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

15 de setembro de 2025

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 LaTeX (em sua versão 2) e compilado usado as ferramentas LuaLaTeX e BibTeX, tais ferramentas fornecidas pelas distribuições TeXLive e MacTeX, respectivamente nos sistema 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 15 de setembro de 2025 (782 minutos após a meia-noite).

Sumário

Parte I

Fundamentos

Capítulo 1

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 alfabeto em questão. As palavras de uma linguagem formal, em alguns cenários são também pode ser chamadas de termos, ou ainda, de fórmulas [23].

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[1] 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, 12]), essa estrutura externa funciona como um universo avaliativo, no qual as palavras da linguagem pode ser interpretadas (ter seu significado exposto).

[1] 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 [34], ou seja, uma visão de computabilidade por máquinas de computação, os chamados autômatos finitos [2, 18]. Além desse objetivo, essa parte do documento também irá forma uma pequena teoria para as linguagens formais usando modelos de representação, computação e geração, para a 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.

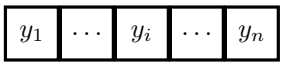
1.2 Sobre Autômatos Finitos

Como dito em [13, 14] 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:

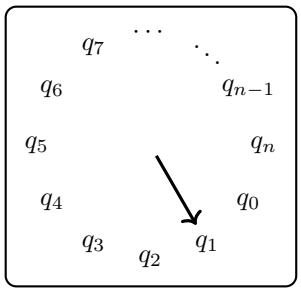
- Um conjunto finito de memórias[2], estas sendo subdivididas em células, cada uma das quais capaz de comportar um único símbolo por vez.
- Uma unidade de controle[3] que administra o estado atual do autômato e é responsável por executar as instruções (programa) da máquina.

[2] Na literatura também é usado o termo fita em vez de memória [26].

[3] Também chamada de unidade de processamento (UCP).



Memória do autômato



Unidade de Controle

Figura 1.1: Representação informal do conceito de autômato finito com uma única memória retirado de [14].

Com respeito as memórias é comum assumir a existência de um **dispositivo de leitura e (ou) escrita**[1] 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 [13, 14], 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 [14], 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 [24], Kleene [19], Mealy [25], Moore [27], Rabin e Scott [30, 31].

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) [13] Qualquer conjunto finito e não vazio Σ 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 Σ.

[1] Também é usado a nomenclatura cabeçote [13, 14].

Exemplo 1.3.1 Os conjuntos $\{0,1,2,3\}, \{a,b,c\}, \{\heartsuit, \spadesuit, \diamond, \clubsuit\}$ 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 Σ.

Definição 1.2 (Comprimento das palavras) Seja w uma palavra qualquer sobre um certo alfabeto Σ, o comprimento[a] de w, denotado por $|w|$, corresponde ao número de símbolos existentes em w.

[a] 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, 18, 20], 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 (Associativade 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. □

Sobre qualquer alfabeto Σ sempre é definida uma palavra especial chamada **palavra vazia** [18, 20], essa palavra especial não possui nenhum símbolo, e em geral é usado o símbolo λ para denotar a palavra vazia [8, 13]. Como mencionado em [8, 14] sobre a palavra vazia é importante destacar que:

$$w\lambda = \lambda w = w \tag{1.1}$$
$$|\lambda| = 0 \tag{1.2}$$

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 Σ a potência de w é definida recursivamente para todo $n \in \mathbb{N}$ como sendo:

$$w^0 = \lambda \tag{1.3}$$
$$w^{n+1} = ww^n \tag{1.4}$$

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) $w_1^3 = w_1w_1^2 = w_1w_1w_1^1 = w_1w_1w_1w_1^0 = w_1w_1w_1\lambda = ababab.$

(b) $w_2^2 = w_2w_2^1 = w_2w_2w_2^0 = w_2w_2\lambda = w_2w_2 = bacbac.$

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) $w^mw^n = w^{m+n}$.

Prova Direto das Definições 1.3 e 1.4, e portanto, ficará como exerício ao leitor. □

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) [13] 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 Σ um alfabeto a potência de Σ é definida recursivamente para todo $n \in \mathbb{N}$ como:

$$\Sigma^0 = \{\lambda\} \tag{1.5}$$
$$\Sigma^{n+1} = \{aw \mid a \in \Sigma, w \in \Sigma^n\} \tag{1.6}$$

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 Σ^n é sempre finito, uma vez que, o conjunto Σ^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 Σ um alfabeto o fecho positivo e o fecho de Kleene de Σ, denotados respectivamente por Σ^+ e Σ^*, correspondem aos conjuntos: $\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i$ 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 [14] o mesmo corresponde ao monoide livremente gerado pelo conjunto Σ 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 qu e 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:

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

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

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

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) Analoga ao item anterior.

(iii) Trivial, pois basta notar que $\lambda, w \in (PRE(w) \cap SUF(w))$, e portanto, tem-se claramente que $\#(PRE(w) \cap SUF(w)) > 1$. □

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

Prova | Direto do item (iii) do Teorema 2. □

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

Definição 1.10 | (Linguagem) Dado um alfabeto Σ, 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 Σ.

(a) Σ^*.

(b) $\{0^n b^n \mid n \in \mathbb{N}\}$.

(c) $\{\lambda, 0, 1\}$.

(d) Σ^{22}.

(e) $\emptyset$.

Similarmente ao que ocorre com os alfabetos, as linguagens por serem conjuntos "herdam" as operações básicas da teoria dos conjuntos [21, 22, 1], isto é, estão definidas sobre as linguagens as operações de união, interseção, completo, 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:

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

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) $L_3 L1 = \{w \mid w = a^p \text{ ou } w = a^{p+1}b \text{ ou } a^p bba \text{ onde } p \text{ é um número primo}\}$.

(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:

$$L^0 = \{\lambda\} \tag{1.10}$$
$$L^{n+1} = LL^n \tag{1.11}$$

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.

$$L^+ = \bigcup_{i=1}^{\infty} L^i \tag{1.12}$$
$$L^* = \bigcup_{i=0}^{\infty} L^i \tag{1.13}$$

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 [34]. 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, 20, 26], assim sendo, aqui serão apresentadas as linguagens formais da mais simples para a mais complexas seguindo a hierarquia de Chomsky [11], 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 $V \cap \Sigma = \emptyset$, Σ é um alfabeto, $S \in V$ é uma variável destacada chamada de **variável inicial** e P é um conjunto de regras de reescrita[a] da forma $w \to w'$ onde

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

[a] Também é comum encontrar na literatura a nomenclatura regras de produção [8, 20].

Atenção Na escrita do conjunto P sempre que $w \to w_1$ e $w \to w_2$ com $w_1 \neq w_2$, é escrito simplesmente $w \to w_1 \mid w_2$, 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 $A \to aABa \mid B$ e $B \to \lambda$ 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 $w \to w' \in P$.

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 $A \to aABa$.

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 $S \to 11A, A \to B0$ e $B \to 000$ 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 sequencia de derivações $w_1 \vdash w_2 \vdash \cdots \vdash w_n$ acontecer, as palavras $w_1, w_2, \cdots, w_n$ 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 Σ 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 $|w| = max\Big(\{|w'| \mid w' \in L^3\}\Big)$.

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). $L^mL^n = L^{m+n}$.

(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 demostre 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[a], para que a igualdade:

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

seja verdadeira.

[a] Trivial aqui diz respeito a uma linguagem que não seja o próprio alfabeto ou o conjunto $\{\lambda\}$.

Parte II

Linguagens Regulares

Capítulo 2

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, 18, 20, 26], 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, Σ é um alfabeto, $\delta : Q \times \Sigma \to Q$ 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[a].

[a] Em algumas referências também é usado o termo conjunto de estados de aceitação [15].

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, δ 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, 18, 20].

Embora o estudo formal dos autômatos finitos tenha começado muito antes, sua formulação moderna foi estabelecida pelo artigo[1] do ano de 1959 escrito por Michael Rabin e Dana Scott [31]. Em tal artigo Rabin e Scott chamaram o conjunto de estados

[1] Esse artigo levou Michael Rabin e Dana Scoot 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 $q_1, q_2, \cdots, q_R$ 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 preocupa 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 usado tanto para designar o singular quanto o plural, ficando a distinção a critério das sentenças envolvendo tais singlas.

A função de transição (δ) pode ser interpretada semanticamente como sendo o programa que o autômato executa, assim uma aplicação qualquer de δ 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 [14]. 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.

