2 Autômatos finitos

2.1 AUTÔMATOS FINITOS DETERMINÍSTICOS

Este é um livro sobre modelos matemáticos de computadores e algoritmos. Neste e nos próximos dois capítulos, iremos definir modelos de computação progressivamente poderosos, dispositivos cada vez mais sofisticados para aceitar e gerar linguagens. Visto à luz do plano geral desta obra, este capítulo irá parecer uma introdução relativamente modesta: adotamos para o computador real um modelo extremamente restrito, denominado **autômato finito,**[1] **ou máquina de estados finitos**. O autômato finito compartilha com um computador real a característica de que ele tem uma "unidade central de processamento" com capacidade fixa, finita. Ele recebe como entrada uma cadeia, que lhe é fornecida em uma fita de entrada. Ele não produz nenhuma saída, exceto uma indicação, informando se essa entrada foi ou não considerada aceitável. Ele é, em outras palavras, um dispositivo de reconhecimento de linguagem, conforme o que foi descrito no final do Capítulo 1. O que torna o autômato finito um modelo tão restrito dos computadores reais é a *completa ausência de memória* externa ao seu processador central fixo.

Um modelo computacional assim tão simples poderia, à primeira vista, ser considerado muito trivial para merecer um estudo sério: qual seria a utilidade de um computador sem memória? No entanto, um autômato finito não é realmente isento de memória. Ele simplesmente tem uma capacidade de memória que é fixada "em sua fabricação" a qual não pode mais ser expandida. Pode-se argumentar que a capacidade de memória de qualquer computador é limitada – por restrições no orçamento, por limitações físicas, ou, em última instância, pelo tamanho do universo. É uma interessante questão filosófica discutir se o melhor modelo matemático para um computador teria uma memória finita ou ilimitada. Estudaremos esses dois tipos de modelo, começando pelo modelo finito, e, depois, nos concentrando muito mais no modelo ilimitado.

Mesmo que se pense, como nós, que a forma correta para modelar computadores e algoritmos deva considerar uma memória ilimitada, é importante primeiro ter certeza de que a teoria dos autômatos finitos esteja bem assimilada. Descobriremos assim sua riqueza e elegância, e, no momento em que ela estiver dominada, estaremos mais aptos a apreciar exatamente o impacto da adição de memória auxiliar sobre o seu poder de computação.

[1] Historicamente o termo *autômato* se referia a máquinas (robôs) projetadas para executarem atividades pré-programadas em resposta a instruções codificadas.

Uma razão adicional para estudarmos autômatos finitos é que eles permitem projetar vários tipos de algoritmos e programas de computador. Por exemplo, a fase de *análise léxica* de um compilador (durante a qual são identificados os elementos básicos de um programa, tais como 'begin' e '+') é freqüentemente baseada na simulação de um autômato finito. O problema de localizar uma ocorrência de uma cadeia dentro de outra – por exemplo, se quisermos saber se as cadeias *ar, água, terra* e *fogo* ocorrem ou não no texto de *Elementos da Teoria da Computação*[1], – também pode ser resolvida eficientemente por métodos originados na teoria dos autômatos finitos.

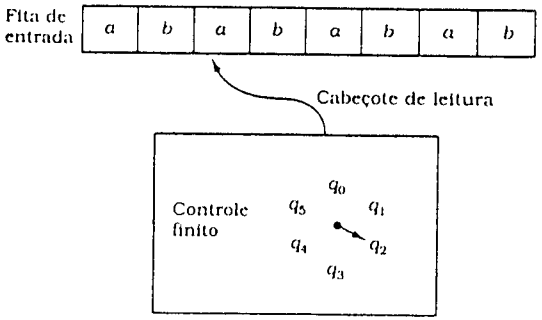


Figura 2-1

Vamos agora descrever com mais cuidado a operação de um autômato finito. Cadeias alimentam o dispositivo, que as recebe por meio da leitura de uma **fita de entrada**. Esta se compõe de células, com um símbolo gravado em cada célula da fita (veja a Figura 2-1). A parte principal da máquina propriamente dita é um bloco com mecanismos internos que podem se apresentar, em um momento específico, em algum dos seus diferentes **estados** internos. Esse bloco – denominado **controle finito** – pode ler o símbolo gravado em qualquer posição na fita de entrada com a ajuda de um **cabeçote móvel de leitura.** Inicialmente, o cabeçote de leitura está posicionado na célula mais à esquerda da fita, e o controle finito, configurado em um **estado inicial** pré-estabelecido. A intervalos regulares, o autômato lê um símbolo da fita de entrada e, então, entra em um novo estado, *que depende somente do estado atual e do símbolo que acabou de ser lido* – essa é a razão por que conhecemos esse dispositivo como *autômato finito determinístico*, em contraposição à versão *não-determinística*, a ser apresentada na próxima seção. Após a leitura de um símbolo de entrada, o cabeçote de leitura se movimenta uma célula para a direita na fita de entrada, posicionando-se de modo que, no próximo movimento, ele leia o símbolo contido na próxima célula da fita. Esse processo é repetido continuamente: um símbolo é lido, o cabeçote de leitura move-se para a direita e muda o

[1] Três dessas palavras aparecem neste livro.

estado do controle finito. Finalmente, o cabeçote de leitura alcança o final da cadeia de entrada. O autômato, então, indica sua aprovação ou desaprovação quanto ao que foi lido, com base no estado a que foi conduzido ao final desse processo: caso o estado resultante pertencer ao conjunto de **estados finais** do autômato, a cadeia de entrada é considerada **aceita**. A **linguagem aceita** pela máquina é o conjunto das cadeias que ela reconhece.

Quando essa descrição informal é expressa em seus princípios matemáticos básicos, o resultado é a seguinte definição formal:

Definição 2.1.1: Um **autômato finito determinístico** é uma quíntupla $M = (K, \Sigma, \delta, s, F)$, onde

K é um conjunto finito de **estados**,
Σ é um alfabeto,
$s \in K$ é o **estado inicial**,
$F \subseteq K$ é o conjunto de **estados finais**, e
δ, a **função de transição**, é uma função de $K \times \Sigma$ para K.

As regras segundo as quais o autômato M escolhe seu próximo estado são codificadas em sua função de transição. Assim, estando M no estado $q \in K$, e sendo o símbolo lido da fita de entrada $a \in \Sigma$, então $\delta(q, a) \in K$ será o estado, univocamente determinado, para o qual M[1] transitará.

Tendo formalizado a estrutura básica de um autômato finito determinístico, devemos em seguida representar, em termos matemáticos, a noção de *computação* de uma cadeia de entrada por um autômato, que se pode caracterizar como uma seqüência de **configurações**, que representam o aspecto da máquina (o controle finito, o cabeçote de leitura e a fita de entrada) em momentos sucessivos. Mas, como um autômato finito determinístico não tem a capacidade de mover o cabeçote de leitura para alguma posição já lida da cadeia de entrada, a subcadeia à esquerda do cabeçote de leitura nunca poderá interferir na operação da máquina. Assim sendo, uma configuração é determinada pelo estado corrente e pela parte ainda não lida da cadeia a ser processada. Em outras palavras, uma **configuração de um autômato finito determinístico** $(K, \Sigma, \delta, s, F)$ é qualquer elemento de $K \times \Sigma^*$. Por exemplo, a configuração ilustrada na Figura 2-1 é $(q_2, abaabab)$.

A relação binária $\vdash_M$ se aplica a um par de configurações de M se e somente se a máquina puder passar, de uma configuração para a outra, como resultado de um simples movimento. Deste modo, se (q, w) e (q', w') são duas configurações de M, então $(q, w) \vdash_M (q', w')$ se e somente se $w = aw'$ para algum símbolo $a \in \Sigma$, e $\delta(q, a) = q'$. Neste caso, dizemos que (q, w) **leva a** (q', w') **em um passo**. Note que de fato $\vdash_M$ é uma função de $K \times \Sigma^+$ para $K \times \Sigma^*$, isto é, para cada configuração, exceto aquelas da forma (q, ε), há uma única configuração seguinte. Uma configuração da forma (q, ε) indica que M consumiu todas as suas entradas encerrando assim sua operação.

Denotamos o fechamento transitivo reflexivo de $\vdash_M$ como $\vdash_M^*$; $(q, w) \vdash_M^* (q', w')$ lê-se "(q, w) **produz um resultado** (q', w')" (após algum número,

[1] N. de R. T. Erro no original: o livro diz K em lugar de M.

eventualmente nulo, de passos). Diz-se que uma cadeia $w \in \Sigma^*$ é **aceita** por M se e somente se existir algum estado $q \in F$, tal que $(s, w) \vdash_M^* (q, \varepsilon)$. Por fim, **a linguagem aceita por** M, $L(M)$, é o conjunto de todas as cadeias aceitas por M.

Exemplo 2.1.1: Seja M o autômato finito determinístico $(K, \Sigma, \delta, s, F)$, onde

$$K = \{q_0, q_1\},$$
$$\Sigma = \{a, b\},$$
$$s = q_0,$$
$$F = \{q_0\},$$

e δ a função tabelada a seguir.

q	σ	$\delta(q, \sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_1
q_1	b	q_0

É, então, fácil ver que $L(M)$ é o conjunto de todas as cadeias em $(a, b)^*$ que têm um número par de b's. M passa do estado q_0 para q_1 ou, de q_1, de volta a q_0, sempre que um b é lido, e ignora os a's, permanecendo sempre em seu estado corrente sempre que um a é lido. Portanto, M conta o número de b's, em módulo 2[1] e, como q_0 (o estado inicial) é também o único estado final, M reconhece uma cadeia, se e somente se o número de b's na cadeia for par.

Fornecendo-se a M uma entrada $aabba$, sua configuração inicial será $(q_0, aabba)$. Então

$$\begin{aligned}(q_0, aabba) &\vdash_M (q_0, abba)\\ &\vdash_M (q_0, bba)\\ &\vdash_M (q_1, ba)\\ &\vdash_M (q_0, a)\\ &\vdash_M (q_0, \varepsilon)\end{aligned}$$

Portanto, $(q_0, aabba) \vdash_M^* (q_0, \varepsilon)$, logo $aabba$ é aceita por M. ✧

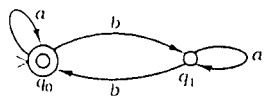


Figura 2-2

A representação tabular da função de transição, usada nesse exemplo não é a mais clara descrição que se pode fazer de uma máquina. Geralmente usamos uma representação gráfica mais conveniente, chamada **diagrama de**

[1] N. de R. T. A Contagem em módulo K é efetuada obtendo-se o resto da divisão do número por K.

estados (Figura 2-2). O diagrama de estados é um grafo orientado, com algumas informações adicionais incorporadas à figura: os estados são representados como nós do grafo, havendo uma seta, rotulada com a, do vértice q para q', sempre que $\delta(q, a) = q'$. Estados finais são denotados como círculos duplos, e o estado inicial é apontado pelo símbolo >. A Figura 2-2 mostra o diagrama de estados correspondente ao autômato finito determinístico do Exemplo 2.1.1.

