begin{aligned} x \in L_w &\stackrel{Def. 2.16}{\iff} wx \in L \\ &\iff wx \in \mathcal{L}(A) \\ &\iff \widehat{\delta}(q_0, wx) \in F \\ &\stackrel{Prop. 2}{\iff} \widehat{\delta}(\widehat{\delta}(q_0, w), x) \in F \\ &\stackrel{Hip.}{\iff} \widehat{\delta}(q, x) \in F \\ &\iff x \in \mathcal{L}(A_q) \end{aligned}$$

Dessa forma tem-se que  $\mathcal{L}(A_q) = L_w$ , e obviamente  $\mathcal{L}(A_{q_0}) = L$ . Desde que  $A$  é fixo, tem-se que  $L_w$  depende apenas do estado obtido quando a computação  $w$  começa em  $q_0$ , e assim o número de  $L_w$  distintos, ou seja, os elementos de  $\mathcal{H}_L$  não pode ser maior que o número de estados em  $A$ , portanto,  $\#\mathcal{H}_L \leq \#Q$ .  $\square$

O próximo lema estabelece que para alguma linguagem  $L$  no caso  $\mathcal{H}_L$  ser finito, então sua cardinalidade será o limite superior no número de estados em um AFD capaz de reconhecer  $L$ .

**Lema 3** Seja  $L \subseteq \Sigma^*$ . Se  $\mathcal{H}_L$  é finito, então existe um AFD  $A_L$  tal que  $L = \mathcal{L}(A_L)$  e  $A_L$  possui exatamente  $\#\mathcal{H}_L$  estados.

**Prova** Dado  $L \subseteq \Sigma^*$  assumamos que  $\mathcal{H}_L$  é finito, dito isto pode-se construir o seguinte AFD  $A_L = \langle \mathcal{H}_L, \Sigma, \delta, q_0, F \rangle$  onde:

$$q_0 = L \quad (2.27)$$

e para todo  $L_w \in \mathcal{H}_L$  e  $a \in \Sigma$  tem-se

$$\delta(L_w, a) = L_{wa} \quad (2.28)$$

e

$$F = \{L_w \in \mathcal{H}_L \mid \lambda \in L_w\} \quad (2.29)$$

sobre a definição de  $A_L$  por indução sobre o tamanho de  $w \in \Sigma^*$  pode-se facilmente verificar que,

$$\hat{\delta}(q_0, w) = L_w \quad (2.30)$$

além disso, claramente  $A_L$  possui exatamente  $\#\mathcal{H}_L$  estados, dito isto, note que:

$$\begin{aligned} w \in L & \stackrel{\text{Def. 2.16}}{\iff} \lambda \in L_w \\ & \stackrel{\text{Eq. (2.29)}}{\iff} L_w \in F \\ & \stackrel{\text{Eq. (2.30)}}{\iff} \hat{\delta}(q_0, w) \in F \\ & \iff w \in \mathcal{L}(A_L) \end{aligned}$$

e portanto,  $L = \mathcal{L}(A_L)$  concluindo assim a prova.  $\square$

**Definição 2.17** (Dimensão de um AFD) Seja  $A = \langle Q, \Sigma, \delta', q_0, F \rangle$  um AFD a dimensão de  $A$ , denotado por  $\dim(A)$ , é igual a quantidade de estados em  $A$ , isto é,  $\dim(A) = \#Q$ .

**Definição 2.18** (AFD Mínimo) Seja  $L$  uma linguagem regular tal que  $L = \mathcal{L}(A)$  para algum um AFD  $A$ . O AFD  $A$  será dito ser mínimo se, e somente se, para todo outro AFD  $B$  tal que  $L = \mathcal{L}(B)$  tem-se que  $\dim(A) \leq \dim(B)$ .

O próximo resultado mostra que não existe nenhum autômato como menos estados que o AFD construído na prova do Lema 3, ou seja, o método de construção mostrado na na prova do Lema 3 gera o AFD mínimo de qualquer linguagem.

**Lema 4** Seja  $L$  uma linguagem regular, assim o AFD  $A_L$  construído no Lema 3 é o único (a menos de isomorfismo) mínimo AFD que aceita  $L$ .

**Prova** Suponha que existe outro AFD mínimo  $N = \langle Q, \Sigma, \delta', q'_0, F' \rangle$  tal que  $L = \mathcal{L}(N)$  diferente do AFD  $A_L = \langle \mathcal{H}_L, \Sigma, \delta, L, F \rangle$  construído na prova do Lema 3. Agora será definida uma função  $f : Q \rightarrow \mathcal{H}_L$  definida simplesmente como:

$$f(q) = \begin{cases} L, & \text{se } q = q_0 \\ \mathcal{L}(A_q), & \text{senão} \end{cases}$$

para todo  $q \in Q$ , onde  $\mathcal{L}(A_q)$  é da mesma forma que na prova do Lema 2 onde o AFD fixo é exatamente  $A_L$ . Por esta definição é claro que:

- (1) Se  $q_i \neq q_j$ , então tem-se que  $f(q_i) \neq f(q_j)$ , consequentemente a função  $f$  é injetora.
- (2)  $f$  preserva a condição de estado inicial.



- (3) Para  $a \in \Sigma, w \in \Sigma^*$  e  $q, p \in Q$  assumamos que  $\delta(q, a) = p$  e  $f(q) = \mathcal{L}(A_q) = L_w$  assim para qualquer  $u \in \Sigma^*$  tem-se que

$$\begin{aligned}
 u \in f(p) &\iff u \in \mathcal{L}(A_p) \\
 &\iff \widehat{\delta}(p, u) \in F \\
 &\iff \widehat{\delta}(\widehat{\delta}(q, a), u) \in F \\
 &\iff \widehat{\delta}(q, au) \in F \\
 &\iff au \in \mathcal{L}(A_q) \\
 &\iff au \in f(q) \\
 &\iff au \in L_w \\
 &\iff wau \in L \\
 &\iff u \in L_{wa}
 \end{aligned}$$

portanto,  $f(p) = L_{wa}$ , ou seja,  $f$  preserva transições<sup>a</sup>.