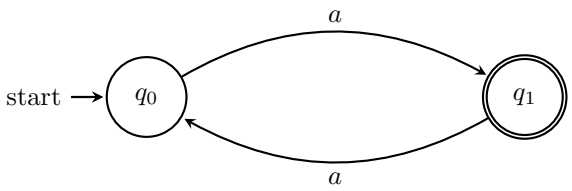


Figura 2.1: Representação visual do AFD no Exemplo 2.1.1.

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.

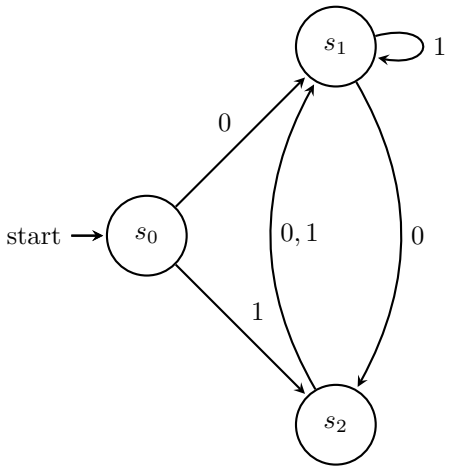


Figura 2.2: Representação visual do AFD S do Exemplo 2.1.4.

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 δ é estendida para uma função $\widehat{\delta} : Q \times \Sigma^* \to Q$ usando recursividade como se segue.

$$\widehat{\delta}(q, \lambda) = q \tag{2.1}$$
$$\widehat{\delta}(q, wa) = \delta(\widehat{\delta}(q, w), a) \tag{2.2}$$

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 $\widehat{\delta}(q_0, w) \in F$ e é rejeitada por A em qualquer outro caso.

É fácil perceber que $\widehat{\delta}(q_0, w) \in F$ 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,

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

com $q_n \in F$, sendo uma sequencia 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 $\widehat{\delta}(q_0, \lambda) \in F$ 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:

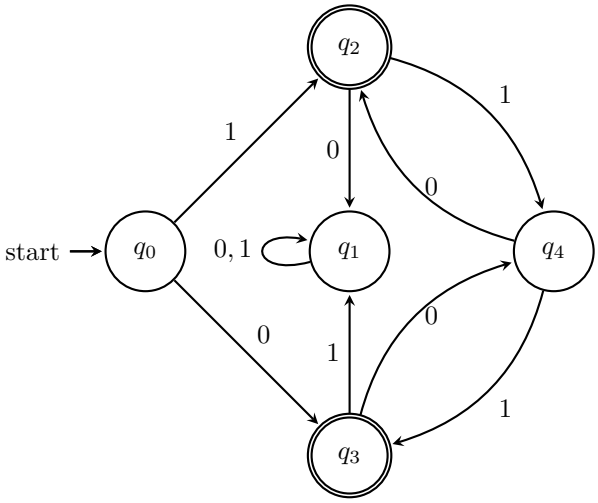


Figura 2.3: Um AFD com dois estados finais.

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:

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

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.

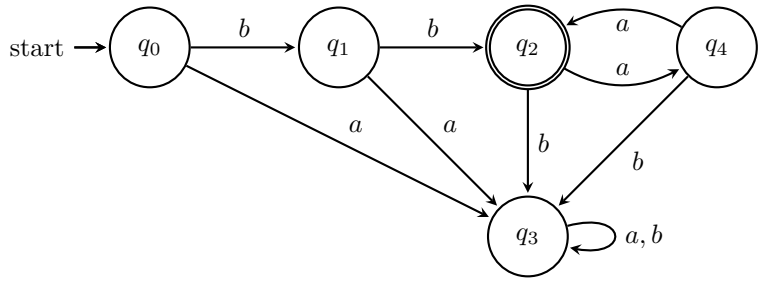


Figura 2.4: AFD A_1 que reconhece a linguagem $\{bba^{2n} \mid n \in \mathbb{N}\}$.

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,

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

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 $\widehat{\delta}(q_0, bba^{2n}) \in F$, ou seja, $\widehat{\delta}(q_0, bba^{2n}) = q_2$.

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

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

Logo, por **(B)**, **(H)** e **(P)** tem-se que $\widehat{\delta}(q_0, bba^{2n}) \in \mathcal{L}(A_1)$ 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 $\widehat{\delta}(q_0, w) = q_2$, entretanto, pela definição de δ (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 $\widehat{\delta}(q_0, w) = q_2$, então $w = bba^{2n}$ com $n \in \mathbb{N}$, e portanto, $w \in L$, completando assim a prova. □

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,

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

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. □

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 [19], 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 [20], 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 [14]. 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, Σ, q_0 e F são da mesma forma que na Definição 2.1, já $\delta_N : Q \times \Sigma \to \wp(Q)$ 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 δ é descrita pela Tabela 2.1 a seguir é um AFN.

Q \ Σ	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 δ_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.

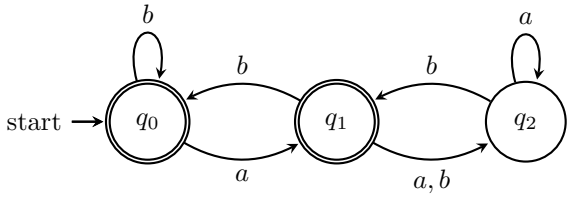


Figura 2.5: Grafo de transição do AFN do Exemplo 2.2.1.

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 δ_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 $\delta_N : Q \times \Sigma^* \to \wp(Q)$ definida pela seguinte recursão.

$$\widehat{\delta_N}(q, \lambda) = \{q\} \tag{2.4}$$

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

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,

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

mas tem-se que,

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

e

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

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

$$\widehat{\delta_N}(q_0, ab) = \{q_0, q_2\} \qquad (2.9)$$

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

$$\widehat{\delta_N}(q_0, aba) = \{q_0, q_1\} \qquad (2.10)$$

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[1], 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.

[1] Como explicado em [14] 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.

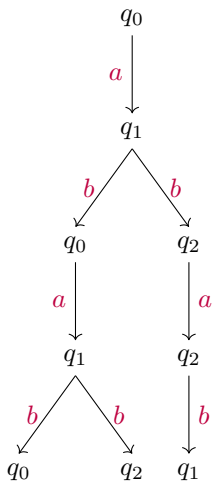


Figura 2.6: Árvore de computação da palavra "*abab*" no AFN da Figura 2.5.

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 $\widehat{\delta}(q_0, w) \cap F \neq \emptyset$ 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 "$aabbbba$" 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 "$aabbbba$" é aceita por tal AFN. Por outro lado, a palavra "$aabb$" não é aceita pelo AFN.

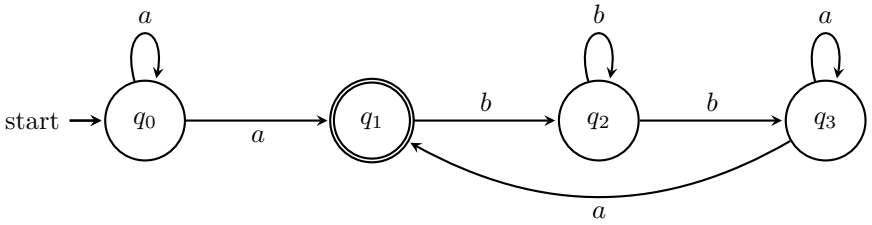


Figura 2.7: Grafo de transição de um AFN.

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:

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

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.

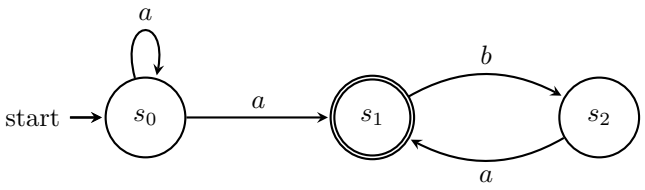


Figura 2.8: Grafo de transição de um AFN.

Prova (⇒) 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 δ_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:

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

agora fazendo,

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

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

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

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)ipótese 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,

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

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[a] 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 $\widehat{\delta_N}(s_2, a) = \{s_1\}$ 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, $w \in L$. □

[a] Fica para o leitor a tarefa de provar que para todo $m \geq 1$ tem-se que $s_1 \in \widehat{\delta_N}(s_0, a^m)$.

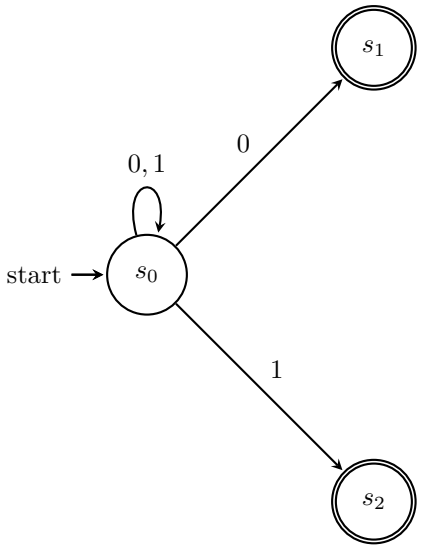


Figura 2.9: Grafo de transição de um AFN S.