Exemplo 2.1.2: Vamos projetar um autômato finito determinístico M que reconhece a linguagem $L(M) = \{w \in \{a, b\}^* : w$ não contém três b's consecutivos$\}$. Seja $M = (K, \Sigma, \delta, s, F)$, onde:

$$K = \{q_0, q_1, q_2, q_3\},$$
$$\Sigma = \{a, b\},$$
$$s = q_0,$$
$$F = \{q_0, q_1, q_2\},$$

e δ é dado pela seguinte tabela:

q	σ	$\delta(q,\sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_0
q_1	b	q_2
q_2	a	q_0
q_2	b	q_3
q_3	a	q_3
q_3	b	q_3

O diagrama de estados é mostrado na Figura 2-3. Para verificar que M de fato aceita a linguagem especificada, note que, não sendo lidos três b's consecutivos, M estará no estado q_i (onde i é 0, 1 ou 2), imediatamente após leitura de uma seqüência de i b's consecutivos, os quais: ou iniciam a cadeia de entrada, ou então são precedidos por um a. Em particular, sempre que um a é lido e M está no estado q_0, q_1 ou q_2, M retorna para seu estado inicial q_0. Os estados q_0, q_1 e q_2 são todos estados finais, assim, qualquer cadeia de entrada que não contenha três b's consecutivos será aceita. Entretanto, uma seqüência de três b's irá levar M ao estado q_3, o qual não é final, e M irá, então, permanecer nesse estado, independentemente dos símbolos encontrados no restante da cadeia de entrada. Diz-se que o estado q_3 é um *estado morto*, e, se M alcançar o estado q_3, diz-se que ele ficou preso (*trapped*), uma vez que nenhuma entrada adicional pode levá-lo a escapar desse estado. ✧

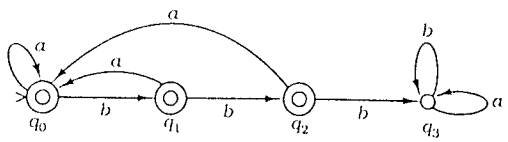


Figura 2-3

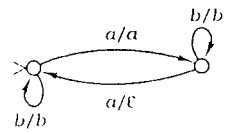
Problemas para a Seção 2.1

2.1.1 Seja M um autômato finito determinístico. Sob exatamente quais circunstâncias $\varepsilon \in L(M)$? Prove sua resposta.

2.1.2 Descreva informalmente as linguagens aceitas pelos autômatos finitos determinísticos apresentados na próxima página.

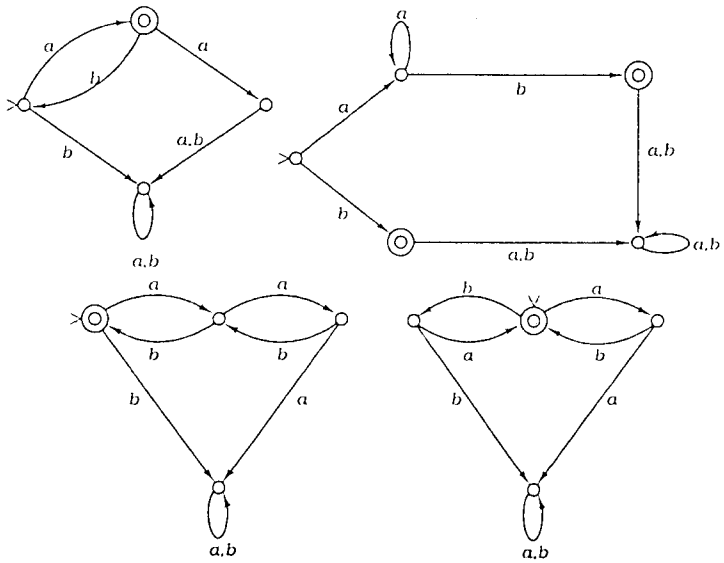
2.1.3 Construa autômatos finitos determinísticos que aceitem cada uma das seguintes linguagens:
(a) $\{w \in \{a, b\}^*$: cada a em w é imediatamente precedido por um $b\}$.
(b) $\{w \in \{a, b\}^*$: w contém a subcadeia $abab\}$.
(c) $\{w \in \{a, b\}^*$: w não contém aa nem bb como uma subcadeia$\}$.
(d) $\{w \in \{a, b\}^*$: w tem um número ímpar de a's e um número par de b's$\}$.
(e) $\{w \in \{a, b\}^*$: w tem ab e ba como subcadeias$\}$.

2.1.4 Um **transdutor de estados finitos determinístico** é um dispositivo muito parecido com um autômato finito determinístico, exceto que seu propósito não é o de aceitar cadeias ou linguagens, mas transformar cadeias de entrada em cadeias de saída. Informalmente, sua operação se inicia em um estado inicial preestabelecido, e move-se de estado para estado, de acordo com a entrada, exatamente como um autômato finito determinístico o faria. Em cada passo, entretanto, ele emite (ou escreve em uma fita de saída) uma cadeia de zero, um ou mais símbolos, em função do estado atual e do símbolo corrente de entrada. O diagrama de estados utilizado para representar um transdutor de estados finitos determinístico assemelha-se ao de um autômato finito determinístico, exceto que o rótulo utilizado sobre uma seta possui a forma a/w, que significa "se o símbolo de entrada for a, siga essa seta e emita w como saída". Por exemplo, o transdutor de estados finitos determinístico sobre $\{a, b\}$ mostrado a seguir transmite todos os b's encontrados na cadeia de entrada, mas omite um de cada dois a's encontrados.



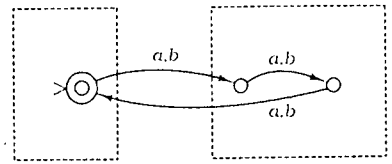
(a) Desenhe diagramas de estados representando transdutores de estados finitos determinísticos sobre $\{a, b\}$ que façam o seguinte:
(i) Com a entrada w, produzam a saída a^n, onde n é o número de ocorrências da subcadeia ab em w.
(ii) Com a entrada w, produzam a saída a^n, onde n é o número de ocorrências da subcadeia aba em w.
(iii) Com a entrada w, produzam uma cadeia de comprimento $|w|$* cujo i-ésimo símbolo seja um a se $i \geq 1$ e se o i-ésimo e o $(i-1)$-ésimo símbolos de w forem diferentes; nos demais casos, o i-ésimo símbolo da saída deve ser um b. Por exemplo, da entrada $aabba$ o transdutor deve gerar a saída $ababa$, e a entrada $aaaab$ deve produzir a saída $abbba$.

* N. de R. T. Erro no original: comprimento $|w|$ em lugar de comprimento w.



(b) Defina formalmente:
  (i) um transdutor de estados finitos determinístico;
  (ii) a noção de configuração para tal autômato;
  (iii) a relação $\vdash$ (movimento em um passo) entre configurações;
  (iv) a noção de que tal autômato produz a saída u para a entrada w;
  (v) a noção de que tal autômato computa uma função.

2.1.5 Um autômato finito determinístico de duas fitas é um dispositivo similar a um autômato finito determinístico, mas destinado a aceitar pares de cadeias. Cada um dos seus estados está em um de dois conjuntos; dependendo do conjunto a que o estado pertença, a função de transição refere-se à primeira ou à segunda fita. Por exemplo, o autômato mostrado acima reconhece todos os pares de cadeias $(w_1, w_2) \in \{a,b\}^* \times \{a,b\}^*$, tais que $|w_2| = 2|w_1|$.



(a) Desenhe diagramas de estados para autômatos finitos determinísticos de duas fitas que aceitem cada um dos seguintes:
  (i) Todos os pares de cadeias (w_1, w_2) em $\{a,b\}^* \times \{a,b\}^*$ tais que $|w_1| = |w_2|$ e $w_1(i) \neq w_2(i)$ para todos os i.
  (ii) Todos os pares de cadeias (w_1, w_2) em $\{a,b\}^* \times \{a,b\}^*$ tais que o comprimento de w_2 seja duas vezes o número de a's contidos em w_1 mais três vezes o número de b's contidos em w_1.
  (iii) $\{(a^n b, a^n b^m)\ n, m \geq 0\}$.
  (iv) $\{(a^n b, a^m b^n)\ n, m \geq 0\}$.

(b) Defina formalmente:
  (i) o autômato finito determinístico de duas fitas;
  (ii) a noção de configuração para tal autômato;
  (iii) a relação $\vdash$ de movimento em um passo entre configurações;
  (iv) a noção de que tal autômato reconhece um par ordenado de cadeias;
  (v) a noção de que tal autômato reconhece um conjunto de pares ordenados de cadeias.

2.1.6 Este problema refere-se aos de número 2.1.4 e 2.1.5. Mostre que, se $f: \Sigma^* \to \Sigma^*$ é uma função que pode ser computada por um transdutor de estados finitos determinístico, então $\{(w, f(w)): w \in \Sigma^*\}$ é um conjunto de pares de cadeias aceitas por algum autômato finito determinístico de duas fitas.

2.1.7 Dizemos que o estado q de um autômato finito determinístico $M = (K, \Sigma, \delta, q_0, F)$ é *alcançável* se existir $w \in \Sigma^*$ tal que $(q_0, w) \vdash_M^* (q, \varepsilon)$. Mostre que, se excluirmos de M qualquer estado não-alcançável, teremos ainda um autômato que reconhece a mesma linguagem. Forneça um algoritmo eficiente para determinar o conjunto de todos os estados alcançáveis de um autômato finito determinístico.