- (4) Agora note que pela definição de estado final e pela construção de  $A_L$  tem-se que

$$q \in F' \iff \widehat{\delta}(q, \lambda) \in F' \iff \lambda \in \mathcal{L}(A_q) \iff \mathcal{L}(A_q) \in F \iff f(q) \in F$$

portanto, a função  $f$  preserva estados finais.

- (5) Agora dado  $w \in \Sigma^*$  assumamos que  $\widehat{\delta}(q_0, w) = q$ , assim pela prova do Lema 2 para todo  $L_w \in \mathcal{H}_L$ , ou seja, para todo  $L_w \in \text{Ima}(f)$  tem-se que  $L_w = \mathcal{L}(A_q)$ , e portanto, pela definição de  $f$  tem-se que existe  $q \in \text{Dom}(f)$  tal que  $L_w = f(q)$ , ou seja,  $f$  é sobrejetora.

Desde que  $f$  é injetora e sobrejetora tem-se que  $f$  é uma bijeção, e portanto,  $\#Q = \#\mathcal{H}_L$ , logo  $\dim(N) = \dim(A_L)$ . Agora desde que  $f$  preserva o estado inicial, os estados finais e as transições tem-se que  $f$  é um isomorfismo do AFN  $N$  para o AFD  $A_L$ , e portanto, eles são o mesmo AFD se diferenciando apenas pela rotulação dos seus estados.  $\square$

<sup>a</sup>Isto é o mesmo que dizer que  $f(\delta'(q, a)) = \delta(f(q), a)$ .

Pode-se agora finalmente enunciar o Teorema Myhill-Nerode que estabelece uma caracterização para as linguagens regulares.

**Teorema 7** (Teorema Myhill-Nerode) Uma linguagem  $L \subseteq \Sigma^*$  será regular se, e somente se,  $\mathcal{H}_L$  é finito e existe um AFD mínimo  $A$  com exatamente  $\#\mathcal{H}_L$  estados tal que  $\mathcal{L}(A) = L$ .

Prova | Direto dos Lemas 2, 3 e 4.  $\square$

## 2.5 Algoritmos de Minimização de AFD

Na seção anterior foi apresentado o Teorema Myhill-Nerode, que garante a existência de um AFD mínimo para cada uma das linguagens regulares. Neste seção serão apresentados dois métodos alternativos para encontrar o AFD mínimo a partir um AFD dado. Esse métodos são algoritmos baseados na ideia de estados distinguíveis, o primeiro deles a ser apresentado aqui é o algoritmo descrito pela primeira vez em 1956 por Edward Moore [15, 26]<sup>5</sup>, em seguida será apresentado o algoritmo de Hopcroft [16], apresentado inicialmente em 1971.

**Definição 2.19** (Estados Equivalentes) Seja  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  um AFD. A relação de equivalência entre dois estados  $q, q' \in Q$  será denotado por  $q \equiv q'$  e será verdadeira quando para todo  $w \in \Sigma^*$  tem-se que  $\widehat{\delta}(q, w) \in F \iff \widehat{\delta}(q', w) \in F$ .

<sup>5</sup> A critério de curiosidade, se o AFD não possui estados inacessíveis, então o algoritmo de minimização de Moore roda em complexidade  $O(\#Q^2 \# \Sigma)$ .

Antes de apresentar os algoritmos em si, como esse texto foi escrito para aulas em um curso de Ciência da Computação, acho por pertinente, apresentar alguns resultados a respeito de estados equivalentes de um AFD, o primeiro resultado, apresentado a seguir, diz que se dois estados são equivalentes, então seus sucessores também o são.

**Lema 5** Dado um AFD  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ . Se  $q \equiv q'$ , então  $\delta(q, a) \equiv \delta(q', a)$ .

**Prova** Dado  $a \in \Sigma$ , por contrapositiva será mostrado que:

$$\text{Se } \delta(q, a) \not\equiv \delta(q', a), \text{ então } q \not\equiv q'.$$

Inicialmente assuma que  $\delta(q, a) \not\equiv \delta(q', a)$ , assim pela Definição 2.19 existe um  $w \in \Sigma^*$  tal que um dos dois casos a seguir acontece:

$$(1) \widehat{\delta}(\delta(q, a), w) \in F \text{ e } \widehat{\delta}(\delta(q', a), w) \notin F \text{ ou}$$

$$(2) \widehat{\delta}(\delta(q, a), w) \notin F \text{ e } \widehat{\delta}(\delta(q', a), w) \in F.$$

em particular quando  $w = \lambda$ , para o primeiro caso (a prova é similar para o caso (2)) tem-se pela Definição 2.2 que:

$$\widehat{\delta}(\delta(q, a), w) \in F \text{ e } \widehat{\delta}(\delta(q', a), w) \notin F \iff \delta(q, a) \in F \text{ e } \delta(q', a) \notin F$$

e desde que  $a \in \Sigma^*$  pela Definição 2.19 tem-se que  $q \not\equiv q'$ , o que conclui a prova da contrapositiva. Desde que a contrapositiva é verdadeira tem-se que afirmação original “Se  $q \equiv q'$ , então  $\delta(q, a) \equiv \delta(q', a)$ ” é também verdadeira.  $\square$

**Teorema 8** Seja  $A_{/\equiv} = \langle Q_{/\equiv}, \Sigma, \delta_{/\equiv}, [q_0], F_{/\equiv} \rangle$  o AFD quociente obtido a partir de um AFD  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ , com  $\delta_{/\equiv}([q], a) = [\delta(q, a)]$  para todo  $q \in Q$  e  $a \in \Sigma$ , tem-se que, para todo  $w \in \Sigma^*$  que  $\widehat{\delta_{/\equiv}}([q], w) = [\widehat{\delta}(q, w)]$ .