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 δ_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[2]. 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

[2] 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 foma uv com $u \in \{0,1\}^*$ e $v \in \{0,1\}$, consequentemente $w \in L$. □

De forma ingênua o leitor pode vim 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, 20] 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. □

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 [18, 26].

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

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

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,

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

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. □

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.

Corolario 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. □

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 [31], tem a característica de poder vim a produzir durante sua execução alguns estados inacessíveis[3] no AFD resultante.

[3] Um estado q em um AFD é dito inacessível se não existe um $w \in \Sigma^*$ tal que $\widehat{\delta}(q_0, w) = q$. Vale também ressaltar como destaco em [8, 18] que estados inacessíveis não aumentam o poder de computação nos AFD.

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 interativamente um conjunto de estados $Q' \subseteq \wp(Q)$, que no pior caso tem-se que $Q' = \wp(Q)$.

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

início
  Inicialize os conjuntos Q_u e Q' com um estado rotulado por $\{q_0\}$
  Inicialize o conjunto F' como sendo vazio
  repita
    Selecione um estado $X \in Q_u$
    para cada $a \in \Sigma$ **faça**
      Determine o conjunto $Y = \bigcup_{q \in X} \delta_N(q, a)$
      se $Y \notin Q'$ **então**
        Adicione o estado rotulado por Y em Q' e em Q_u
      fim
      Defina a transição $\delta(X, a) = Y$
    fim
    Remova X de Q_u
  até $Q_u = \emptyset$;
  para cada $X \in Q'$ **faça**
    se $X \cap F \neq \emptyset$ **então**
      Adicione X ao conjunto F'
    fim
  fim
  retorna $A' = \langle Q', \Sigma, \delta, \{q_0\}, F'\rangle$
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 equivalem 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 δ é como se segue.

Q \ Σ	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 λ-Autômatos Finitos Não-determinísticos

Os λ-Autômatos Finitos Não-determinísticos, ou simplesmente, λ-AFN são como dito em [26], 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 λ-transições, a seguir tais autômatos serão apresentados formalmente.

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

A representação usando grafos de transição dos λ-AFN e similar a representação dos AFN da seção anterior, a única diferença é que podem haver transições rotuladas pelo símbolo λ, isto é, podem existir no grafo arestas entre vértices que são rotuladas por λ, 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 λ-AFN.

Q \ $\Sigma \cup \{\lambda\}$	0	1	λ
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 λ-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 λ-AFN.

Q \ $\Sigma \cup \{\lambda\}$	0	1	λ
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 $\underline{\delta_N}$ do AFN no Exemplo 2.3.2.

e representado pela figura 2.10 a seguir.

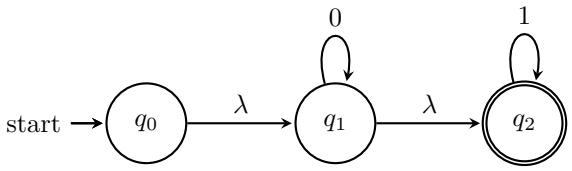


Figura 2.10: Grafo de transição do λ-AFN do Exemplo 2.3.2.

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

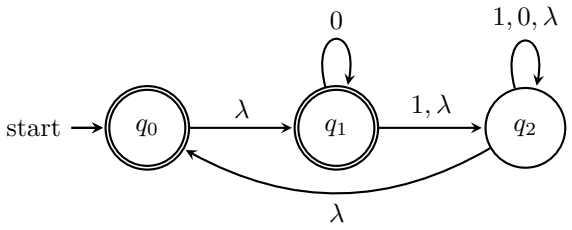


Figura 2.11: Grafo de transição do λ-AFN do Exemplo 2.3.1.

Uma interpretação para as transições da forma $\underline{\delta_N}(q, \lambda) = X$ 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 $\underline{\delta_N}$ garante que as transições em um λ-AFN acontecem apenas em duas situações, a primeira em relação símbolos individuais do alfabeto Σ 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 δ_λ) Seja $A = \langle Q, \Sigma, \underline{\delta_N}, q_0, F\rangle$ um λ-AFN, então a função $\delta_\lambda : Q \to \wp(Q)$ é definida como,

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

onde $n = \#Q - 1$ e

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

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

Exemplo 2.3.4 Considere o λ-AFN da Figura 2.10 tem-se para o estado q_1 que,

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

sabe-se que,

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

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

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

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,

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

Uma interpretação semântica para a função δ_λ é que ela representa a resposta ao questionamento: "Estando no estado q e executando n λ-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 λ-AFN, então a função $\widehat{\delta_\lambda} : \wp(Q) \to \wp(Q)$ é definida como,

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

Exemplo 2.3.5 Considere o λ-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 δ_λ e $\widehat{\delta_\lambda}$ pode-se apresentar a extensão da função de transição dos λ-AFN.

Definição 2.14 (λ-Transição não-determinística estendida) Dado um λ-AFN $A = \langle Q, \Sigma, \underline{\delta_N}, q_0, F \rangle$, a função $\underline{\delta_N}$ é estendido para a função $\widehat{\underline{\delta_N}} : Q \times \Sigma^* \to \wp(Q)$ definida pela seguinte recursão:

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

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

Exemplo 2.3.6 Considere o λ-AFN da Figura 2.11 tem-se a seguinte computação para a palavra

"10":

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

mas,

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

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

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

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

Definição 2.15 (Linguagem de um λ-AFN) Seja $A = \langle Q, \Sigma, \underline{\delta_N}, q_0, F\rangle$ um λ-AFN a linguagem aceita por A, denotado por $\mathcal{L}(A)$, corresponde ao seguinte conjunto $\mathcal{L}(A) = \{w \in \Sigma^* \mid \widehat{\underline{\delta_N}}(q_0, w) \cap F \neq \emptyset\}$.

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

Exemplo 2.3.7 O λ-AFN representado pelo grafo de transição da Figura 2.12 a seguir reconhece a linguagem $L = \{w \in \{1, 2, 3\}^* \mid w = 1^i 2^j 3^k \text{ com } i, j, k \in \mathbb{N}\}$.

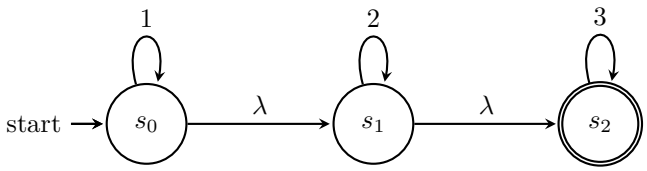


Figura 2.12: Grafo de transição do λ-AFN do Exemplo 2.3.7.

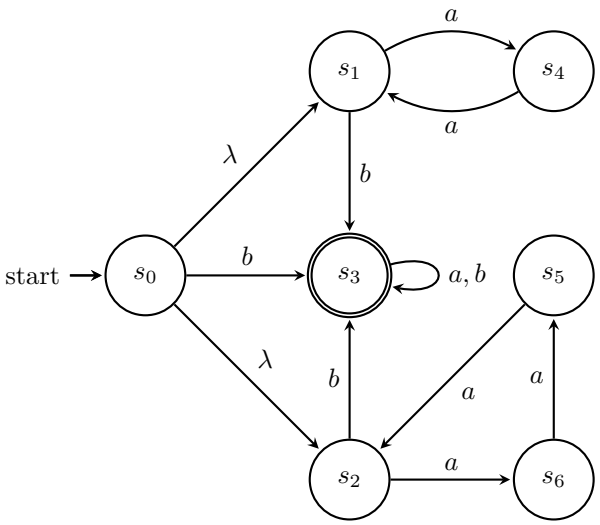


Figura 2.13: Grafo de transição do λ-AFN do Exemplo 2.3.8.

Exemplo 2.3.8 O λ-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 λ-AFN-AFD) Se $L = \mathcal{L}(A)$ para algum λ-AFN A, então existe um AFD A' tal que $L = \mathcal{L}(A')$.

Prova Suponha que $L = \mathcal{L}(A)$ para algum λ-AFN $A = \langle Q, \Sigma, \underline{\delta_N}, q_0, F\rangle$ 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:

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

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

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

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

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

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

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

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

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.

□

[a] A prova desse fato fica como exercício ao leitor.