2.2 AUTÔMATOS FINITOS NÃO-DETERMINÍSTICOS

Nesta seção, adicionamos um poderoso e intrigante recurso aos autômatos finitos. Esse recurso é chamado **não-determinismo**, e consiste essencialmente na capacidade de mudar de estado de forma apenas parcialmente

determinada pelo estado corrente e pelo símbolo de entrada. Assim, devemos agora permitir vários possíveis "estados seguintes" para uma dada combinação de estado corrente e símbolo de entrada. Tendo lido a cadeia de entrada, o autômato pode escolher, a cada passo, dentre esses possíveis estados seguintes válidos: a escolha não é determinada por nada que esteja especificado em nosso modelo e, portanto, diz-se que ele é *não-determinístico*. Assim, tal escolha não é inteiramente arbitrária: somente podem ser escolhidos aqueles possíveis estados seguintes que são válidos a partir de um dado estado, para um dado símbolo de entrada.

Tais dispositivos não-determinísticos não foram concebidos para serem utilizados como modelos realistas de computadores. Eles são simplesmente uma generalização notacional útil dos autômatos finitos, já que podem simplificar significativamente a descrição desses autômatos. Além disso, veremos a seguir que o não-determinismo é uma característica não-essencial dos autômatos finitos: todo autômato finito não-determinístico tem um autômato finito determinístico equivalente. Portanto, devemos tirar proveito da poderosa notação dos autômatos finitos não-determinísticos, sempre levando em conta que, se necessário, poderemos a qualquer momento refazer tudo no nível mais baixo da linguagem dos autômatos determinísticos usuais.

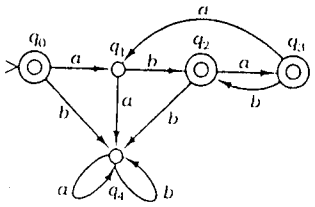


Figura 2-4

Para constatar que pode ser muito mais conveniente projetar um autômato finito não-determinístico do que um autômato finito determinístico, considere a linguagem $L = (ab \cup aba)^*$, que é aceita pelo autômato finito determinístico ilustrado na Figura 2-4. Mesmo na forma de diagrama, leva um certo tempo para que nos certifiquemos de que se trata realmente de um autômato finito determinístico; deve-se verificar que há exatamente duas setas saindo, uma de cada vértice, rotuladas com a e b. E algum raciocínio é necessário para se convencer de que a linguagem aceita por esse dispositivo relativamente complexo é a simples linguagem $(ab \cup aba)^*$. Poder-se-ia esperar descobrir que um autômato finito determinístico mais simples aceitasse L; infelizmente, pode-se demonstrar que nenhum autômato finito determinístico com menos de cinco estados pode aceitar essa linguagem (mais adiante neste capítulo desenvolveremos métodos para *minimizar* o número de estados dos autômatos finitos determinísticos).

De qualquer forma, L é aceita pelo dispositivo *não-determinístico* simples mostrado na Figura 2-5. Quando esse dispositivo está no estado q_1 e o símbolo de entrada é b, há dois possíveis estados seguintes, q_0 e q_2. Portanto, a Figura 2-5 não representa um autômato finito determinístico. Contudo, há um modo natural de interpretar o diagrama como um dispositivo que aceita L. Uma cadeia é aceita se existir alguma maneira de levar esse dispositivo do estado inicial (q_0) para um estado final (nesse caso, o próprio q_0), ao percorrer as setas rotuladas com os símbolos da cadeia. Por exemplo, a cadeia ab é aceita transitando-se de q_0 para q_1 e daí para q_0; aba é aceita transitando-se de q_0 para q_1, daí para q_2 e deste para q_0. Naturalmente, o dispositivo pode fazer tentativa errada e ir de q_0 para q_1, daí para q_0 e depois para q_1 com a entrada aba, terminando em um estado não-final, mas isso é irrelevante, pois existe *alguma* maneira de ele conseguir, a partir do estado inicial, atingir um estado final com essa entrada. Por outro lado, a cadeia de entrada abb não é aceita, já que não há como voltar de q_0 para q_0 usando essa cadeia.

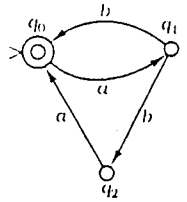


Figura 2-5

De fato, nota-se que, partindo de q_0, não há estado para onde transitar quando a entrada é b. Essa é outra característica dos autômatos finitos não-determinísticos: assim como, a partir de alguns estados e com algumas entradas, pode haver vários possíveis estados seguintes, assim também, com outras combinações de estados e símbolos de entrada, simplesmente pode não haver movimentos possíveis.

No diagrama de estados de um autômato não-determinístico também podemos colocar setas rotuladas com a cadeia vazia ε. Por exemplo, o dispositivo da Figura 2-6 reconhece a mesma linguagem L. De q_2 essa máquina pode retornar para q_0, tanto após a leitura do símbolo a como imediatamente, sem consumir qualquer entrada.

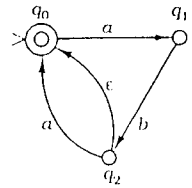


Figura 2-6

Os dispositivos ilustrados nas Figuras 2-5 e 2-6 são instâncias do seguinte modelo geral:

Definição 2.2.1: Um autômato finito não-determinístico é uma quíntupla $M = (K, \Sigma, \Delta, s, F)$, onde:

K é um conjunto finito de **estados**,
Σ é um **alfabeto**,
$s \in K$ é o **estado inicial**,
$F \subseteq K$ é o conjunto de **estados finais**, e
Δ, a **relação de transição**, é um subconjunto de $K \times (\Sigma \cup \{\varepsilon\}) \times K$.

Cada tripla $(q, u, p) \in \Delta$ é chamada uma **transição** de M – o correspondente formal de uma seta rotulada a de q para p no diagrama de estados de M. Se M está no estado q, e o símbolo de entrada corrente é a, então M pode seguir qualquer transição das formas (q, a, p) ou (q, ε, p); se uma transição (q, ε, p) é executada, então nenhum símbolo de entrada será lido.

As definições formais de computações por autômatos finitos não-determinísticos são muito similares às estabelecidas para autômatos finitos determinísticos. Uma configuração de M é, novamente, definida como um elemento de $K \times \Sigma^*$. A relação $\vdash_M$ (**produz resultado em um passo**) entre configurações é definida como segue: $(q, w) \vdash_M (q', w')$ se e somente se há um $u \in \Sigma \cup \{\varepsilon\}$ tal que $w = uw'$ e $(q, u, q') \in \Delta$. Note que $\vdash_M$ não precisa ser uma função; para algumas configurações (q, w), pode haver vários pares (q', w') – ou nenhum – tais que $(q, w) \vdash_M (q', w')$. Como antes, $\vdash_M^*$ é o fechamento transitivo reflexivo de $\vdash_M$ e uma cadeia $w \in \Sigma^*$ é dita **aceita** por M se e somente se houver um estado $q \in F$ tal que $(s, w) \vdash_M^* (q, \varepsilon)$. Por fim $L(M)$, a linguagem aceita por M, é o conjunto de todas as cadeias aceitas por M.

Exemplo 2.2.1: A Figura 2-7 mostra um dos vários possíveis autômatos finitos não-determinísticos que aceitam o conjunto de todas as cadeias contendo uma ocorrência do padrão *bb* ou do padrão *bab* (veja Seção 2.5 para um estudo sistemático de autômatos para detectar padrões na cadeia de entrada). Formalmente, essa máquina é $(K, \Sigma, \Delta, s, F)$, onde

$$K = \{q_0, q_1, q_2, q_3, q_4\},$$
$$\Sigma = \{a, b\},$$
$$s = q_0,$$
$$F = \{q_4\},$$

e

$$\Delta = \{(q_0, a, q_0), (q_0, b, q_0), (q_0, b, q_1), (q_1, b, q_2), (q_1, a, q_3), (q_2, \varepsilon, q_4), (q_3, b, q_4), (q_4, a, q_4), (q_4, b, q_4)\}.$$

Quando M é alimentada com a cadeia de entrada *bababab* diversas seqüências de movimento podem resultar. Por exemplo, M pode terminar

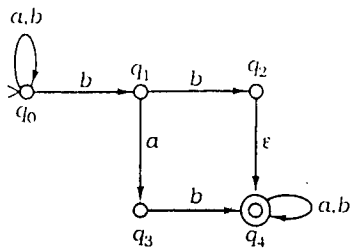


Figura 2-7

sua operação no estado não-final q_0, se forem utilizadas somente as transições (q_0, a, q_0) e (q_0, b, q_0):

$$\begin{aligned}(q_0, bababab) &\vdash_M (q_0, ababab)\\ &\vdash_M (q_0, babab)\\ &\vdash_M (q_0, abab)\\ &\vdots\\ &\vdash_M (q_0, \varepsilon)\end{aligned}$$

A mesma cadeia de entrada pode direcionar M do estado q_0 ao estado final q_4 e de fato isso pode ocorrer de três modos diferentes. Um dos quais é o seguinte:

$$\begin{aligned}(q_0, bababab) &\vdash_M (q_1, ababab)\\ &\vdash_M (q_3, babab)\\ &\vdash_M (q_4, abab)\\ &\vdash_M (q_4, bab)\\ &\vdash_M (q_4, ab)\\ &\vdash_M (q_4, b)\\ &\vdash_M (q_4, \varepsilon)\end{aligned}$$

Como uma cadeia é aceita por um autômato finito não-determinístico, se, e somente se, houver ao menos uma seqüência de movimentos conduzindo a um estado final, segue-se que $bababab \in L(M)$. ✧

Exemplo 2.2.2: Seja Σ o alfabeto $\{a_1, \ldots, a_n\}$, contendo n símbolos, onde $n \geq 2$, e considere a seguinte linguagem:

$$L = \{w: \text{ há um símbolo } a_i \in \Sigma \text{ que } \textit{não} \text{ aparece em } w\}.$$

Isto significa que, L contém todas as cadeias em Σ^* em que não ocorrem todos os símbolos em Σ. Por exemplo, se $n = 3$, então $\varepsilon, a_1, a_2, a_1 a_1 a_3 a_1 \in L$, mas $a_3 a_1 a_3 a_1 a_2 \notin L$.

É relativamente fácil projetar um autômato finito não-determinístico $M=(K, \Sigma, \Delta, s, F)$ que reconheça essa linguagem um tanto sofisticada. Aqui K contém $n + 1$ estados $K = \{s, q_1, q_2, \ldots, q_n\}$, todos os quais são finais ($F = K$). Δ tem dois tipos de transições (veja na Figura 2-8 uma ilustração para o caso de $n = 3$). As *transições iniciais* são aquelas da forma (s, ε, q_i) para todos os i, $1 \leq i \leq n$ e todas as transições *principais* são triplas da forma (q_i, a_j, q_i), onde $i \neq j$. Isso completa a lista de transições em Δ.