**Prova** Por indução sobre o tamanho de  $w \in \Sigma^*$  será mostrado a seguinte igualdade  $\widehat{\delta_{/\equiv}}([q], w) = [\widehat{\delta}(q, w)]$ .

**(B)**ase: Quando  $|w| = 0$  ( $w = \lambda$ ) tem-se trivialmente que  $\widehat{\delta_{/\equiv}}([q], \lambda) = [q] = [\widehat{\delta}(q, \lambda)]$ .

**(H)**ipótese indutiva: Suponha que para todo  $w \in \Sigma^*$  com  $|w| \geq 0$  tem-se que  $\widehat{\delta_{/\equiv}}([q], w) = [\widehat{\delta}(q, w)]$ .

**(P)**asso indutivo: Dado  $w = ua$  com  $u \in \Sigma^*$ ,  $|u| \geq 0$  e  $a \in \Sigma$  tem-se que,

$$\begin{aligned} \widehat{\delta_{/\equiv}}([q], w) &= \widehat{\delta_{/\equiv}}([q], ua) \\ &= \delta_{/\equiv}(\widehat{\delta_{/\equiv}}([q], u), a) \\ &\stackrel{(HI)}{=} \delta_{/\equiv}([\widehat{\delta}(q, u)], a) \\ &= [\delta(\widehat{\delta}(q, u), a)] \\ &= [\widehat{\delta}(q, ua)] \\ &= [\widehat{\delta}(q, w)] \end{aligned}$$

Agora por **(B)**, **(H)** e **(P)** pode-se efetivamente enunciar que para todo  $w \in \Sigma^*$  tem-se que  $\widehat{\delta_{/\equiv}}([q], w) = [\widehat{\delta}(q, w)]$ , o que completa a prova.  $\square$

**Teorema 9** Seja  $A_{/\equiv} = \langle Q_{/\equiv}, \Sigma, \delta_{/\equiv}, [q_0], F_{/\equiv} \rangle$  o AFD quociente obtido a partir de um AFD  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ , tem-se que  $q \in F$  se, e somente se,  $[q] \in F_{/\equiv}$

**Prova**  $(\Rightarrow)$  Trivial pela própria definição de  $F_{/\equiv}$ .  $(\Leftarrow)$  É suficiente mostrar que se  $q \equiv p$  e  $q \in F$ , então  $p \in F$ . Para provar isto, suponha que  $q \equiv p$  e  $q \in F$ , mas note que  $q \in F \iff \widehat{\delta}(q, \lambda) \in F$ , mas como  $q \equiv p$ , por definição tem-se para todo  $w \in \Sigma^*$  que  $\widehat{\delta}(q, w) \in F \iff \widehat{\delta}(p, w) \in F$ , assim no particular quando  $w = \lambda$  tem-se que  $\widehat{\delta}(p, \lambda) \in F$ , mas  $\widehat{\delta}(p, \lambda) = p$ , portanto,  $p \in F$ .  $\square$

**Teorema 10** Seja  $A_{/\equiv} = \langle Q_{/\equiv}, \Sigma, \delta_{/\equiv}, [q_0], F_{/\equiv} \rangle$  o AFD quociente obtido a partir de um AFD  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ . Então  $\mathcal{L}(A_{/\equiv}) = \mathcal{L}(A)$ .

**Prova** Basta notar que para qualquer  $w \in \Sigma^*$  tem-se que:

$$\begin{aligned} w \in \mathcal{L}(A_{/\equiv}) &\iff \widehat{\delta_{/\equiv}}([q_0], w) \in F_{/\equiv} \\ &\stackrel{\text{Teo. 8}}{\iff} [\widehat{\delta}(q_0, w)] \in F_{/\equiv} \\ &\stackrel{\text{Teo. 9}}{\iff} \widehat{\delta}(q_0, w) \in F \\ &\iff w \in \mathcal{L}(A) \end{aligned}$$

Portanto,  $\mathcal{L}(A_{/\equiv}) = \mathcal{L}(A)$ .  $\square$

O Teorema 10 é uma excelente justificativa para os algoritmos que iremos apresentar a seguir, pois ambos os algoritmos constroem exatamente o AFD  $A_{/\equiv}$  a partir de um AFD  $A$  dado como entrada. Para o funcionamento dos algoritmo de Moore em especial, é necessária definição de estados distinguíveis, tal definição (apresentada a seguir) é construída a partir da definição de equivalência entre estados.

**Definição 2.20** Seja  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  um AFD, dois estados  $q_i, q_j \in Q$  são ditos distinguíveis, sempre que  $q_i$  não for equivalente a  $q_j$ .

Note que pela Definição 2.20 dois estados  $q_i$  e  $q_j$  serão sempre distinguíveis em qualquer um dos casos a seguir:

- (1)  $q_i \in F$  e  $q_j \notin F$  ou
- (2)  $q_i \notin F$  e  $q_j \in F$  ou
- (3) Para algum  $w \in \Sigma^*$  acontece que  $\delta(q_i, w) \notin F$  e  $\delta(q_j, w) \in F$  ou  $\delta(q_i, w) \in F$  e  $\delta(q_j, w) \notin F$ .

Usando os casos (1) e (2), pode-se implementar um algoritmo auxiliar que será usando no Algoritmo de Moore, esse algoritmo auxiliar tem o objetivo de descobrir estados candidatos a serem equivalentes em um AFD, e obviamente este algoritmo roda em  $O(\#Q^2)$ .