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

Prova Trivial e ficará como exercício ao leitor. □

Corolario 3 Uma linguagem L é regular se, e somente se, existe um λ-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 λ-AFN A tal que $L = \mathcal{L}(A)$. ($\Leftarrow$) Suponha que $L = \mathcal{L}(A)$ para algum λ-AFN A, agora pelo Teorema 5 existe um AFD A' tal que $L = \mathcal{L}(A')$, e portanto, por definição L é regular. □

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, 18, 20], algumas simples modificações no Algoritmo 2 fazem com que o novo algoritmo gerado seja capaz de remover as λ-transições e não sejam produzidos estados inacessíveis a seguir é apresentado este novo algoritmo.

Algoritmo 2: Algoritmo para remoção de λ-transições de um λ-AFN.

Entrada: Um λ-AFN $A = \langle Q, \Sigma, \underline{\delta_N}, q_0, F \rangle$
Saída: Um AFD $A' = \langle Q', \Sigma, \delta, \delta_\lambda(q_0), F' \rangle$

início
  Inicialize os conjuntos Q_u e Q' com o estado rotulado por $\delta_\lambda(q_0)$
  Inicialize o conjunto F' como sendo vazio
  repita
    Selecione um estado $X \in Q_u$
    para cada $a \in \Sigma$ **faça**
      Determine o conjunto $Y = \widehat{\delta_\lambda}\Big(\bigcup_{q \in X} \underline{\delta_N}(q, a)\Big)$
      se $Y \notin Q'$ **então**
        Adicione um estado rotulado por Y em Q' e em Q_u
      fim
      Defina a transição $\delta(X, a) = Y$
    fim
    Remova X de Q_u
  até $Q_u = \emptyset$;
  para cada $X \in Q'$ **faça**
    se $X \cap F \neq \emptyset$ **então**
      Adicione X ao conjunto F'
    fim
  fim
  retorna $A' = \langle Q', \Sigma, \delta, \delta_\lambda(q_0), F' \rangle$
fim

Exemplo 2.3.9 Aplicando o Algoritmo 2 ao λ-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 δ é descrito pela Tabela a seguir

Q \ Σ	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 λ-AFN do Exemplo 2.3.8.

Nestas últimas seções foram usados os símbolos δ, δ_N e $\underline{\delta_N}$ para denotar as funções de transições dos AFD, AFN e λ-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, 18, 20]) que independente do tipo de autômato, sua função de transição será sempre representada apenas por δ.

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 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[4] (1923-1987) e Anil Nerode (1932-), que o provaram na Universidade de Chicago em 1958 no artigo [29], 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.

[4] O professor Myhill também é conhecido por seu Teorema de isomorfismo[28], 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, 32].

Definição 2.16 (A família $\mathcal{H}_L$) Seja L uma linguagem qualquer[a] sobre o alfabeto Σ, 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, $\mathcal{H}_L = \{L_w \mid w \in \Sigma^*\}$

[a] 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:

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

concluindo a prova. □

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. □

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,

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

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, $\#\mathcal{H}_L \leq \#Q$. □

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 $L \subseteq \Sigma^*$ assuma 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:

$$q_0 = L \tag{2.27}$$

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

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

e

$$F = \{L_w \in \mathcal{H}_L \mid \lambda \in L_w\} \tag{2.29}$$

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

$$\widehat{\delta}(q_0, w) = L_w \tag{2.30}$$

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

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

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

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 $f : Q \to \mathcal{H}_L$ definida simplesmente como:

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

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 $q, p \in Q$ assuma 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[a].

(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 $w \in \Sigma^*$ assuma 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 $L_w \in Ima(f)$ tem-se que $L_w = \mathcal{L}(A_q)$, e portanto, pela definição de f tem-se que existe $q \in Dom(f)$ 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. □

[a] 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. □

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 distinguiveis, o primeiro deles a ser apresentado aqui é o algoritmo descrito pela primeira vez em 1956 por Edward Moore [16, 27][5], em seguida será apresentado o algoritmo de Hopcroft [17], apresentado inicialmente em 1971.

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

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 $\widehat{\delta}(q,w) \in F \iff \widehat{\delta}(q',w) \in F$.

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 respeitos de estados equivalêntes 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) $\widehat{\delta}(\delta(q,a),w) \in F$ e $\widehat{\delta}(\delta(q',a),w) \notin F$ ou

(2) $\widehat{\delta}(\delta(q,a),w) \notin F$ 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. □

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,

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

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. □

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, $p \in F$. □

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:

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

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

O Teorema 10 é uma excelente justificativa para os algoritmos que iremos apresentar a seguir, pois ambos os algoritmos constroem extamente o AFD $A_{/\equiv}$ a partir de um AFD A dado como entrada. Para o funcionamento dos algoritmo de Moore em especial, é necessário 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 × #Q
3  |  para cada q_i ∈ Q faça
4  |  |  para cada q_j ∈ Q faça
5  |  |  |  se (q_i ∈ F e q_j ∉ F) ou (q_i ∉ F e q_j ∈ F) então
6  |  |  |  |  Marque a posição M_{i,j} com um ✓
7  |  |  |  fim
8  |  |  fim
9  |  fim
10 |  retorna M
11 fim
```

Notavelmente o Algoritmo 3 pode ser melhorado de diversar 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 distinguiveis) (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

```
início
    Defina uma matriz M de ordem #Q × #Q
    para cada q_j ∈ Q faça
        para cada q_j ∈ Q faça
            se M_{i,j} não estiver marcado então
                para cada q_j ∈ Q faça
                    se δ(q_i, a) ≢ δ(q_j, a) então
                        Marque a posição M_{i,j} com um ✓
                    fim
                fim
            fim
        fim
    fim
    retorna M
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 $[q] = [p \in Q \mid M_{q,p}$ não está marcada com ✓$]$.
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 $L = \{w \in \{0,1\}^* \mid w = u11v, u, v \in \{0,1\}^*\}$.

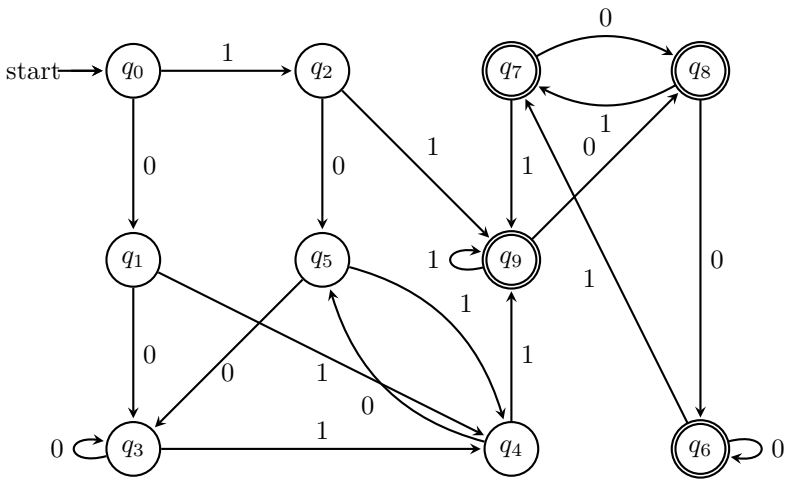


Figura 2.14: Um AFD.

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

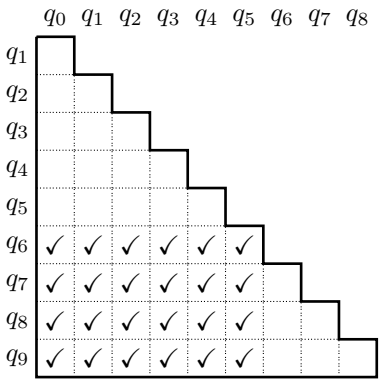


Figura 2.15: Matriz de estados do AFD da Figura 2.14 candidatos a serem equivalentes.

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 matriz será atualizada, ficando na forma abaixo.

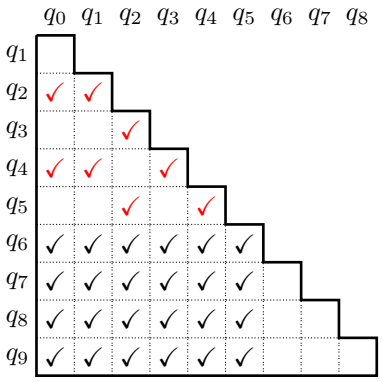


Figura 2.16: Matriz de estados equivalentes do AFD da Figura 2.14.

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,

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

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.