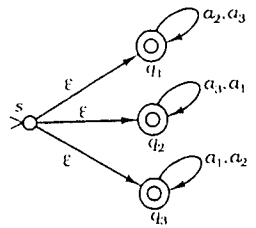


Figura 2-8

Intuitivamente, M inicia sua operação sobre uma dada entrada, *adivinhando um símbolo que esteja ausente* na cadeia de entrada, e transitando para o estado correspondente. Se o símbolo selecionado for a_i, então o estado q_i será o alvo dessa transição. Nesse estado, o autômato verifica que, de fato, o símbolo adivinhado não ocorre na cadeia. Se isso for verdade, ocorre uma aceitação da cadeia de entrada. Esse autômato ilustra muito bem a notável força dos dispositivos não-determinísticos: eles podem *adivinhar e acertar sempre*, já que uma computação bem-sucedida é tudo o que se requer para a aceitação. Como veremos mais adiante, na Seção 2.5, qualquer autômato finito determinístico que reconheça a mesma linguagem deverá ser muito mais complexo. ✧

Um autômato finito determinístico é apenas um caso particular de autômato finito não-determinístico: em um autômato finito determinístico, a relação de transição Δ é uma função de $K \times \Sigma$ para K. Em outras palavras, um autômato finito não-determinístico $(K, \Sigma, \delta, s, F)$ será determinístico se e somente se não exibir transições da forma (q, ε, p) em Δ, e, para cada $q \in K$ e $a \in \Sigma$, houver um e um só $p \in K$ tal que $(q, a, p) \in \Delta$. É, portanto, evidente que a classe das linguagens aceitas por autômatos determinísticos é um *subconjunto* da classe de linguagens aceitas por autômatos não-determinísticos. Um pouco surpreendente é que essas classes são, de fato, *iguais*. Apesar da aparente potência e generalidade oferecidas pelos autômatos não-determinísticos, eles não são mais poderosos do que os determinísticos em termos das linguagens que aceitam: um autômato finito não-determinístico sempre pode ser convertido em um autômato determinístico *equivalente*.

Formalmente, dizemos que dois autômatos finitos M_1 e M_2 (determinístico ou não-determinístico) são **equivalentes** se e somente se $L(M_1) = L(M_2)$. Assim, dois autômatos são considerados equivalentes se aceitarem a mesma linguagem, mesmo que para isso "utilizem métodos diferentes". Por exemplo, os três autômatos nas Figuras 2-4, 2-5 e 2-6 são equivalentes.

Teorema 2.2.1: Para cada autômato finito não-determinístico, há um autômato finito determinístico equivalente.

Prova: Seja $M = (K, \Sigma, \Delta, s, F)$ um autômato finito não-determinístico. Devemos construir um autômato finito determinístico $M' = (K', \Sigma, \delta, s', F')$ equivalente a M. A idéia-chave é visualizar um autômato finito não-determinístico como se ele ocupasse, em qualquer momento, não um simples estado mas o *conjunto* de todos os estados que podem ser alcançados a partir do estado inicial por meio da parte da cadeia de entrada consumida até então. Portanto, se M tiver cinco estados $\{q_0, \ldots, q_4\}$, e após a leitura de uma certa cadeia de entrada, atingir o estado q_0, q_2 ou q_3, mas não q_1 ou q_4, seu estado poderia ser considerado como sendo o *conjunto* $\{q_0, q_2, q_3\}$, em vez de algum membro não-determinado desse conjunto. Se o próximo símbolo de entrada pudesse levar M de q_0 para q_1 ou q_2, de q_2 para q_0 e de q_3 para q_2, então o próximo estado de M poderia ser considerado como sendo o conjunto $\{q_0, q_1, q_2\}$.

A construção a seguir formaliza essa idéia. O conjunto de estados de M' será $K' = 2^K$, o conjunto-potência do conjunto original de estados de M. O conjunto de estados finais de M' consistirá em todos os subconjuntos de K que contêm, pelo menos, um estado final de M. A definição da função de transição de M' será ligeiramente mais complexa. A idéia básica é a de que uma transição de M', ao ler um símbolo de entrada $a \in \Sigma$, simula uma transição de M em resposta à entrada do símbolo a, *possivelmente seguida por qualquer número* de ε-transições em vazio M. Para formalizar essa idéia, precisamos de uma definição especial.

Para qualquer estado $q \in K$, seja $E(q)$ o conjunto de todos os estados de M alcançáveis a partir do estado q, sem a leitura de qualquer símbolo de entrada, isto é,

$$E(q) = \{p \in K: (q,\varepsilon) \vdash^*_M (p,\varepsilon)\}.$$

Em outros termos, $E(q)$ é o *fechamento* do conjunto $\{q\}$ sob a relação

$$\{(p, r): \text{existe uma transição } (p, \varepsilon, r) \in \Delta\}.$$

Portanto, $E(q)$ pode ser *computado* pelo seguinte algoritmo:

Inicialmente faça $E(q) := \{q\}$;
Enquanto existir uma transição $(p, \varepsilon, r) \in \Delta$ com $p \in E(q)$
e $r \notin E(q)$ faça: $E(q) := E(q) \cup \{r\}$.

Esse algoritmo é uma especialização de nosso algoritmo geral para o cálculo de fechamentos (lembre-se do último algoritmo da Seção 1.6) para a situação em questão. Garante-se que ele terminará após, no máximo, $|K|$ iterações, porque cada execução da iteração adiciona mais um estado a $E(q)$, e há, no máximo, $|K|$ estados a serem adicionados. Veremos diversas instâncias desse tipo de algoritmo de fechamento mais adiante.

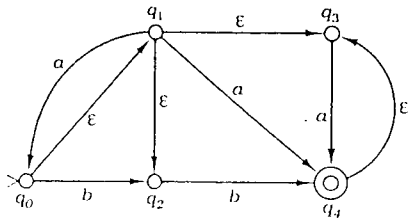


Figura 2-9

Exemplo 2.2.3: No autômato da Figura 2-9, temos $E(q_0)=\{q_0, q_1, q_2, q_3\}$, $E(q_1)=\{q_1, q_2, q_3\}$ e $E(q_2) = \{q_2\}$. ✧

Agora estamos prontos para definir formalmente o autômato determinístico $M'=(K', \Sigma, \delta, s', F')$ que é equivalente a M. Em particular,

$$K' = 2^K,$$
$$s' = E(s),$$
$$F' = \{Q \subseteq K: Q \cap F \neq \emptyset\},$$

e, para cada $Q \subseteq K$ e cada símbolo $a \in \Sigma$, defina

$$\delta(Q, a) = \bigcup \{E(p): p \in K, \text{ e } (q, a, p) \in \Delta \text{ para algum } q \in Q\}.$$

Isto é, $\delta(Q, a)$ é considerado como o conjunto de todos os estados de M para os quais M pode ir lendo a entrada a (e possivelmente executando várias transições em vazio). Por exemplo, se M é o autômato da Figura 2-9, então $s' = \{q_0, q_1, q_2, q_3\}$. Como as únicas transições de q_1, para a entrada a, são (q_1, a, q_0) e (q_1, a, q_4), segue-se que $\delta(\{q_1\}, a) = E(q_0) \cup E(q_4) = \{q_0, q_1, q_2, q_3, q_4\}$.

Resta mostrar que M' é determinístico e equivalente a M. A demonstração de que M' é determinístico é simples e direta: simplesmente notamos que δ tem valor único e bem definido para todos os $Q \in K'$ e $a \in \Sigma$ por construção. (Note que se $\delta(Q, a) = \emptyset$ para algum $Q \in K'$ e $a \in \Sigma$ não significa que δ não tenha sido bem definida; $\emptyset$ é um membro de K'.)

Agora lançamos a hipótese de que, para qualquer cadeia $w \in \Sigma^*$ e quaisquer estados $p, q \in K$,

$$(q, w) \vdash_M^* (p, \varepsilon) \text{ se e somente se } (E(q), w) \vdash_{M'}^* (P, \varepsilon)$$

para algum conjunto P contendo p. A partir disto, o teorema decorre facilmente: para mostrar que M e M' são equivalentes, considere qualquer cadeia $w \in \Sigma^*$. Então $w \in L(M)$ se e somente se $(s, w) \vdash_M^* (f, \varepsilon)$ para algum $f \in F$ (por definição) se e somente se $(E(s), w) \vdash_{M'}^* (Q, \varepsilon)$ para algum Q contendo f (pela hipótese formulada acima); em outras palavras, se e somente se $(s, w) \vdash_{M'}^* (Q, \varepsilon)$ para algum $Q \in F'$. A última condição é a definição de $w \in L(M')$.

Provamos a hipótese por indução em $|w|$.

Base da indução. Para $|w| = 0$ – isto é, para $w = \varepsilon$ –, devemos mostrar que $(q, \varepsilon) \vdash_M^* (p, \varepsilon)$, se, e somente se, $(E(q), \varepsilon) \vdash_{M'}^* (P, \varepsilon)$ para algum conjunto P contendo p. A primeira afirmação corresponde a dizer que $p \in E(q)$. Como M' é determinístico, a segunda afirmação equivale a dizer que $P = E(q)$ e P contém p, ou seja, $p \in E(q)$. Isso completa a prova da base da indução.

Hipótese da indução. Suponha que a hipótese seja verdadeira para todas as cadeias w de comprimento k ou menor, para algum $k \geq 0$.

Passo da indução. Provemos a hipótese para qualquer cadeia w de comprimento $k + 1$. Seja $w = va$, onde $a \in \Sigma$ e $v \in \Sigma^*$.