---

**Algoritmo 3:** Algoritmo para gerar a matriz de candidatos a serem estados equivalentes.

---

**Entrada:** Um AFD  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$

**Saída:** Uma matriz  $M$  de candidatos a serem equivalentes

```

1 início
2   Defina uma matriz  $M$  de ordem  $\#Q \times \#Q$ 
3   para cada  $q_i \in Q$  faça
4     para cada  $q_j \in Q$  faça
5       se  $(q_i \in F \text{ e } q_j \notin F)$  ou  $(q_i \notin F \text{ e } q_j \in F)$  então
6         Marque a posição  $M_{i,j}$  com um  $\checkmark$ 
7       fim
8     fim
9   fim
10  retorna  $M$ 
11 fim

```

---

Notavelmente o Algoritmo 3 pode ser melhorado de diversas formas, como por exemplo utilizar uma lista de adjacências em vez de uma matriz, como este texto não é foca em algoritmos e estrutura de dados isso não serão feito aqui, então será simplesmente considerado que o algoritmo usa uma matriz, entretanto, para facilitar a exibição nos exemplos só serão apresentados os elementos da matriz abaixo da diagonal principal.

O Algoritmo 4 recebe a matriz de estados candidatos a serem equivalentes  $M$ , que foi gerada pelo Algoritmo 3 anterior e o AFD que gerou tal matriz, em seguida o algoritmo encontrar todos os pares de estados equivalentes (usando a terceira condição de estados distinguíveis)  $(q, p)$ , e marca esses pares de estados, todos os pares  $(q, p)$  não marcados ao final da execução do algoritmo serão então equivalentes.

---

**Algoritmo 4:** Algoritmo de minimização de Moore.

---

**Entrada:** Um AFD  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  e a matriz de candidatos equivalentes  $M$

**Saída:** A matriz de estados equivalentes  $M$

```

1 início
2   Defina uma matriz  $M$  de ordem  $\#Q \times \#Q$ 
3   para cada  $q_i \in Q$  faça
4     para cada  $q_j \in Q$  faça
5       se  $M_{i,j}$  não estiver marcado então
6         para cada  $a \in \Sigma$  faça
7           se  $\delta(q_i, a) \neq \delta(q_j, a)$  então
8             Marque a posição  $M_{i,j}$  com um  $\checkmark$ 
9           fim
10        fim
11      fim
12    fim
13  fim
14  retorna  $M$ 
15 fim

```

---

O Algoritmo de Moore encontra todos os estados equivalentes, agora basta construir o AFD  $A_{/\equiv} = \langle Q_{/\equiv}, \Sigma, \delta_{/\equiv}, [q_0], F_{/\equiv} \rangle$  usando as seguintes regras:

1. Para todo  $q \in Q$  a classe  $[q] = \{p \in Q \mid M_{q,p} \text{ não está marcada com } \checkmark\}$ .
2. Para todo  $[q] \in Q_{/\equiv}$  e  $a \in \Sigma$ , tem-se que  $\delta_{/\equiv}([q], a) = [\delta(q, a)]$ .

*Exemplo 2.5.1* Considerando o AFD descrito na Figura 2.14 a seguir, tal AFD reconhece a linguagem  $L = \{w \in \{0, 1\}^* \mid w = u11v, u, v \in \{0, 1\}^*\}$ .

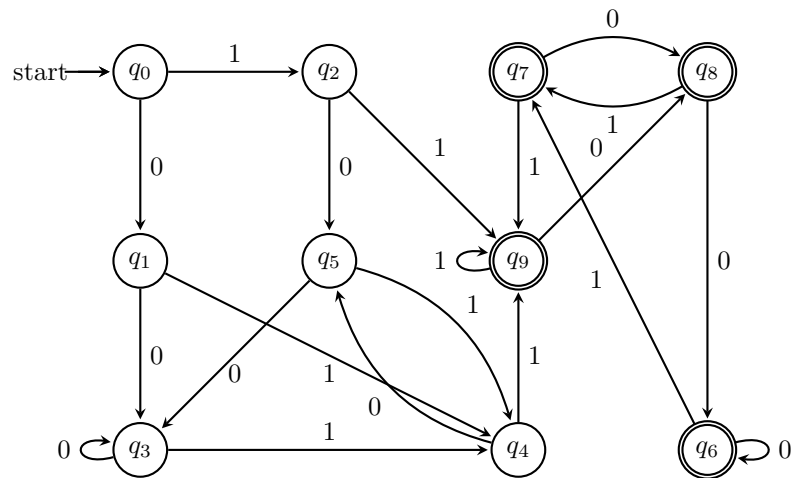


Figura 2.14: Um AFD.

Agora aplicando o autômato da Figura 2.14 ao Algoritmo 3 é obtida a matriz de candidatos a estados equivalentes descrita visualmente a seguir.

	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$	$q_7$	$q_8$
$q_1$									
$q_2$									
$q_3$									
$q_4$									
$q_5$									
$q_6$	✓	✓	✓	✓	✓	✓			
$q_7$	✓	✓	✓	✓	✓	✓			
$q_8$	✓	✓	✓	✓	✓	✓			
$q_9$	✓	✓	✓	✓	✓	✓			

Figura 2.15: Matriz de estados do AFD da Figura 2.14 candidatos a serem equivalentes.

Todas as posições  $(q_i, q_j)$  marcadas com ✓ informam que os estados  $q_i$  e  $q_j$  foram detectados não equivalentes pelo Algoritmo 3, em especial observe que os estados  $q_0$  e  $q_6$  de fato não podem ser equivalentes visto que  $q_0 \notin F$  e  $q_6 \in F$ , por outro lado, como os estados  $q_6, q_8 \in F$  eles são candidatos a serem equivalentes, então não existe uma marca na posição correspondente, assim como também não há uma marca na posição referente aos estados  $q_0$  e  $q_2$ . Passando o autômato e a matriz representada na Figura 2.15 para Algoritmo de Moore (Algoritmo 4) tem-se que a matriz será atualizada, ficando na forma abaixo.