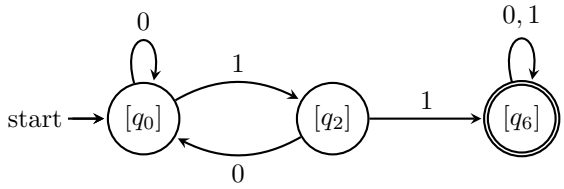


Figura 2.17: Um AFD.

Exemplo 2.5.2 Considere o AFD representado pela Figura 2.18 a seguir.

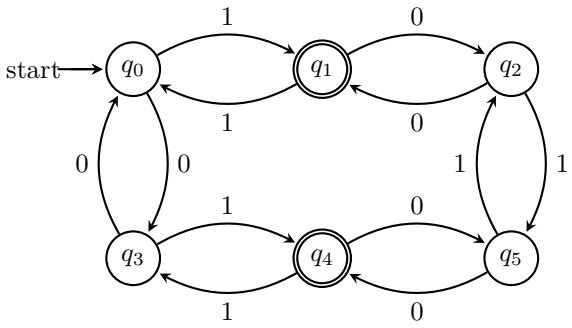


Figura 2.18: Um AFD.

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.

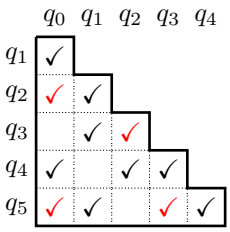


Figura 2.20: Matriz de estados equivalentes do AFD da Figura 2.18.

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.

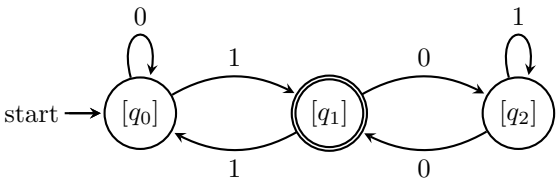


Figura 2.21: Um AFD.

Agora será apresentado o algoritmo de Hopcroft [17] para minimiza um AFD dado, a ideia de Hopcroft é realizar o refinamento progressivo das partições do conjunto de estados, usando a ideia de blocos [6]. O conceito é quebrar o conjunto de estados em subconjuntos contendo apenas estados equivalentes.

[6] 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[a].

[a] 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.

(a). $L_1 = \{w \in \{a, b\}^* \mid w \text{ termina em } ab\}$.

(b). $L_2 = \{w \in \{0, x, 1\}^* \mid w \text{ começa com 1 mas não contém a subpalavra } 0xx1\}$.

(c). $L_3 = \{w \in \{1, 2, 0\}^* \mid w \text{ contém a subpalavra } 1201\}$.

(d). $L_4 = \{w \in \{a,b\}^* \mid w = a^n$ com $n \in \mathbb{N}$ ou se existe b em w, ele é seguido de aa$\}$.

(e). $L_5 = \{w \in \{0,1\}^* \mid w$ não possui nem uma subpalavra 0^{2n} com $n \in \mathbb{N} - \{0\}\}$.

(f). $L_6 = \{w \in \{a,b,c\}^* \mid w \in \{b,c\}^*$ ou todo a em w aparece entre b e $c\}$.

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

(a). $L_1 = \{w \in \{\circ, \star, 0, 1\}^* \mid N_\star(w) > 3 \wedge N_\star(w) \neq 2k, k \in \mathbb{N}\}$.

(b). $L_2 = \{w \in \{0, x, 1\}^* \mid N_x(w) = 6\}$.

(c). $L_3 = \{u0^i1^j v \in \{1,2,0\}^* \mid u, v \in \{2\}^* \wedge i + j = 2k + 1, k \in \mathbb{N} - \{0\}\}$.

(d). $L_4 = \{u\#b\#v \mid u, v \in \{a,b,c\}^*, N_b(u) = 0 \wedge N_b(v) < 5\}$.

Questão 2.3 Considere que $\Sigma = \{0,1\}$, para cada linguagem a seguir construa um AFD, AFN ou λ-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 λ-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 ($\overline{\ }$), 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 Σ 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). $L_9 = \{xy \in L \mid y \in L \iff x = \lambda\}$.

Capítulo 3

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 [26], e o modelo de gramática geradora [11].

3.1 Gramática Regulares

Nesta seção será apresentado um terceiro formalismo para as linguagens regulares, sendo este um formalismo gerador (ou axiomático) [26].

Definição 3.1 Uma gramática formal $G = \langle V, \Sigma, S, P\rangle$ e dita Linear à Direta (à Esquerda), ou simplesmente GLD (GLE), se todas as suas produções são da forma, $A \to wB$ ($A \to Bw$), 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:

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

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:

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

é uma GLE.

Em uma gramática formal G quando para as sentenças $w, w_1, \cdots, w_n \in (V \cup \Sigma)^*$, existem para todo $1 \leq i \leq n$ a regras $w \to w_i \in P$, é comum para simplificar a escrita do conjunto de regras usar a notação $w \to w_1 \mid \cdots \mid w_n$, 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:

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

Um tipo mais rigoroso de gramática, são as chamadas gramática regulares, que como comentado em [20], 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 $A \to Bw$ ($A \to wB$) com $A \in V, B \in V \cup \{\lambda\}$ e $w \in \Sigma \cup \{\lambda\}$.

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

$$\begin{aligned} S &\to 0A \mid 1B \mid \lambda \\ A &\to 1S \mid 1C \\ B &\to 0S \mid 0D \\ C &\to 0S \mid \lambda \\ D &\to 1S \mid \lambda \end{aligned}$$

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 (⇒) 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 $A \to Bw \in P$ onde $B \in V \cup \{\lambda\}, |w| \leq 1$, então $A \to Bw \in P'$.

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

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

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, $\mathcal{L}(G) = \mathcal{L}(G')$. (⇐) Trivial uma vez que toda gramática regular à esquerda (ou à direita) é um caso particular de gramática linear à esquerda (ou à direita). □

Exemplo 3.1.5 Considerando a gramática linear do Exemplo 3.1.3 usando o Teorema 11 é gerado a gramática regular unitária,

$$G' = \langle \{S, B_3, B_2, C_3, C_2, X, D_2, Y\}, \{0,1\}, S, P' \rangle$$

onde P' é formado pelas regras:

$$\begin{aligned} S &\to B_3 1 \mid C_3 1 \\ B_3 &\to B_2 0 \\ B_2 &\to X0 \\ C_3 &\to C_2 1 \\ C_2 &\to Y0 \\ X &\to D_2 1 \\ D_2 &\to S0 \\ Y &\to \lambda \end{aligned}$$

Obviamente a gramática do Exemplo 3.1.5 poderia ser otimizada para usar menos variáveis, porém otimização não é o foco de interesse no Teorema 11. Os próximos resultados estabelecem o poder de geração das gramáticas regulares à direta.

Lema 6 Se $G' = \langle V, \Sigma, S, P \rangle$ é uma gramática regular à direita, então existe uma gramática regular à direita $G' = \langle V', \Sigma, S, P' \rangle$ e toda regra em G' são das formas $A \to aB$ ou $A \to \lambda$ com $a \in \Sigma \cup \{\lambda\}$ e $A, B \in V$.

Prova Suponha que $L = \mathcal{L}(G)$ para alguma gramática regular à direta $G = \langle V, \Sigma, S, P \rangle$, agora construa uma nova gramática $G' = \langle V \cup \{T\}, \Sigma, S, P' \rangle$ com $T \notin V$ e P' sendo construído pelas seguintes regras:

(r_a) Se $A \to aB \in P$ com $a \in \Sigma \cup \{\lambda\}$ e $A, B \in V$, então $A \to aB \in P'$.

(r_b) Se $A \to a \in P$ com $A \in V$ e $a \in \Sigma$, então $A \to aT, T \to \lambda \in P'$.

note que as regras r_a e r_b garante que não existem transições da forma $A \to a$ em P' quando $a \in \Sigma$. Agora por indução sobre o tamanho das formas sentenciais, será mostrado que G' preserva todas as derivações de G.