Para o sentido *somente se*, seja $(q, w) \vdash_M^* (p, \varepsilon)$. Então, há estados r_1 e r_2 tais que

$$(q, w) \vdash_M^* (r_1, a) \vdash_M (r_2, \varepsilon) \vdash_M^* (p, \varepsilon).$$

Isto significa que M alcança o estado p a partir do estado q mediante um certo número de transições, durante os quais a entrada v é lida, seguida por uma transição durante a qual a entrada a é lida, seguida por mais alguns movimentos durante os quais nenhuma entrada é lida. Agora $(q, va) \vdash_M^* (r_1, a)$ é equivalente a $(q, v) \vdash_M^* (r_1, \varepsilon)$ e, como $|v| = k$, pela hipótese da indução, $(E(q), v) \vdash_{M'}^* (R_1, \varepsilon)$ para algum conjunto R_1 contendo r_1. Como $(r_1, a) \vdash_M (r_2, \varepsilon)$, há uma tripla $(r_1, a, r_2) \in \Delta$ e, portanto, pela construção de M', $E(r_2) \subseteq \delta(R_1, a)$. Mas como $(r_2, \varepsilon) \vdash_M^* (p, \varepsilon)$, segue-se que $p \in E(r_2)$ e, portanto, $p \in \delta(R_1, a)$. Assim $(R_1, a) \vdash_{M'} (P, \varepsilon)$ para algum P contendo p e, desse modo, $(E(q), va) \vdash_{M'}^* (R_1, a) \vdash_{M'} (P, \varepsilon)$.

Para provar o outro sentido, suponha que $(E(q), va) \vdash_{M'}^* (R_1, a) \vdash_{M'} (P, \varepsilon)$ para algum P contendo p e algum R_1, tal que $\delta(R_1, a) = P$. Agora, pela definição de δ, $\delta(R_1, a)$ é a união de todos os conjuntos $E(r_2)$, onde, para algum estado $r_1 \in R_1$, (r_1, a, r_2) é uma transição de M. Como $p \in P = \delta(R_1, a)$, existe algum particular r_2, tal que $p \in E(r_2)$ e, para algum $r_1 \in R_1$, (r_1, a, r_2) é uma transição de M. Então $(r_2, \varepsilon) \vdash_M^* (p, \varepsilon)$ pela definição de $E(r_2)$. Além disso, pela hipótese de indução, $(q, v) \vdash_M^* (r_1, \varepsilon)$ e, portanto, $(q, va) \vdash_M^* (r_1, a) \vdash_M (r_2, \varepsilon) \vdash_M^* (p, \varepsilon)$.

Isso completa a prova da hipótese e do teorema. ■

Exemplo 2.2.4: A prova do Teorema 2.2.1 tem a salutar propriedade de ser *construtiva*, no sentido de que proporciona na realidade um *algoritmo* para construir, a partir de qualquer autômato finito não-determinístico M, um equivalente determinístico M'.

Vamos, então, aplicar esse algoritmo ao autômato não-determinístico da Figura 2-9. Como M tem 5 estados, M' terá $2^5 = 32$ estados. Entretanto, desses estados serão relevantes, para a operação de M', apenas aqueles que podem ser alcançados a partir do estado s' como resultado da leitura de alguma cadeia de entrada. Obviamente, qualquer estado em K' que não seja alcançável a partir de s' é irrelevante para a operação de M' e para a linguagem por ele aceita. Devemos construir essa parte alcançável de M', partindo de s' e introduzindo um novo estado, somente quando ele for igual a $\delta(q, a)$ para algum estado $q \in K'$ já existente e algum $a \in \Sigma$.

Definimos $E(q)$ para cada estado q de M. Como $s' = E(q_0) = \{q_0, q_1, q_2, q_3\}$,

$$(q_1, a, q_0), (q_1, a, q_4) \text{ e } (q_3, a, q_4)$$

são todas as transições (q, a, p) para algum $q \in s'$. Segue-se que

$$\delta(s', a) = E(q_0) \cup E(q_4) = \{q_0, q_1, q_2, q_3, q_4\}.$$

Analogamente,

$$(q_0, b, q_2) \text{ e } (q_2, b, q_4)$$

são todas as transições da forma (q, b, p) para algum $q \in s'$. Assim,

$$\delta(s', b) = E(q_2) \cup E(q_4) = \{q_2, q_3, q_4\}.$$

Repetindo esse cálculo para os estados recém-introduzidos, temos o seguinte:

$$\begin{aligned}
\delta(\{q_0, q_1, q_2, q_3, q_4\}, a) &= \{q_0, q_1, q_2, q_3, q_4\}.\\
\delta(\{q_0, q_1, q_2, q_3, q_4\}, b) &= \{q_2, q_3, q_4\},\\
\delta(\{q_2, q_3, q_4\}, a) &= E(q_4) = \{q_3, q_4\},\\
\delta(\{q_2, q_3, q_4\}, b) &= E(q_4) = \{q_3, q_4\}.
\end{aligned}$$

Em seguida,

$$\begin{aligned}
\delta(\{q_3, q_4\}, a) &= E(q_4) = \{q_3, q_4\},\\
\delta(\{q_3, q_4\}, b) &= \emptyset,
\end{aligned}$$

e, por fim

$$\delta'(\emptyset, a) = \delta'(\emptyset, b) = \emptyset.$$

A parte relevante de M' é ilustrada na Figura 2.10. F', o conjunto de estados finais, contém cada um dos conjuntos de estados dos quais q_4 é um membro, já que q_4 é o único membro de F; assim, na ilustração, os três estados $\{q_0, q_1, q_2, q_3, q_4\}$, $\{q_2, q_3, q_4\}$ e $\{q_3, q_4\}$ de M' são estados finais. ✧

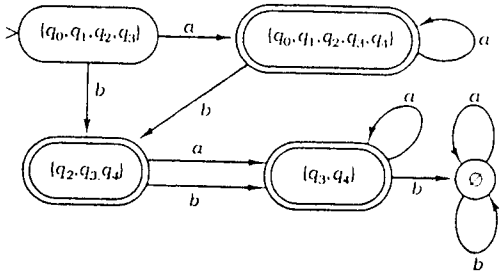


Figura 2-10

Exemplo 2.2.5: Lembre do autômato finito não-determinístico com $n + 1$ estados do Exemplo 2.2.2, com $\Sigma = \{a_1,\ldots,a_n\}$, que aceita a linguagem $L = \{w \in \Sigma^*$: há um símbolo $a_i \in \Sigma$ que não ocorre em $w\}$. Intuitivamente, não há como um autômato finito determinístico aceitar a mesma linguagem dispondo de tão poucos estados.

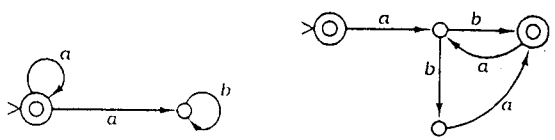
De fato, a construção, nesse caso, é exponencial. O autômato determinístico M' equivalente tem como estado inicial o conjunto $s' = E(s) = \{s, q_1, q_2,\ldots, q_n\}$. Mesmo neste caso, M' apresenta diversos estados irrelevantes – de fato, metade dos estados em 2^K são irrelevantes. Por exemplo, o estado $\{s\}$ não pode ser alcançado a partir de s' nem de qualquer estado que contenha algum q_i mas não s. Lamentavelmente, esses são todos os estados irrelevantes: como se pode notar, são todos os demais 2^n estados de K' – o que significa dizer que todos os estados na forma $\{s\} \cup Q$ para algum subconjunto Q não-vazio de $\{q_1,\ldots, q_n\}$ – são alcançáveis a partir de s'.

Poder-se-ia esperar, naturalmente, que outros tipos de otimizações fossem capazes de reduzir o número de estados em M'. A Seção 2.5 contém um estudo sistemático dessas otimizações. Infelizmente, conclui-se dessa análise que, no presente caso do autômato M', o número exponencial de estados em M' é exatamente o mínimo possível. ✧

Problemas para a Seção 2.2

2.2.1 (a) Quais das seguintes cadeias são aceitas pelo autômato finito não-determinístico mostrado à esquerda na figura abaixo?
(i) a
(ii) aa
(iii) aab
(iv) ε

(b) Repita para as seguintes cadeias, e para autômato finito não-determinístico à direita:



(i) ε
(ii) ab
(iii) $abab$
(iv) aba
(v) $abaa$

2.2.2 Escreva expressões regulares que representem as linguagens aceitas pelos autômatos finitos não-determinísticos do Problema 2.2.1.

2.2.3 Desenhe diagramas de estados que representem os autômatos finitos não-determinísticos que reconhecem as seguintes linguagens.
(a) $(ab)^*(ba)^* \cup aa^*$
(b) $((ab \cup aab)^*a^*)^*$
(c) $((a^*b^*a^*)^*b)^*$
(d) $(ba \cup b)^* \cup (bb \cup a)^*$

2.2.4 Algum autores definem um autômato finito não-determinístico como sendo uma quíntupla $(K, \Sigma, \Delta, S, F)$, onde K, Σ, Δ e F têm o significado habitual e S é um conjunto finito de estados iniciais, do mesmo modo que F é um conjunto finito de estados finais. O autômato pode, não-deterministamente, começar a operar a partir de qualquer desses estados iniciais.
(a) Mostre que a linguagem $L \subseteq \{a_1,\ldots, a_n\}^*$, que consiste de todas as cadeias em que falta pelo menos um desses símbolos (lembre-se do Exemplo 2.2.2) seria aceita por um tal autômato, com n estados $q_1,\ldots, q_n$, sendo todos eles tanto finais como iniciais, e obedecendo à relação de transição $\Delta = \{(q_i, a_j, q_i)$: $i \neq j\}$.
(b) Explique por que essa definição não é mais geral que a nossa sob qualquer aspecto significativo.

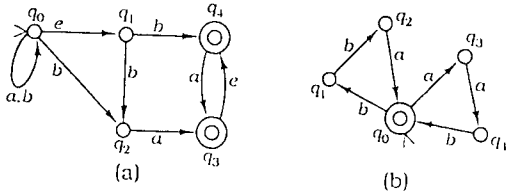
2.2.5 Por meio de quais seqüências de passos, diferentes das apresentadas no Exemplo 2.2.1, o autômato finito não-determinístico da Figura 2-7 pode aceitar a entrada *bababab*?

2.2.6 (a) Encontre um autômato finito não-determinístico simples que aceite $(ab \cup aab \cup aba)^*$.
(b) Converta o autômato finito não-determinístico do item (a) em um autômato finito determinístico pelo método apresentado na Seção 2.2.
(c) Procure entender como opera a máquina construída na Parte (b). Você pode encontrar uma máquina determinística equivalente com menos estados?