	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$	$q_7$	$q_8$
$q_1$									
$q_2$	✓	✓							
$q_3$			✓						
$q_4$	✓	✓		✓					
$q_5$			✓		✓				
$q_6$	✓	✓	✓	✓	✓	✓			
$q_7$	✓	✓	✓	✓	✓	✓			
$q_8$	✓	✓	✓	✓	✓	✓			
$q_9$	✓	✓	✓	✓	✓	✓			

Figura 2.16: Matriz de estados equivalentes do AFD da Figura 2.14.

Os símbolos ✓ vermelhos foram introduzido na execução do Algoritmo de Moore, agora com a tabela atualizada basta verificar as colunas para identificar os estados equivalentes, e assim construir as classes de equivalência, todos os estados (linhas) não marcado em um coluna  $q_i$  são equivalentes ao estado  $q_i$  assim pela matriz representada na Figura 2.16 tem-se que,

$$\begin{aligned} [q_0] &= \{q_0, q_1, q_3, q_5\} \\ [q_2] &= \{q_2, q_4\} \\ [q_6] &= \{q_6, q_7, q_8, q_9\} \end{aligned}$$

e seguindo as regras para gerar a função de transição,

$$\delta([q_0], 1) = [q_2]$$

$$\delta([q_0], 0) = [q_0]$$

$$\delta([q_2], 1) = [q_6]$$

$$\delta([q_2], 0) = [q_0]$$

$$\delta([q_6], 1) = [q_6]$$

$$\delta([q_6], 0) = [q_6]$$

e  $F_{/\equiv} = \{[q_6]\}$ , assim tem-se AFD mínimo descrito na Figura 2.17.

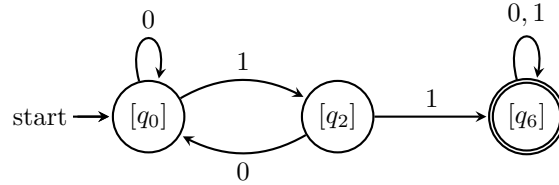


Figura 2.17: Um AFD.

Exemplo 2.5.2

Considere o AFD representado pela Figura 2.18 a seguir.

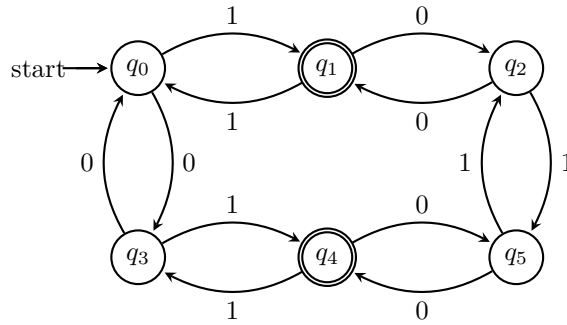


Figura 2.18: Um AFD.

Usando o Algoritmo 3 é obtida a seguinte matriz de candidatos a estados equivalentes,

	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$
$q_1$	✓				
$q_2$			✓		
$q_3$				✓	
$q_4$	✓			✓	✓
$q_5$			✓		✓

Figura 2.19: Matriz de estados do AFD da Figura 2.18 candidatos a serem equivalentes.

Aplicando a matriz representada pela Figura 2.19 ao algoritmo de Moore tal matriz será atualizada para a forma a seguir.

	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$
$q_1$	✓				
$q_2$	✓	✓			
$q_3$		✓	✓		
$q_4$	✓		✓	✓	
$q_5$	✓	✓		✓	✓

Figura 2.20: Matriz de estados equivalentes do AFD da Figura 2.18.

Agora usando as regras de construção do AFD quociente após a aplicação do algoritmo de Moore ficamos com as seguintes classes de equivalência.

$$[q_0] = \{q_0, q_3\}$$

$$[q_1] = \{q_1, q_4\}$$

$$[q_2] = \{q_2, q_5\}$$

e seguindo as regras para gerar a função de transição tem-se,

$$\delta([q_0], 1) = [q_1]$$

$$\delta([q_0], 0) = [q_0]$$

$$\delta([q_1], 1) = [q_0]$$

$$\delta([q_1], 0) = [q_2]$$

$$\delta([q_2], 1) = [q_2]$$

$$\delta([q_2], 0) = [q_1]$$

Logo o AFD quociente e da forma descrita pela Figura 2.21 a seguir.

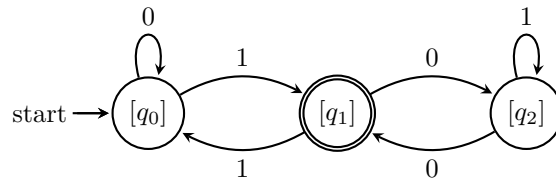


Figura 2.21: Um AFD.

Agora será apresentado o algoritmo de Hopcroft [16] para minimizar um AFD dado, a ideia de Hopcroft é realizar o refinamento progressivo das partições do conjunto de estados, usando a ideia de blocos<sup>6</sup>. O conceito é quebrar o conjunto de estados em subconjuntos contendo apenas estados equivalentes.

<sup>6</sup> Que em última análise serão classes de equivalência

**Definição 2.21** Seja  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  um AFD, um bloco é qualquer  $X \subseteq Q$  tal que todos os  $q_i \in X$  são do mesmo tipo<sup>a</sup>.

<sup>a</sup>Tipo aqui corresponde a mesma ideia do algoritmo de Moore, ou seja, o tipo de estados finais e não-finais.

## 2.6 Questionário

**Questão 2.1** Para cada linguagem a seguir construa um AFD.

- $L_1 = \{w \in \{a, b\}^* \mid w \text{ termina em } ab\}$ .
- $L_2 = \{w \in \{0, x, 1\}^* \mid w \text{ começa com } 1 \text{ mas não contém a subpalavra } 0xx1\}$ .
- $L_3 = \{w \in \{1, 2, 0\}^* \mid w \text{ contém a subpalavra } 1201\}$ .