- **(B)**ase da indução: Quando $S \vdash_G w$ em uma única derivação, três casos podem ocorrer:
  (a) Quando $w = \lambda$ claramente foi aplicada uma regra da forma $S \to \lambda \in P$, e pela contrução de P' claramente $S \to w \in P'$, assim $S \vdash_{G'} w$.
  (b) Quando $w = a$ claramente foi aplicada uma regra da forma $S \to a \in P$, e pela contrução de P' tem-se que $S \to aT, T \to \lambda \in P'$, consequentemente, $S \vdash^*_{G'} w$.
  (c) Quando $w = aB$ claramente foi aplicada uma regra da forma $S \to aB \in P$, e pela contrução de P' tem-se que $S \to aB \in P'$, e portanto, $S \vdash_{G'} w$.
- **(H)**ipótese indutiva: Suponha que $S \vdash^*_G a_1 \cdots a_{n-1} A_n$ com $n \geq 1$ derivações e $S \vdash^*_{G'} a_1 \cdots a_{n-1} A_n$ com n ou $n+1$ derivações.
- **(P)**asso indutivo: Seja $S \vdash^*_G a_1 \cdots a_{n-1} a_n$ com $n+1$ derivações, logo existe uma forma sentencial $a_1 \cdots a_{n-1} A_n$ derivada em G com n derivações e $A_n \to a \in P$ com $a \in \Sigma \cup \{\lambda\}$. Agora por **(H)**, existe $a_1 \cdots a_{n-1} A_n$ é derivada em G com n derivações e em G' com n ou $n+1$ derivações. Como $S \vdash^*_G a_1 \cdots a_{n-1} a_n$ com $n+1$ derivações isso implica que $A_n \to a_n \in P$ com $a_n \in \Sigma \cup \{\lambda\}$, logo quando $a_n = \lambda$ então $A_n \to a_n \in P'$ e, portanto, $S \vdash^*_{G'} a_1 \cdots a_{n-1} a_n$ com $n+1$ derivações, por outro lado, quando $a_n \in \Sigma$ tem-se que $A_n \to a_n T, T \to \lambda \in P'$, e consequentemente, $S \vdash^*_{G'} a_1 \cdots a_{n-1} a_n$ com $n+2$ derivações.

Portanto, o raciocínio indutivo anterior garante que G' preserva todas as derivações de G, logo quando $w \in \mathcal{L}(G)$ tem-se que $w \in \mathcal{L}(G')$, consequentemente, $\mathcal{L}(G) \subset \mathcal{L}(G')$. Por outro lado, pelo construção de G' tem-se sem perda de generalidade[1] que $a_1 \cdots a_n \neq \lambda$ é tal que,

[1] O caso $a_1 \cdots a_n = \lambda$ é trivial e não será demonstrado aqui.

$$\begin{aligned}
a_1 \cdots a_n \in \mathcal{L}(G') &\Rightarrow (\exists A_n \in V')[S \vdash^*_{G'} a_1 \cdots a_n A_n \wedge A_n \to \lambda \in P'] \\
&\Rightarrow (\exists A_1, \cdots, A_n \in V')[S \to a_1 A_1, \\
&\quad \cdots, A_{n-1} \to a_n A_n, A_n \to \lambda \in P'] \\
&\Rightarrow (\exists A_1, \cdots, A_n \in V)[S \to a_1 A_1, \\
&\quad \cdots, A_{n-1} \to a_n A_n, A_n \to \lambda \in P] \\
&\quad \vee (\exists A_1, \cdots, A_{n-1} \in V)[S \to a_1 A_1, \cdots, A_{n-1} \to a_n \in P] \\
&\Rightarrow (\exists A_n \in V)[S \vdash^*_G a_1 \cdots a_n A_n \wedge A_n \to \lambda \in P] \\
&\quad \vee (\exists A_{n-1} \in V)[S \vdash^*_G a_1 \cdots a_{n-1} A_n \wedge A_n \to a_n \in P] \\
&\Rightarrow a_1 \cdots a_n \in \mathcal{L}(G) \vee a_1 \cdots a_n \in \mathcal{L}(G) \\
&\Rightarrow a_1 \cdots a_n \in \mathcal{L}(G)
\end{aligned}$$

E portanto, $\mathcal{L}(G') \subset \mathcal{L}(G)$. Agora desde que $\mathcal{L}(G) \subset \mathcal{L}(G')$ e $\mathcal{L}(G') \subset \mathcal{L}(G)$, tem-se que $\mathcal{L}(G') = \mathcal{L}(G)$, o que completa a prova. □

Teorema 12 $L = \mathcal{L}(G)$ para alguma gramática regular à direta G se, e somente se, L é uma linguagem regular.

Prova (⇒) Suponha que $L = \mathcal{L}(G')$ para alguma gramática regular à direta $G' = \langle V, \Sigma, S, P\rangle$ tal que $L = \mathcal{L}(G')$, sem perda de generalidade, pelo Lema 6 pode-se assumir que toda regra em P é da forma $A \to aB$ ou $A \to \lambda$ com $A, B \in V$ e $a \in \Sigma \cup \{\lambda\}$, dito isto, pode-se agora construir um λ-AFN $M = \langle V, \Sigma, \underline{\delta_N}, S, F\rangle$ tal que:

$$B \in \underline{\delta_N}(A, a) \iff A \to aB \in P$$

como $A, B \in V$ e $a \in \Sigma \cup \{\lambda\}$ e $F = \{A \in V \mid A \to \lambda \in P\}$. Agora será mostrado por indução sobre o tamanho das derivações em G' que se w é derivada por G' e $w \in \Sigma^*$, então é aceita por M.

- **(B)**ase da indução: Quando w é derivada em G' com uma única derivação tem-se então duas situações possíveis:
  (1) Quando $w = \lambda$, obrigatoriamente existe uma regra da forma $S \to \lambda$, e pela construção de M tem-se que $S \in \underline{\delta_N}(S, \lambda)$, logo $\widehat{\underline{\delta_N}}(S, \lambda) \cap F \neq \emptyset$ e, portanto, $\lambda \in L(M)$.
  (2) Quando $w = aB$, existe em P uma regra da forma $S \to aB$ com $a \in \Sigma \cup \{\lambda\}$ e $B \in V$, assim pela construção de M tem-se que $B \in \underline{\delta_N}(A, a)$. Como $aB \notin \Sigma^*$ não há mais nada a fazer nesse caso.
- **(H)**ipótese indutiva: Suponha que $S \vdash^*_{G'} w$ em n derivação com $n \geq 1$ tal que:
  (1) Se $w \in \Sigma^*$, então $w \in \mathcal{L}(M)$.
  (2) Se $w = a_1 \cdots a_{n-1} B$ com $a_i \in \Sigma \cup \{\lambda\}$ para todo $1 \leq i \leq n-1$ e $B \in V$, então $B \in \widehat{\underline{\delta_N}}(S, a_1 \cdots a_{n-1})$.
- **(P)**asso indutivo: Agora dado que $S \vdash^*_{G'} w'$ em $n+1$ derivações, tem-se obrigatoriamente que acontece o caso (2) de **(HI)** e nesse caso duas situações são possíveis:
  (1) Se $w' \in \Sigma^*$, então $w = a_1 \cdots a_{n-1} B$ com $a_i \in \Sigma \cup \{\lambda\}$ para todo $1 \leq i \leq n-1$ e existe em P uma produção $B \to \lambda$, e assim, $w' = a_1 \cdots a_{n-1}$, nesta situação pelo caso (2) de **(HI)** tem-se que $B \in \widehat{\underline{\delta_N}}(S, a_1 \cdots a_{n-1})$ e

como $B \to \lambda$ pela construção de M tem-se que $B \in F$, consequentemente, $\widehat{\underline{\delta_N}}(S, a_1 \cdots a_{n-1}) \cap F \neq \emptyset$ e, portanto, $w' \in \mathcal{L}(M)$.

(2) Se $w' = a_1 \cdots a_{n-1}B$ com $a_i \in \Sigma \cup \{\lambda\}$ para todo $1 \leq i \leq n-1$ e $B \in V$, então pela construção de M tem-se que $B \in \widehat{\underline{\delta_N}}(S, w)$ como $w' \notin \Sigma^*$ não há mais nada a fazer nesse caso.

Portanto, o raciocínio indutivo anterior garante que sempre que w é derivada por G' e $w \in \Sigma^*$ tem-se que $w \in \mathcal{L}(M)$ e assim pode-se afirmar pelo Corolário 3 que L é regular.
($\Leftarrow$) Suponha que L é uma linguagem regular assim por definição existe um AFD $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ tal que $L = \mathcal{L}(M)$, assim construa uma gramática regular à direita $G = \langle Q, \Sigma, q_0, P\rangle$ onde o conjunto P é definido usando as regras a seguir,

(a) Se $\delta(q_i, a) = q_j$, então $q_i \to aq_j \in P$.

(b) Se $q_i \in F$, então $q_i \to \lambda \in P$.