2.2.7 Repita o Problema 2.2.6 para a linguagem $(a \cup b)^*aabab$.

2.2.8 Repita o Problema 2.2.6 para a linguagem $(a \cup b)^*a\,(a \cup b)\,(a \cup b)\,(a \cup b)\,(a \cup b)$.

2.2.9 Construa autômatos finitos determinísticos equivalentes aos autômatos não-determinísticos apresentados a seguir.



2.2.10 Descreva rigorosamente o que ocorre quando o método apresentado nesta seção é aplicado a um autômato finito que já é determinístico.

2.3 AUTÔMATOS FINITOS E EXPRESSÕES REGULARES

O principal resultado da seção anterior foi que a classe das linguagens aceitas por autômatos finitos não se altera mesmo que uma nova e aparentemente poderosa característica – o não-determinismo – esteja incorporado. Isso sugere que a classe de linguagens aceitas por autômatos finitos apresenta uma espécie de *estabilidade*: dois diferentes esquemas, um aparentemente mais poderoso que o outro, acabam definindo a mesma classe de linguagens. Nesta seção, provaremos outra importante característica dessa classe de linguagens, evidência adicional do quão notavelmente estável ela é: a classe de linguagens aceitas por autômatos finitos, determinísticos ou não-determinísticos, é a mesma que a classe das *linguagens regulares* – aquelas que podem ser descritas pelas expressões regulares (lembre-se da discussão na Seção 1.8).

A classe das linguagens regulares foi apresentada como sendo o fechamento de certas linguagens finitas sob as operações de união, concatenação e estrela de Kleene. Devemos, portanto, iniciar provando propriedades similares de fechamento para a classe das linguagens aceitas por autômatos finitos:

Teorema 2.3.1: *A classe de linguagens aceitas por autômatos finitos é fechada em relação às operações de*

(a) união;
(b) concatenação;
(c) estrela de Kleene;
(d) complementação;
(e) intersecção.

Prova: Em cada caso, mostramos como, dados dois autômatos, M_1 e M_2 (somente M_1, nos casos da estrela de Kleene e da complementação), é possível construir um autômato M que reconhece a linguagem correspondente.

(a) *União.* Sejam $M_1 = (K_1, \Sigma, \Delta_1, s_1, F_1)$ e $M_2 = (K_2, \Sigma, \Delta_2, s_2, F_2)$ dois autômatos finitos não-determinísticos; devemos construir *um* autômato finito não-determinístico M, tal que $L(M) = L(M_1) \cup L(M_2)$. A construção de M, relativamente simples e intuitivamente clara, é ilustrada na Figura 2-11. Basicamente, M se serve do não-determinismo para adivinhar se a cadeia de entrada está em $L(M_1)$ ou em $L(M_2)$ e, então, processa a cadeia exatamente como o autômato correspondente processaria; segue-se que $L(M)=L(M_1) \cup L(M_2)$. Vamos apresentar os detalhes e provas formais para esse caso.

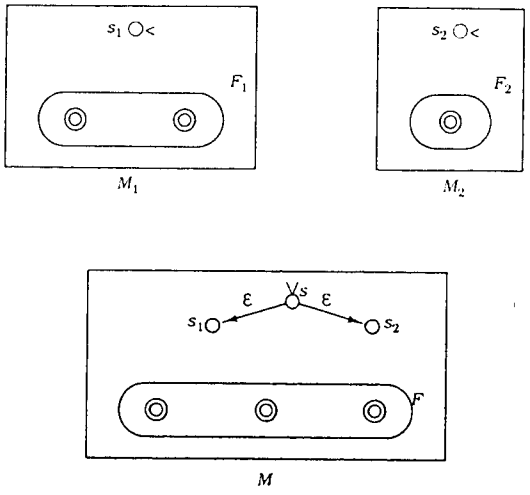


Figura 2-11

Sem perda de generalidade, podemos admitir que K_1 e K_2 sejam conjuntos disjuntos. Nesse caso, o autômato finito M, que reconhece $L(M_1) \cup L(M_2)$, é definido como segue (veja a Figura 2-11): $M = (K, \Sigma, \Delta, s, F)$, onde s é um novo estado que não está em K_1 ou K_2.

$$K = K_1 \cup K_2 \cup \{s\},$$
$$F = F_1 \cup F_2,$$
$$\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, \varepsilon, s_1), (s, \varepsilon, s_2)\}.$$

Assim, M começa qualquer computação escolhendo não-deterministicamente entrar no estado inicial de M_1 ou no estado inicial de M_2, e daí em diante, M simula ou M_1 ou M_2. Formalmente, se $w \in \Sigma^*$, então $(s, w) \vdash^*_M (q, \varepsilon)$ para algum $q \in F$, se e somente se $(s_1, w) \vdash^*_{M_1} (q, \varepsilon)$ para algum $q \in F_1$ *ou* $(s_2, w) \vdash^*_{M_2} (q, \varepsilon)$ para algum $q \in F_2$. Portanto, M reconhece w, se e somente se M_1 reconhece w ou M_2 reconhece w, e portanto, $L(M) = L(M_1) \cup L(M_2)$.
(b) *Concatenação.* Novamente, suponha que M_1 e M_2 sejam autômatos finitos não-determinísticos; construamos um autômato finito não-determinístico M, tal que $L(M) = L(M_1) \circ L(M_2)$. A construção é mostrada esquematicamente na Figura 2-12: M agora opera simulando M_1 por alguns momentos e então "salta"[*] não-deterministamente de um estado final de M_1 para o estado inicial de M_2. Daí em diante, M simula M_2. Omitimos a definição e prova formais.

[*] N. de R. T. Note que, na Figura 2-12, os estados finais de M_1 deixam de ser finais quando utilizados em M, uma vez que M reconhece apenas as sentenças de $L(M_1) \circ L(M_2)$. Notar também que os estados finais de M são os mesmos que os de M_2.

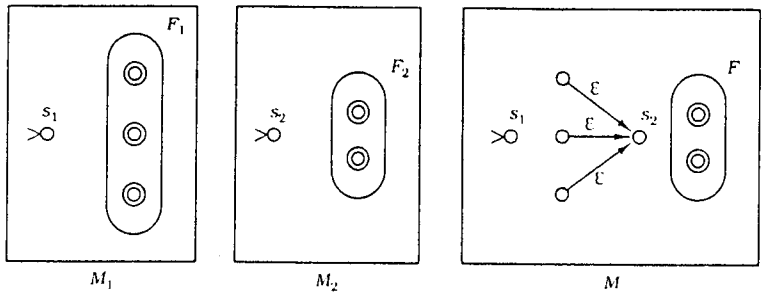


Figura 2-12

(c) *Estrela de Kleene.* Seja M_1 um autômato finito não-determinístico; construamos um autômato finito não-determinístico M, tal que $L(M) = L(M_1)^*$. A técnica utilizada é similar à que foi empregada no caso da concatenação, e é ilustrada na Figura 2-13. M compõe-se dos estados de M_1 e de todas as transições de M_1; qualquer estado final de M_1 é um estado final de M. Além disso, M tem um novo estado inicial s. Esse novo estado inicial também é final, assim, a cadeia vazia ε é aceita. De s há uma transição em vazio para o estado inicial s_1 de M_1, de tal maneira que a operação de M_1 pode ser iniciada logo após M ter sido iniciado no estado s. Por fim, outras transições em vazio são adicionadas a partir de cada estado final de M_1 de volta para s_1; desse modo, uma vez que uma cadeia de $L(M_1)$ tenha sido lida, a computação pode prosseguir a partir do estado inicial de M_1.

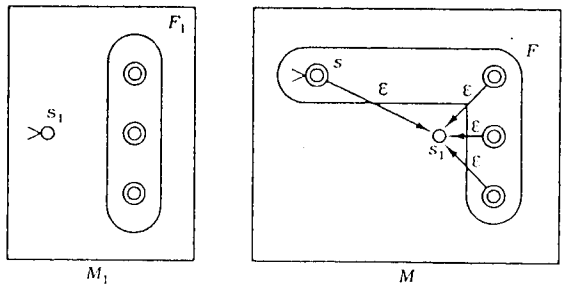


Figura 2-13

d) Complementação. Seja $M = (K, \Sigma, \delta, s, F)$ *um autômato finito determinístico.* Então, a linguagem complementar $\overline{L} = \Sigma^* - L(M)$ é aceita pelo autômato finito determinístico $\overline{M} = (K, \Sigma, \delta, s, K - F)$. Isto é, $\overline{M}$ é idêntico a M, exceto que estados finais e não-finais são intercambiados.

e) Intersecção. Basta levar em conta que†

$$L_1 \cap L_2 = \Sigma^* - ((\Sigma^* - L_1) \cup (\Sigma^* - L_2)),$$

e, assim, o fechamento em relação à intersecção pode ser obtido a partir do fechamento sob união e complementação (veja itens (a) e (d) acima). ■

†N. de R. T. Usando o Teorema de De Morgan.

Chegamos então à conclusão mais importante desta seção, a identificação da equivalência de duas importantes técnicas para a especificação finita de linguagens – um gerador e um reconhecedor de linguagem:

Teorema 2.3.2: *Uma linguagem é regular se e somente se ela for aceita por um autômato finito.*

Prova: [*Somente se*] Lembre-se de que a classe de linguagens regulares é a menor classe de linguagens contendo o conjunto vazio $\emptyset$ e os conjuntos unitários a, onde a é um símbolo e é fechado em relação às operações de união, concatenação e estrela de Kleene. É evidente (veja a Figura 2-14) que o conjunto vazio e todos os conjuntos unitários são de fato aceitos por autômatos finitos. Pelo Teorema 2.3.1, as linguagens representadas por autômatos finitos são fechadas em relação à união, concatenação e estrela de Kleene. Portanto, todas as linguagens regulares são aceitas por autômatos finitos.