- (d).  $L_4 = \{w \in \{a, b\}^* \mid w = a^n \text{ com } n \in \mathbb{N} \text{ ou se existe } b \text{ em } w, \text{ ele é seguido de } aa\}$ .
- (e).  $L_5 = \{w \in \{0, 1\}^* \mid w \text{ não possui nem uma subpalavra } 0^{2n} \text{ com } n \in \mathbb{N} - \{0\}\}$ .
- (f).  $L_6 = \{w \in \{a, b, c\}^* \mid w \in \{b, c\}^* \text{ ou todo } a \text{ em } w \text{ aparece entre } b \text{ e } c\}$ .

**Questão 2.2** Para cada linguagem a seguir construa um AFD, AFN ou  $\lambda$ -AFN que reconhecer a linguagem. Considerando que  $N_a(w)$  corresponde ao número de  $a \in \Sigma$  na palavra  $w$ .

- (a).  $L_1 = \{w \in \{\circ, \star, 0, 1\}^* \mid N_\star(w) > 3 \wedge N_\circ(w) < 5\}$ .
- (b).  $L_2 = \{w \in \{0, x, 1\}^* \mid wN_x(w) = 6\}$ .
- (c).  $L_3 = \{u0^i1^jv \in \{1, 2, 0\}^* \mid u, v \in \{2\}^* \wedge i + j = 2k + 1, k \in \mathbb{N} - \{0\}\}$ .
- (d).  $L_4 = \{ubv \mid u, v \in \{a, b, c\}^*, N_b(u) = 0 \wedge N_b(v) < 5\}$ .

**Questão 2.3** Considere que  $\Sigma = \{0, 1\}$ , para cada linguagem a seguir construa um AFD, AFN ou  $\lambda$ -AFN que reconhecer a linguagem descrita.

- (a). A linguagem de todas as palavras começando com 0 e terminando com 1 e que não apareça a sub-palavra 01010.
- (b). A linguagem de todas as palavras começando com 10 e que não possuem a sub-palavra 001.
- (c). A linguagem de todas as palavras terminando com  $(11)^2$  ou  $(11)^4$ .
- (d). A linguagem de todas as palavras que possuem as sub-palavras 010 ou 101.

**Questão 2.4** Considere que  $\Sigma = \{a, b, c, d\}$  construa um AFD que reconhece a linguagem de todos os  $w \in \Sigma^*$  tal que  $w$  satisfaz as seguintes restrições.

- (i)  $w$  nunca deve conter a subpalavra  $abcd$ ;
- (ii) Sempre que  $w$  iniciar em  $a$  obrigatoriamente deve terminar  $cd$ ;
- (iii) Se  $w$  iniciar com  $b$  então não deve conter a subpalavra  $bc$ ;
- (iv)  $w$  sempre contém a subpalavra  $ac$ .

**Questão 2.5** Considere que  $\Sigma = \{0, 1\}$  construa um AFD que reconhece a linguagem de todos os  $w \in \Sigma^*$  tal que  $w$  satisfaz as seguintes restrições.

- (i) Sempre que  $w$  iniciar com 00 tem-se que  $|w| < 10$ ;
- (ii) Sempre que  $w$  iniciar em 10 obrigatoriamente deve terminar 00;
- (iii) Se  $w$  iniciar com 01 não deve conter a subpalavra 101.

**Questão 2.6** Considere que  $\Sigma = \{0, 1\}$  construa um AFD que reconhece a linguagem de todos os  $w \in \Sigma^*$  tal que  $w$  satisfaz as seguintes restrições.

- (i)  $w$  nunca inicia com 00 ou 11;
- (ii) Sempre que  $w$  possuir a subpalavra 101 deve possuir a subpalavra 010 e
- (iii) Em  $w$  a subpalavra 101 deve sempre aparece antes da subpalavra 010;



**Questão 2.7** Considere que  $\Sigma = \{a, c, t, g\}$  construa um AFD, AFN ou  $\lambda$ -AFN que reconhece a linguagem de todos os  $w \in \Sigma^*$  tal que  $w$  satisfaz as seguintes restrições.

- (i) Quando  $w$  iniciar com  $ag$  deve conter pelo menos duas subpalavra  $tg$ ;
- (ii) Quando  $w$  possuir terminar em  $tt$  deve possuir a subpalavra  $act$  e
- (iii) Em  $w$  a subpalavra  $tc$  nunca acontece e
- (v)  $|w| \geq 4$ .

**Questão 2.8** Seja  $M$  seu autômato construído para a Questão 2.6, construa um AFD  $M'$  tal que para todo  $w \in \Sigma^*$  tem-se que  $w \in \mathcal{L}(M)$  se, e somente se,  $w^r \in \mathcal{L}(M')$ . Ou seja, construa um AFD que reconhece a linguagem reversa do seu autômato da Questão 2.6.

**Questão 2.9** Seja  $M$  seu autômato construído para a Questão 2.7, construa um AFD  $M'$  tal que para todo  $w \in \Sigma^*$  tem-se que  $w \in \mathcal{L}(M)$  se, e somente se,  $w^r \in \mathcal{L}(M')$ . Ou seja, construa um AFD que reconhece a linguagem reversa do seu autômato da Questão 2.7.

**Questão 2.10** Considerando as linguagens  $L_1, L_2, L_3, L_4, L_5$  e  $L_6$  descritas na Questão 2.1 e considerando as operações usuais de união ( $\cup$ ), interseção ( $\cap$ ) e complemento ( $-$ ), construa um AFD para as linguagens descritas abaixo.

- (a).  $L_2 \cup L_3$ .
- (b).  $L_1 \cap L_4$ .
- (c).  $\overline{L_5}$ .
- (d).  $\overline{L_3 \cap L_5}$
- (e).  $\overline{\overline{L_6} \cap L_1}$

**Questão 2.11** Seja  $L \subseteq \Sigma$  para um alfabeto  $\Sigma$  qualquer com  $\# \in \Sigma$ , demonstre que todas as linguagens a seguir são regulares.