Agora note que para todo $w \in \Sigma^*$ com $w = a_1 \cdots a_n$ tem-se que,

$$\begin{aligned}
w \in \mathcal{L}(M) &\iff \widehat{\delta}(q_0, w) \in F \\
&\iff \widehat{\delta}(q_0, a_1 \cdots a_n) \in F \\
&\iff (\exists q_f \in F)[\widehat{\delta}(q_0, w) = q_f] \\
&\iff (\exists q_1, \cdots, q_{n-1} \in Q, q_f \in F) \\
&\quad\ [\delta(q_0, a_1) = q_1 \wedge \cdots \wedge \delta(q_{n-1}, a_n) = q_f] \\
&\iff (\exists q_1, \cdots, q_{n-1} \in Q, q_f \in F) \\
&\quad\ [q_0 \to a_1 q_1, \cdots, q_{n-1} \to a_n q_f, q_f \to \lambda \in P] \\
&\iff q_0 \vdash^* a_1 \cdots a_n \\
&\iff q_0 \vdash^* w \\
&\iff w \in \mathcal{L}(G)
\end{aligned}$$

portanto, $\mathcal{L}(M) = \mathcal{L}(G)$ o que conclui a prova. □

Lema 7 Se L é gerada por uma gramática à direita, então L^r é gerada por uma gramática regular à direita.

Prova Suponha que L é gerada por uma gramática à direita G, ou seja, $L = \mathcal{L}(G)$, assim pelo Teorema 12 tem-se que L é regular, logo existe um AFD $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ tal que $L = \mathcal{L}(M)$, agora construa um λ-AFN $M_1 = \langle Q \cup \{q_f\}, \Sigma, \underline{\delta_N}, q_0, \{q_f\}\rangle$ tal que,

$$\underline{\delta_N}(q, a) = \begin{cases} \{\delta(q, a)\}, & \text{se } q \in Q, a \in \Sigma \\ \{q_f\}, & \text{se } q \in F, a = \lambda \\ \emptyset, & \text{qualquer outro caso} \end{cases}$$

claramente $L = \mathcal{L}(M_1)$, agora construa um novo λ-AFN $M_2 = \langle Q \cup \{q_f\}, \Sigma, \underline{\delta'_N}, q_f, \{q_0\}\rangle$ onde para todo $q \in Q \cup \{q_f\}$ e $a \in \Sigma \cup \{\lambda\}$ tem-se que,

$$q_i \in \underline{\delta'_N}(q_j, a) \iff q_j \in \underline{\delta_N}(q_i, a)$$

pela construção de M_2 é claro que $w \in \mathcal{L}(M_1) \iff w^r \in \mathcal{L}(M_2)$ e, portanto, $L^r = \mathcal{L}(M_2)$. Desde que M_2 é um λ-AFN pelo Corolário 3 tem-se que L^r é uma linguagem regular, consequentemente, pelo Teorema 12 existe uma gramática regular à direita G' tal que $L = \mathcal{L}(G')$, o que conclui a prova. □

O próximo resultado mostra que gramática regulares à esquerda e à direita são equivalentes.

Teorema 13 (Mudança de direção regular) L é gerada por uma gramática à esquerda se, e somente se, L é gerada por uma gramática regular à direita.

Prova

($\Rightarrow$) Suponha que L é gerada por uma gramática à esquerda $G = \langle V, \Sigma, S, P\rangle$, assim sem perda de generalidade[2] pode-se assumir que todas as regras em P são da forma $A \to Ba$ com $A \in V, B \in V \cup \{\lambda\}$ e $a \in \Sigma \cup \{\lambda\}$, agora construa um λ-AFN $M = \langle V \cup \{q_f\}, \Sigma, \underline{\delta_N}, S, \{q_f\}\rangle$ onde,

$$\begin{aligned} B \in \underline{\delta_N}(A, a) &\iff A \to Ba \in P \\ q_f \in \underline{\delta_N}(A, \lambda) &\iff A \to \lambda \in P \end{aligned}$$

Agora note que para todo $w = a_1 \cdots a_n \in \Sigma^*$ tem-se que,

$$\begin{aligned} w \in \mathcal{L}(G') &\iff a_1 \cdots a_n \in \mathcal{L}(G') \\ &\iff S \vdash^*_{G'} a_1 \cdots a_n \\ &\iff (\exists A_1 \cdots A_n, S \in V) \\ &\quad [S \to A_n a_n, A_n \to A_{n-1}a_{n-1}, \cdots, A_2 \to A_1 a_1, A_1 \to \lambda \in P'] \\ &\iff (\exists A_1 \cdots A_n, S \in V) \\ &\quad [A_n \in \underline{\delta_N}(S, a_n), A_{n-1} \in \underline{\delta_N}(A_n, a_{n-1}), \cdots, A_1 \in \underline{\delta_N}(A_2, a_1), \\ &\quad q_f \in \underline{\delta_N}(A_1, \lambda)] \\ &\iff a_n \cdots a_1 \in \mathcal{L}(M) \\ &\iff w^r \in \mathcal{L}(M) \end{aligned}$$

Logo $\mathcal{L}(M) = \mathcal{L}(G)^r$, desde que M é um λ-AFN tem-se pelo Corolário 3 que $\mathcal{L}(G)^r$ é regular, assim pelo Teorema 12 existe uma gramática regular à direita $\hat{G_1}$ que a gera, ou seja, $\mathcal{L}(\hat{G_1}) = \mathcal{L}(G)^r$, mas pelo Lema 7 irá existir outra gramática regular à direita $\hat{G_2}$ tal que $\mathcal{L}(\hat{G_2}) = \mathcal{L}(\hat{G_1})^r$, mas $\mathcal{L}(\hat{G_1})^r = (\mathcal{L}(G)^r)^r = (L^r)^r = L$, portanto, L é gerada por uma gramática regular à direita.
($\Leftarrow$) Suponha que que L é gerada por uma gramática regular à direita, ou seja, que existe uma gramática regular a direita G tal que $L = \mathcal{L}(G)$, assim pelo Lema 7 irá existir outra gramática regular à direita $G_1 = \langle V, \Sigma, S, P\rangle$ tal que $L^r = \mathcal{L}(G_1)$, sem perda de generalidade pode-se assumir todas as suas produções em G_1 são da forma $A \to aB$ com $A \in V, B \in V \cup \{\lambda\}$ e $a \in \Sigma \cup \{\lambda\}$. Dito isso construa uma nova gramática $G_2 = \langle V, \Sigma, S, P'\rangle$ onde $P' = \{A \to Ba \mid A \to aB \in P\}$, claramente G_2 é regular à esquerda. Mas pela construção de G_2 fica claro que $S \vdash^*_{G_1} w \iff S \vdash^*_{G_2} w^r$, logo $\mathcal{L}(G_1)^r = \mathcal{L}(G_2)$, mas desde que, $\mathcal{L}(G_1)^r = (L^r)^r = L$, tem-se então que L é gerada por uma gramática linear à esquerda, o que completa a prova. □

[2] Basta gerar uma nova gramática onde toda regra da forma $A \to a$ com $a \in \Sigma$ foi substituída pelas regras $A \to Ca$ e $C \to \lambda$ onde C é uma variável nova criada.

Exemplo 3.1.6 Dado a gramática regular à direita $G_1 = \langle \{A, B, C\}, \{a, b\}, A, P_1\rangle$ com P_1 é formado pelas seguintes regras,

$$\begin{aligned} A &\to aC \mid B \\ B &\to bB \mid \lambda \\ C &\to aA \end{aligned}$$

claramente $\mathcal{L}(G_1) = \{w \in \{a, b\}^* \mid w = a^{2m}b^n \text{ com } m, n \in \mathbb{N}\}$, agora usando a construção exposta pelo Teorema 13, é possível construir a gramática regular à esquerda $G_2 = \langle \{A, B, C\}, \{a, b\}, B, P_2\rangle$ onde P_2 é formado pelas seguintes regras,

$$\begin{aligned} B &\to Bb \mid Ab \mid A \\ A &\to Ca \mid \lambda \\ C &\to Aa \end{aligned}$$

e obviamente $\mathcal{L}(G_2) = \{w \in \{a, b\}^* \mid w = a^{2m}b^n \text{ com } m, n \in \mathbb{N}\}$.

3.2 Expressões regulares

Até agora as linguagens foram vistas sobre a ótica das máquinas de computação, isto é, sobre a perspectiva dos autômatos finitos, e sobre a ideia de gramática regulares, ou seja, estruturas geradoras. Em tais perspectivas as linguagens são vistas como sendo conjuntos de palavras sobre os quais as máquinas tinha a tarefa de reconhecer seus elementos, ou como elementos que deveriam ser gerados (ou derivados) pelas gramáticas.