Exemplo 2.3.1: Considere a expressão regular $(ab \cup aab)^*$. Um autômato finito não-determinístico que aceita a linguagem denotada por essa expressão regular pode ser criado utilizando-se os métodos utilizado para demonstrar as várias partes do Teorema 2.3.1, como ilustrado na Figura 2-14. ✧

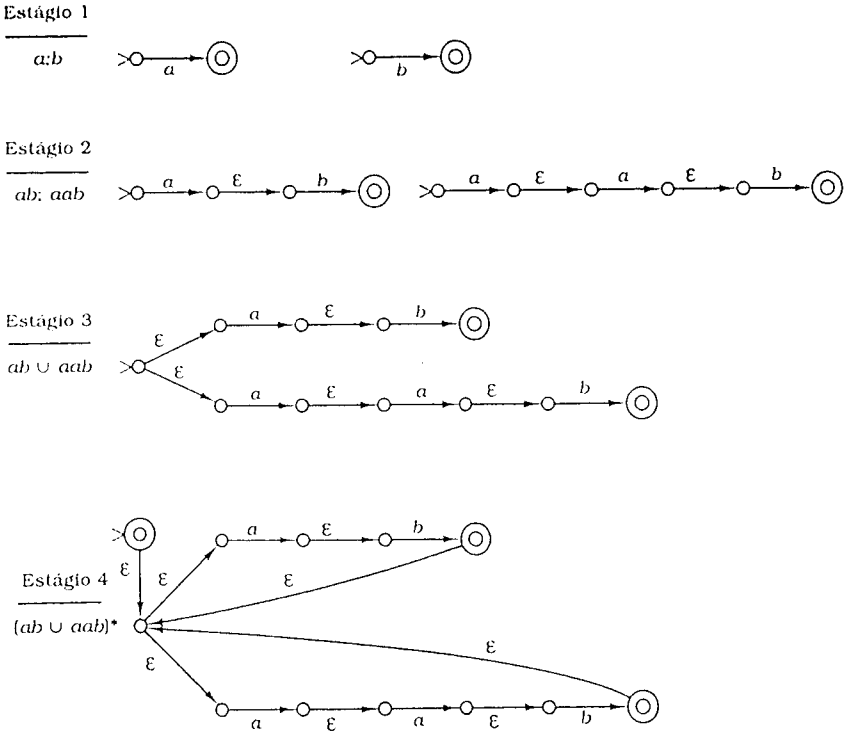


Figura 2-14

[*Se*] Seja $M = (K, \Sigma, \Delta, s, F)$ um autômato finito (não necessariamente determinístico). Devemos construir uma expressão regular R tal que $L(R) = L(M)$. Devemos representar $L(M)$ como a união de um número finito (eventualmente grande) de linguagens simples. Seja $K = \{q_1, \ldots q_n\}$, e $s = q_1$. Para $i, j = 1, \ldots, n$ e $k = 0, \ldots, n$, definimos $R(i, j, k)$ como o conjunto de todas as cadeias em Σ^* que podem levar M do estado q_i para o estado q_j sem passar por qualquer estado *intermediário* cujo número seja $k + 1$ ou maior – exceto para os estados terminais q_i e q_j, os quais podem receber números maiores que k. Em outras palavras, $R(i, j, k)$ é o conjunto de cadeias formado por todos os caminhos de q_i a q_j *de ordem* k (lembre-se da técnica similar empregada na computação do fechamento transitivo-reflexivo de uma relação, na Seção 1.6, na qual foram considerados caminhos com vértices intermediários cujos números eram progressivamente maiores). Quando $k = n$, resulta:

$$R(i, j, n) = \{(w \in \Sigma^* : (q_i, x) \vdash^*_M (q_j, \varepsilon)\}.$$

Portanto,

$$L(M) = \cup\{R(1, j, n) : q_j \in F\}.$$

O ponto crucial é que todos esses conjuntos $R(i, j, k)$ são regulares, e portanto $L(M)$ também o será.

A prova de que cada $R(i, j, k)$ é regular dá-se por indução sobre k. Para $k = 0$, ou $R(i, j, 0)$ é $\{a \in \Sigma \cup \{\varepsilon\} : (q_i, a, q_j) \in \Delta\}$ caso $i \neq j$, ou então $\{\varepsilon\} \cup \{a \in \Sigma \cup \{\varepsilon\} : (q_i, a, q_j) \in \Delta\}$, caso $i = j$. Cada um desses conjuntos é finito e, portanto, regular.

Para o passo indutivo, consideremos que, para todo i, j, $R(i, j, k - 1)$ tenham sido definidos como linguagens regulares. Então, cada conjunto $R(i, j, k)$ poderá ser definido combinando-se linguagens regulares, previamente especificadas, por meio das operações regulares de união, estrela de Kleene e concatenação, como segue:

$$R(i, j, k) = R(i, j, k - 1) \cup R(i, k, k - 1)R(k, k, k - 1)^* R(k, j, k - 1).$$

Essa equação afirma que, para ir de q_i a q_j sem passar por um estado que apresente numeração maior que k, M pode:

(1) ir de q_i para q_j sem passar por um estado com numeração maior que $k - 1$; ou
(2) (*a*) ir de q_i a q_k; então (b) ir de q_k para q_k zero ou mais vezes; e depois (c) ir de q_k a q_j; sem passar em caso algum por qualquer estado intermediário que tenha um número maior que $k - 1$.

Portanto, a linguagem $R(i, j, k)$ é regular para todos os i, j, k, o que completa a indução. ■

Exemplo 2.3.2: Vamos construir uma expressão regular para a linguagem aceita pelo autômato finito determinístico da Figura 2-15. Esse autômato reconhece a linguagem

$$\{w \in \{a, b\}^* : w \text{ tem } 3k + 1 \text{ } b\text{'s para algum } k \in \mathbf{N}\}.$$

Realizar explicitamente a construção da prova da parte [*se*] do teorema pode ser muito tedioso (nesse simples caso, trinta e seis expressões regulares teriam de ser construídas!). As coisas ficam consideravelmente simplificadas se o autômato não-determinístico M apresentar duas propriedades simples:

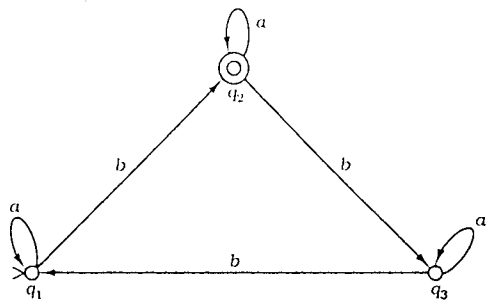


Figura 2-15

(a) O autômato possui um único estado final, ou seja $F = \{f\}$.
(b) Além disso, se $(q, u, p) \in \Delta$, então $q \neq f$ e $p \neq s$; isto é, não há transições entrando no estado inicial, nem emanando do estado final.

Essa "forma especial" não representa uma perda de generalidade, porque sempre podemos adicionar a qualquer autômato M um novo estado inicial s e um novo estado final f, junto com transições em vazio de s para o estado inicial de M e de todos os estados finais de M para f (veja a Figura 2-16(a), onde o autômato da Figura 2-15 é colocado nesse formato especial). Numere agora os estados do autômato $q_1, q_2, \ldots, q_n$, como foi sugerido pela construção, de modo que $s = q_{n-1}$, e $f = q_n$. Obviamente, a expressão regular procurada é $R(n - 1, n, n)$.

Devemos computar primeiro os $R(i, j, 0)$'s. A partir destes, computamos os $R(i, j, 1)$'s, e assim por diante, como sugerido na demonstração. Em cada estágio, representamos cada $R(i, j, k)$ como um rótulo em uma seta indo do estado q_i para o estado q_j. Omitimos as setas rotuladas por $\emptyset$ e os laços rotulados $\{\varepsilon\}$. Com essa convenção, o autômato inicial representa os valores corretos dos $R(i, j, 0)$'s – veja a Figura 2-16 (a). (Isso é assim porque, em nosso autômato inicial, ocorre que, para cada par de estados (q_i, q_j), há no máximo uma transição na forma (q_i, u, q_j) em Δ. Em outro autômato, podemos ter de combinar, por união, todas as transições de q_i para q_j, como está sugerido na demonstração).

Agora computamos os $R(i, j, 1)$'s; eles são mostrados na Figura 2-16(b). Note imediatamente que o estado q_1 não precisa ser considerado no resto da construção; todas as cadeias que levam M à situação de aceitação passando pelo estado q_1 foram consideradas e levadas em conta nos $R(i, j, 1)$'s. Podemos dizer que o estado q_1 foi *eliminado*. Neste caso, transformamos o autômato finito da Figura 2-16(a) em um *autômato finito generalizado* equivalente, com transições que podem ser chamadas não somente por símbolos em Σ ou ε, mas por *expressões regulares* completas. O autômato finito generalizado resultante tem um estado a menos que o original, uma vez que q_1 foi eliminado.

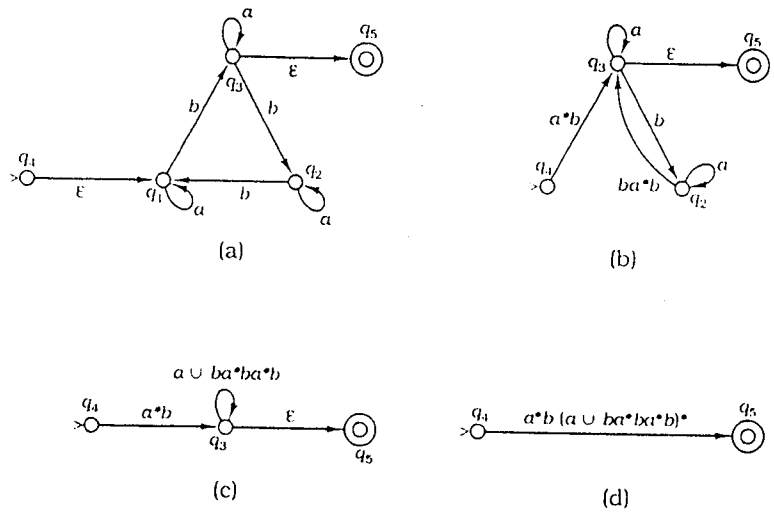


Figura 2-16

Vamos examinar cuidadosamente o que acarreta, no caso geral, a eliminação de um estado q (veja a Figura 2-17). Para cada par de estados $q_i \neq q$ e $q_j \neq q$, tal que há uma seta rotulada com α de q_i para q, e uma seta rotulada com β de q para q_j, adicionamos uma seta de q_i a q_j chamada $\alpha\gamma^*\beta$, onde γ é o rótulo da seta de q para ele próprio (se não houver tal seta, então $\gamma = \emptyset$ e, portanto, $\gamma^* = \{\varepsilon\}$, assim o rótulo torna-se $\alpha\beta$). Se já houvesse uma seta de q_i para q_j, rotulada com δ, então a nova seta será rotulada com $\delta \cup \alpha\gamma^*\beta$.