- (a).  $L_1 = \{\#^{|w|} \in \Sigma^* \mid w \in L\}$ .
- (b).  $L_2 = \{d_{\#}(w) \in \Sigma^* \mid w \in L\}$ , onde  $d_{\#}(w)$  corresponde a uma palavra  $w' \in \Sigma^*$ , de forma que  $w'$  é a forma reduzida de  $w$  após remover todos os  $\#$ .
- (c).  $L_3 = \{x \in \Sigma^* \mid x, y \in L \wedge y \in \Sigma^*\}$ .
- (d).  $L_4 = \{y \in \Sigma^* \mid x, y \in L \wedge x \in \Sigma^*\}$ .
- (e).  $L_5 = \{x \in \Sigma^* \mid x, y \in L \wedge x \in \Sigma^*\}$ .
- (f).  $L_6 = \{xy \in \Sigma^* \mid x, y \in \Sigma^* \wedge yx \in L\}$ .
- (g).  $L_7 = \{w \in \Sigma^* \mid www \in L\}$ .
- (h).  $L_8 = \{x \in L \mid xy \in L \iff y = \lambda\}$ .
- (i).  $L_9 = \{xy \in L \mid y \in L \iff x = \lambda\}$ .



# Linguagens Regulares: Modelos Alternativos

*“As far as I am aware this  
pronunciation is incorrect in all known  
languages. ”*

*Kenneth Kleene, falando sobre a  
pronuncia do último no de seu Pai,  
Stephen Kleene.*

No Capítulo 2 foi apresentado a classe das linguagens regulares  $\mathcal{L}_{Reg}$  através da formalização dos autômatos finitos (em diversas classes distintas), entretanto, essa não é a única forma de apresentar e formalizar a classe de linguagens  $\mathcal{L}_{Reg}$ . Neste capítulo serão apresentados dois modelos alternativos para tal classe de linguagens, sendo estes modelos, o modelo denotacional conhecido por expressões regulares [25], e o modelo de gramática geradora [10].

## 3.1 Gramática Regulares

Nesta seção será apresentado um terceiro formalismo para as linguagens regulares, sendo este um formalismo gerador (ou axiomático) [25].

**Definição 3.1** Uma gramática formal  $G = \langle V, \Sigma, S, P \rangle$  é dita Linear à Direta (à Esquerda), ou simplesmente GLD (GLE), se todas as suas produções são da forma,  $A \rightarrow wB$  ( $A \rightarrow Bw$ ), onde  $A \in V, B \in V \cup \{\lambda\}$  e  $w \in \Sigma^*$ .

*Exemplo 3.1.1* A gramática  $G_1 = \langle \{B, S, A\}, \{a, b\}, S, P \rangle$  onde  $P$  é formado pelas regras:

$$\begin{aligned} S &\rightarrow aaB \\ B &\rightarrow bb \end{aligned}$$

é uma GLD.

*Exemplo 3.1.2* A gramática  $G_1 = \langle \{X, S, Y\}, \{0, 1\}, S, P \rangle$  onde  $P$  é formado pelas regras:

$$\begin{aligned} S &\rightarrow X001 \\ S &\rightarrow Y011 \\ X &\rightarrow S01 \\ Y &\rightarrow \lambda \end{aligned}$$

é uma GLE.

Em uma gramática formal  $G$  quando para as sentenças  $w, w_1, \dots, w_n \in (V \cup \Sigma)^*$ , existem para todo  $1 \leq i \leq n$  a regras  $w \rightarrow w_i \in P$ , é comum para simplificar a escrita do conjunto de regras usar a notação  $w \rightarrow w_1 \mid \dots \mid w_n$ , em vez de escrever cada regra em separado.

**Exemplo 3.1.3** Considere a GLE apresentada no Exemplo 3.1.2 o conjunto  $P$  da mesma poderia ser escrito como:

$$\begin{aligned} S &\rightarrow X001 \mid Y011 \\ X &\rightarrow S01 \\ Y &\rightarrow \lambda \end{aligned}$$

Um tipo mais rigoroso de gramática, são as chamadas gramática regulares, que como comentado em [19], podem ser vista como gramáticas lineares em que as regras são capazes de produzir no máximo único símbolo terminal a cada derivação.

**Definição 3.2** (Gramáticas Regulares) Uma gramática regular  $G$  se ela é linear à esquerda (ou à direita) e toda produção é da forma  $A \rightarrow Bw$  ( $A \rightarrow wB$ ) com  $A \in V, B \in V \cup \{\lambda\}$  e  $w \in \Sigma \cup \{\lambda\}$ .

**Exemplo 3.1.4** A estrutura  $G = \langle \{A, B, S\}, \{0, 1\}, S, P \rangle$  com  $P$  formado pelas regras:

$$\begin{aligned} S &\rightarrow 0A \mid 1B \mid \lambda \\ A &\rightarrow 1S \\ B &\rightarrow 0S \end{aligned}$$

é uma gramática regular.

De forma natural duas gramática  $G_1$  e  $G_2$  serão ditas equivalentes sempre que elas gerarem a mesma linguagens (volte a Definição 1.19 se necessário for para relembrar). O próximo resultado estabelece que gramática lineares (em um única direção) e gramática regulares tem o mesmo poder de geração de linguagens.

**Teorema 11**  $L = \mathcal{L}(G)$  para alguma gramática linear (esquerda ou direita)  $G$  se, e somente se, existe uma gramática regular  $G'$  na mesma direção (esquerda ou direita) tal que  $L = \mathcal{L}(G')$ .