Neste seção será apresentada uma visão alternativa, essa visão tem um aspecto mais algébrico para as linguagens. Tal visão foi introduzida pelo matemático e lógico americano Stephen Kleene em seu seminal *paper* intitulado como, "*Representation of events in nerve nets and finite automata*" [19]. Sobre a perspectiva introduzida por Kleene é importante mencionar que ela consiste de um sistema formal (com sintaxe e semântica), chamamos esse sistema formal de expressões regulares. Tal sistema tem a caracteristica de ser denotacional, isto é, a linguagem é denotada por uma expressão (em geral simples).

Definição 3.3 (Conjunto das Expressões Regulares (Sintaxe)) Seja Σ uma alfabeto, o conjunto de todas as expressões regulares sobre Σ, denotado por Exp_Σ, é o conjunto indutivamente gerado pelas seguintes regras.

(B)ase: $\emptyset, \lambda$ e cada $a \in \Sigma$, são expressões regulares[a].

(P)asso indutivo: Se $r_1, r_2 \in Exp_\Sigma$, então $r_1 + r_2, r_1 \cdot r_2, r_1^*, (r_1) \in Exp_\Sigma$.

(F)echo: Exp_Σ é exatamente o conjunto dos elementos obtidos a partir **(B)** ou usando-se uma quantidade finita (podendo ser nula) de aplicações de **(P)**.

[a] As expressões regulares da base costumam ser chamadas de expressões regulares primitivas.

No que diz respeito as expressões regulares é comum assumir que $+, \cdot, (,)^* \notin \Sigma$, assim uma expressão regular nem sempre é uma palavra sobre Σ, em geral os símbolos $+$ e $\cdot$ são lidos como soma e produto [10], e o * é lido com operador estrela de Kleene, ou simplesmente, estrela. Além disso, como dito em [8] se $r_1, r_2 \in Exp_\Sigma$, então costuma-se escrever $r_1 r_2$ em vez de $r_1 \cdot r_2$.

Exemplo 3.2.1 Considerando o alfabeto $\{0, 1\}$ tem-se que $(1+1)0, 01 \in Exp_\Sigma$. Essa afirmação pode ser verificada construindo a ávore de dedução de tais palavras, e isso é feito abaixo.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{1}\,\textbf{(B)}}{1+1}\,\textbf{(P)}}{(1+1)}\,\textbf{(P)} \quad \dfrac{}{0}\,\textbf{(B)}}{(1+1)0}\,\textbf{(P)} \qquad \dfrac{\dfrac{}{0}\,\textbf{(B)} \quad \dfrac{}{1}\,\textbf{(B)}}{01}\,\textbf{(P)}$$

e isso mostra que de fato $(1+1)0, 01 \in Exp_{\{0,1\}}$.

Exemplo 3.2.2 Considerando o alfabeto $\{a, b, c\}$ tem-se que:

$$\dfrac{\dfrac{}{a}\,\textbf{(B)} \quad \dfrac{\dfrac{\dfrac{}{\emptyset}\,\textbf{(B)} \quad \dfrac{\dfrac{\dfrac{}{b}\,\textbf{(B)} \quad \dfrac{\dfrac{}{c}\,\textbf{(B)}}{c^*}\,\textbf{(P)}}{bc^*}\,\textbf{(P)}}{(bc^*)}\,\textbf{(P)}}{\emptyset + (bc^*)}\,\textbf{(P)}}{(\emptyset + (bc^*))}\,\textbf{(P)}}{a(\emptyset + (bc^*))}\,\textbf{(P)}$$

logo $a(\emptyset + (bc^*)) \in Exp_{\{a,b,c\}}$.

A seguir será formalizado o conceito de semântica para as expressões regulares, sendo que tal semântica pode ser visto como uma **semântica denotacional**[3] [33], que apresenta significado as operações de soma e multiplicação. Antes porém, vale ressaltar que, uma expressão regular não é, em geral, uma palavra de alguma linguagem sobre Σ.

[3] Aquela em que as valoração usadas, são funções que mapeiam palavras da linguagem para funções parciais que representam o comportamento dos programas.

Definição 3.4 (Semântica das Expressão Regulares) Seja Exp_Σ o conjunto das expressões regulares sobre Σ, a semântica (ou interpretação) de Exp_Σ é uma função $\mathcal{L} : Exp_\Sigma \to \wp(\Sigma^*)$ definida recursivamente para todo $r, r_1, r_2 \in Exp_\Sigma$ pelas seguintes regras.

(i) Se $r \in \Sigma \cup \{\lambda\}$, então $\mathcal{L}(r) = \{r\}$.

(ii) Se $r = \emptyset$, então $\mathcal{L}(r) = \emptyset$.

(iii) Se $r = r_1 + r_2$, então $\mathcal{L}(r) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$.

(iv) Se $r = r_1 \cdot r_2$, então $\mathcal{L}(r) = \mathcal{L}(r_1)\mathcal{L}(r_2)$.

(v) Se $r = r_1^*$, então $\mathcal{L}(r) = (\mathcal{L}(r_1))^*$.

(vi) Se $r = (r_1)$, então $\mathcal{L}(r) = (\mathcal{L}(r_1))$.

3.3 Coisa que não são expressões regulares!

Muitos ambientes de computação e linguagens de programação oferecem suporte a padrões chamados *regex* que são consideravelmente mais gerais e poderosos do que as expressões regulares, vistas na seção anterior. Os *regex* oferecidos por linguagens de programação incluem símbolos especiais que representam negação, classes de caracteres (por exemplo, letras maiúsculas ou dígitos), intervalos contíguos de caracteres, limites de linha e de palavra, repetição limitada[4], referências a subexpressões anteriores (back-references) e até variáveis locais.

[4] Que algo oposto totalmente à repetição ilimitada (fecho de Kleene) permitida pelas expressões vistas aqui

Então resumidamente e de forma direta, apesar de sua etimologia óbvia, uma regex oferecida por uma linguagem de programação não é necessariamente uma expressão regular, e não necessariamente descreve uma linguagem regular[5].

[5] Veja a discussão em http://stackoverflow.com/a/1732454/775369

Outro tipo de padrão que frequentemente é confundido com expressões regulares são os *globs*, que são padrões usados na maioria dos programas de *shells* em sistemas *Unix-like* e em algumas linguagens de script para representar conjuntos de nomes de arquivos. *Globs* incluem símbolos para caracteres arbitrários únicos (?), caracteres únicos de um intervalo específico ([a-z]), substrings arbitrárias (*) e substrings de um conjunto finito especificado (foo,bar,z). *Globs* são, na verdade, significativamente menos poderosos do que expressões regulares [16].

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. λ-ALN: Autômatos Lineares Não-determinísticos com λ-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] J. Carroll and D. Long. *Theory of finite automata: with an introduction to formal languages*. Prentice Hall Upper Saddle River, NJ, New Jersey, 2ł edition edition, 1989.

[11] N. Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

[12] K. Cooper and L. Torczon. *Construindo Compiladores*, volume 1. Elsevier Brasil, 2017.

[13] 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.

[14] 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.

[15] C. De la Higuera. *Grammatical inference: Learning Automata and Grammars*. Cambridge University Press, London, 2010.

[16] J. Erickson. *Models of computation.* http://jeffe.cs.illinois.edu/teaching/algorithms/, 2014. Não publicado.

[17] 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.

[18] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Pearson Education India, USA, 3ł edition, 2008.

[19] S. C. Kleene. Representation of events in nerve nets and finite automata. Technical report, Rand Project Air Force Santa Monica CA, 1951.

[20] P. Linz. *An Introduction to Formal Languages and Automata.* Jones & Bartlett Learning, New York, 2006.

[21] S. Lipschutz. *Teoria dos Conjuntos.* McGraw-Hill do Brasil - LTDA/MEC, 1978.

[22] S. Lipschutz and M. Lipson. *Matemática Discreta.* Bookman Editora, 2013. Coleção Schaum.

[23] J. P. Martins. *Lógica e Raciocínio.* College Publications, 2014.

[24] 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.

[25] G. H. Mealy. A method for synthesizing sequential circuits. *The Bell System Technical Journal*, 34(5):1045–1079, 1955.

[26] P. B. Menezes. *Linguagens Formais e Autômatos.* Sagra-Dcluzzato, 1998.

[27] E. F. Moore. Gedanken-experiments on sequential machines. *Automata Studies*, 34:129–153, 1956.

[28] J. Myhill. Creative sets. *Journal of Symbolic Logic*, 22(1), 1957.

[29] A. Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

[30] M. O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.

[31] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[32] H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953.

[33] D. S. Scott and C. Strachey. *Toward a mathematical semantics for computer languages*, volume 1. Oxford University Computing Laboratory, Programming Research Group Oxford, 1971.

[34] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.