**Prova**  $(\Rightarrow)$  Suponha que  $L = \mathcal{L}(G)$  para alguma gramática  $G = \langle V, \Sigma, S, P \rangle$  tal que  $G$  seja linear à esquerda (a prova é similar para o caso à direita). Agora construa uma nova gramática  $G' = \langle V', \Sigma, S, P' \rangle$  tal que  $P'$  é definido usando as seguintes regras:

R1: Se  $A \rightarrow Bw \in P$  onde  $B \in V \cup \{\lambda\}, |w| \leq 1$ , então  $A \rightarrow Bw \in P'$ .

R2: Se  $A \rightarrow Ca_1 \dots a_n \in P$  onde  $B \in V$  e  $a_i \in \Sigma$  sendo  $1 \leq i \leq n$  e  $n > 1$ , então tem-se que  $A \rightarrow B_n a_n, B_n \rightarrow B_{n-1} a_{n-1}, \dots, B_2 \rightarrow C a_1 \in P'$ .

R3: Se  $A \rightarrow a_1 \dots a_n \in P$  onde  $a_i \in \Sigma$  sendo  $1 \leq i \leq n$  e  $n \geq 2$ , então tem-se que  $A \rightarrow B_n a_n, B_n \rightarrow B_{n-1} a_{n-1}, \dots, B_2 \rightarrow B_1 a_1, B_1 \rightarrow \lambda \in P'$ .

Para as regras R2 e R3 todo  $B_i$  é uma nova variável existente em  $V'$  que não existe originalmente em  $V$ . Claramente a gramática  $G'$  é regular unitária à esquerda. Também não é difícil mostra por indução sobre o tamanho das derivações que para todo  $w \in \Sigma^*$  tem-se que  $S \vdash_G^* w$  se, e somente se,  $S \vdash_{G'}^* w$ , portanto,  $\mathcal{L}(G) = \mathcal{L}(G')$ .  $(\Leftarrow)$  Trivial uma vez que toda gramática regular à esquerda (ou à direita) é um caso particular de gramática linear à esquerda (ou à direita).  $\square$

# Referências Bibliográficas

---

- [1] J. M. Abe and N. Papavero. *Teoria Intuitiva dos Conjuntos*. MAKRON Books, 1991.
- [2] A. V. AHO, M. S. LAM, R. SETHI, and J. D. ULLMAN. *Compiladores: Princípios, Técnicas e ferramentas*. Editora Pearson, 2 edition, 2007.
- [3] J. Avigad. Handbook of proof theory. In *Studies in Logic and the Foundations of Mathematics*, ch. CiteSeer, 1998.
- [4] M. Ayala-Rincón and F. L. C. de Moura. *Fundamentos da Programação Lógica e Funcional – O princípio de Resolução e a Teoria de Reescrita*. Editora UnB, 2014.
- [5] B. Bedregal.  $\lambda$ -ALN: Autômatos Lineares Não-determinísticos com  $\lambda$ -Transições. *TEMA - Tendências em Matemática Aplicada e Computacional*, 12(3):171–182, 2011.
- [6] B. Bedregal. Nondeterministic Linear Automata and a Class of Deterministic Linear Languages. *Preliminary Proceedings LSFA*, pages 183–196, 2015.
- [7] B. Bedregal and B. M. Acióly. Introdução à lógica clássica para a ciência da computação. Notas de Aula, 2007.
- [8] B. Bedregal, B. M. Acióly, and A. Lyra. *Introdução à Teoria da Computação: Linguagens Formais, Autômatos e Computabilidade*. Editora UnP, Natal, 2010.
- [9] S. R. Buss. *Handbook of proof theory*. Elsevier, 1998.
- [10] N. Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.
- [11] K. Cooper and L. Torczon. *Construindo Compiladores*, volume 1. Elsevier Brasil, 2017.
- [12] V. S. Costa. Linguagens Lineares Fuzzy. Master’s thesis, Programa de Pós-graduação em Sistemas e Computação, Universidade Federal do Rio Grande do Norte, UFRN, Natal, RN, 2016.
- [13] V. S. Costa. *Autômatos Fuzzy Hesitantes Típicos: Teoria e Aplicações*. PhD thesis, Programa de Pós-graduação em Sistemas e Computação, Universidade Federal do Rio Grande do Norte, UFRN, Natal, RN, 2020.
- [14] C. De la Higuera. *Grammatical inference: Learning Automata and Grammars*. Cambridge University Press, London, 2010.
- [15] J. Erickson. *Models of computation*. <http://jeffe.cs.illinois.edu/teaching/algorithms/>, 2014. Não publicado.
- [16] J. Hopcroft. An  $n \log n$  algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pages 189–196. Elsevier, 1971.

- [17] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Pearson Education India, USA, 31 edition, 2008.
- [18] S. C. Kleene. Representation of events in nerve nets and finite automata. Technical report, Rand Project Air Force Santa Monica CA, 1951.
- [19] P. Linz. *An Introduction to Formal Languages and Automata*. Jones & Bartlett Learning, New York, 2006.
- [20] S. Lipschutz. *Teoria dos Conjuntos*. McGraw-Hill do Brasil - LTDA/MEC, 1978.
- [21] S. Lipschutz and M. Lipson. *Matemática Discreta*. Bookman Editora, 2013. Coleção Schaum.
- [22] J. P. Martins. *Lógica e Raciocínio*. College Publications, 2014.
- [23] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.
- [24] G. H. Mealy. A method for synthesizing sequential circuits. *The Bell System Technical Journal*, 34(5):1045–1079, 1955.
- [25] P. B. Menezes. *Linguagens Formais e Autômatos*. Sagra-Dcluzzato, 1998.
- [26] E. F. Moore. Gedanken-experiments on sequential machines. *Automata Studies*, 34:129–153, 1956.
- [27] J. Myhill. Creative sets. *Journal of Symbolic Logic*, 22(1), 1957.
- [28] A. Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.
- [29] M. O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.
- [30] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.
- [31] H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953.
- [32] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.