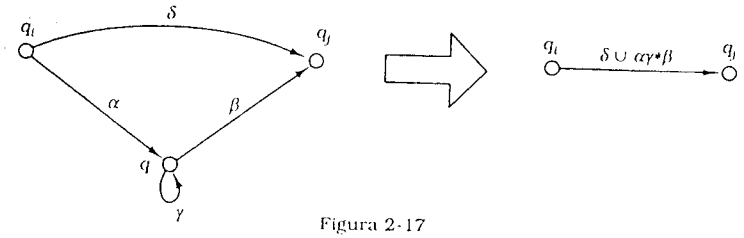


Figura 2-17

Continuando assim, eliminamos o estado q_2 para obter os $R(i, j, 2)$'s na Figura 2-17 (c) e, por fim, eliminamos q_3. Agora excluímos todos os estados, exceto o inicial e os finais, e o autômato generalizado fica reduzido a uma simples transição do estado inicial para o estado final. Podemos então ler a expressão regular que representa a linguagem definida por M como o rótulo dessa transição:

$$R = R(4, 5, 5) = R(4, 5, 3) = a^*b(ba^*ba^*b \cup a)^*,$$

o qual, de fato, é $\{w \in \{a, b\}^*: w \text{ tem } 3k + 1 \text{ } b\text{'s para algum } k \in \mathbb{N}\}$. ✧

Problemas para a Seção 2.3

2.3.1 Na parte (d) da demonstração do Teorema 2.3.1, qual é a razão para insistirmos que M seja determinístico? O que ocorre se permutarmos os estados finais com os não-finais de um autômato finito não-determinístico?

2.3.2 O que daria errado na demonstração da Parte (c) do Teorema 2.3.1 se simplesmente tornássemos s_1 um estado final, e adicionássemos setas de todos os membros de F_1 de volta para s_1 (sem introduzir um novo estado inicial)?

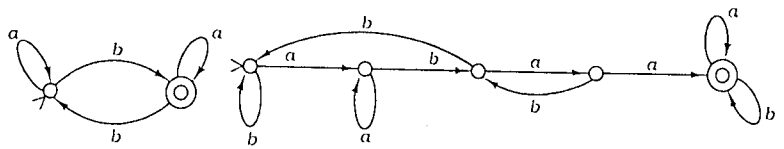
2.3.3 Forneça um método para a construção direta do fechamento relativo à intersecção das linguagens aceitas por autômatos finitos. (*Sugestão*: Considere um autômato cujo conjunto de estados seja o produto cartesiano dos conjuntos de estados dos dois autômatos originais.) Qual das duas construções, a fornecida no texto ou a sugerida neste problema, é mais eficiente quando as duas linguagens são dadas em termos de autômatos finitos *não-determinísticos*?

2.3.4 Utilizando as construções empregadas na demonstração do Teorema 2.3.1, construa autômatos finitos que aceitem as linguagens seguintes:
(a) $a^*(ab \cup ba \cup \varepsilon)b^*$
(b) $((a \cup b)^*(\varepsilon \cup c)^*)^*$
(c) $((ab)^* \cup (bc)^*)ab$

2.3.5 Construa um autômato finito não-determinístico simples que aceite a linguagem $(ab \cup aba)^*a$. Então, aplique a ele a técnica utilizada na Parte (c) da demonstração do Teorema 2.3.1, para obter um autômato finito não-determinístico que aceita $((ab \cup aba)^*a)^*$.

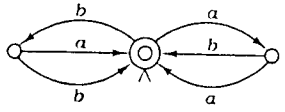
2.3.6 Sejam L, $L' \subseteq \Sigma^*$. Defina as seguintes linguagens:
1. $\text{Pref}(L) = \{w \in \Sigma^*: x = wy \text{ para algum } x \in L, y \in \Sigma^*\}$ (o conjunto de **prefixos** de L).
2. $\text{Suf}(L) = \{w \in \Sigma^*: x = yw \text{ para algum } x \in L, y \in \Sigma^*\}$ (o conjunto de **sufixos** de L).
3. $\text{Subseq}(L) = \{w_1 w_2 \ldots w_k: k \in \mathbb{N}, w_i \in \Sigma^* \text{ para } i = 1, \ldots k, \text{ e há uma cadeia } x = x_0 w_1 x_1 w_2 x_2 \ldots w_k x_k \in L\}$ (o conjunto de **subseqüências** de L).
4. $L/L' = \{w \in \Sigma^*: wx \in L \text{ para algum } x \in L'\}$ (o **quociente direito** de L por L').
5. $\text{Max}(L) = \{w \in L: \text{se } x \neq \varepsilon, \text{ então } wx \notin L\}$.
6. $L^R = \{w^R: w \in L\}$.

Mostre que, se L é aceita por algum autômato finito, então, assim também ocorre com cada uma das seguintes linguagens:
(a) Pref(L)
(b) Suf(L)
(c) Subseq(L)
(d) L/L', onde L' é aceita por algum autômato finito.
(e) L/L', onde L' é *qualquer* linguagem.
(f) Max(L)
(g) L^R

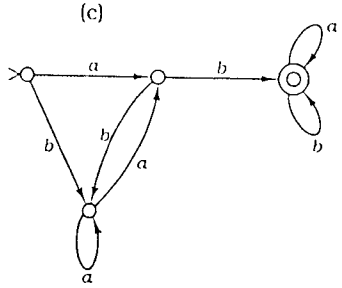


(a)

(c)



(b)



(d)

2.3.7 Aplique a técnica empregada no Exemplo 2.3.2 para obter as expressões regulares correspondentes a cada um dos autômatos finitos acima. Simplifique as expressões resultantes o máximo que você puder.

2.3.8 Para qualquer número natural $n \geq 1$, definir o autômato finito não-determinístico seguinte:
$M_n = (K_n, \Sigma_n, \Delta_n, s_n, F_n)$, com $K_n = \{q_1, q_2, \ldots, q_n\}$, $s_n = q_1$, $F_n = \{q_1\}$, $\Sigma_n = \{a_{ij}: i, j = 1, \ldots, n\}$ e $\Delta_n = \{(q_i, a_{ij}, q_j): i, j = 1, \ldots, n\}$.
(a) Descrever $L(M_n)$ em português.
(b) Escreva uma expressão regular para $L(M_3)$.
(c) Escreva uma expressão regular para $L(M_5)$.
Presume-se que, para todos os polinômios p, exista um n, tal que nenhuma expressão regular para $L(M_n)$ apresenta comprimento menor que $p(n)$ símbolos.

2.3.9 (a) Por analogia com o Problema 2.1.5, defina um **autômato finito não-determinístico de 2 fitas** e a noção de que tal autômato reconhece um particular conjunto de pares ordenados de cadeias.
(b) Mostre que $\{(a^m b, a^n b^p): n, m, p \geq 0 \text{ e } n = m \text{ ou } n = p\}$ é aceito por algum autômato finito não-determinístico de duas fitas.
(c) Veremos (Problema 2.4.9) que autômatos finitos não-determinísticos de duas fitas nem sempre podem ser convertidos em determinísticos. Sendo esse o caso, indique as construções utilizadas na demonstração do Teorema 2.3.1 que podem ser estendidas para demonstrar as propriedades de fechamento apresentadas nos seguintes itens:
(i) O conjunto dos pares de cadeias aceitos por autômatos finitos não-determinístico de duas fitas.
(ii) O conjunto dos pares de cadeias aceitos por autômatos finitos determinísticos de duas fitas.
Explique suas respostas.

2.3.10 A linguagem L é dita **definida**, se houver algum k, tal que, para qualquer cadeia w, o fato de $w \in L$ depende somente dos últimos k símbolos de w.
(a) Reescreva essa definição mais formalmente.
(b) Mostre que toda linguagem definida é aceita por um autômato finito.
(c) Mostre que a classe das linguagens definidas é fechada em relação à união e à complementação.
(d) Dê um exemplo de uma linguagem definida L tal que L^* não seja definida.
(e) Dê um exemplo de linguagens definidas L_1, L_2 tais que $L_1 L_2$ não seja definida.

2.3.11 Sejam Σ e Δ dois alfabetos. Considere uma função h de Σ para Δ^*. Estenda h para uma função de Σ^* para Δ^*, como segue.

$$h(\varepsilon) = \varepsilon.$$
$$h(w) = h(u)h(\sigma) \text{ para qualquer } w, u \in \Sigma^*, \sigma \in \Sigma.$$

Por exemplo, se $\Sigma = \Delta = \{a, b\}$, $h(a) = ab$ e $h(b) = aab$, então

$$\begin{aligned} h(aab) &= h(aa)h(b) \\ &= h(a)h(a)h(b) \\ &= abababaab. \end{aligned}$$

Qualquer função $h: \Sigma^* \mapsto \Delta^*$ definida dessa forma a partir de uma função $h: \Sigma \mapsto \Delta^*$ é chamada de **homomorfismo**.

Seja h um homomorfismo de Σ^* para Δ^*.
(a) Mostre que, se $L \subseteq \Sigma^*$ é aceita por um autômato finito, então, $h[L]$ também o será.
(b) Mostre que, se L é aceita por um autômato finito, então, $\{w \in \Sigma^*: h(w) \in L\}$ também o será. (*Sugestão*: Inicie a partir de um autômato finito determinístico M que aceite L e construa um que, ao ler um símbolo de entrada a, tente simular o que M faria na entrada $h(a)$.)

2.3.12 Transdutores determinísticos de estados finitos foram apresentados no Problema 2.1.4. Mostre que, se L é aceita por um autômato finito e f é computada por um transdutor determinístico de estados finitos, então cada uma das seguintes afirmações é verdadeira.
(a) $f[L]$ é aceita por um autômato finito.
(b) $f^{-1}[L]$ é aceita por um autômato finito.

2.4 LINGUAGENS REGULARES E NÃO-REGULARES

Os resultados encontrados nas duas últimas seções determinam que as linguagens regulares são fechadas em relação a diversas operações, e que podem ser especificadas por expressões regulares ou por autômatos finitos, determinísticos ou não-determinísticos. Esses fatos, utilizados individual ou combinadamente, oferecem uma diversidade de técnicas para demonstrar que uma linguagem é regular.

Exemplo 2.4.1: Seja $\Sigma = \{0, 1, \ldots 9\}$ e $L \subseteq \Sigma^*$ o conjunto de representações decimais para inteiros não-negativos (sem 0's iniciais redundantes) divisíveis por 2 ou por 3. Por exemplo, 0, 3, 6, 244 $\in L$, mas 1, 03, 00 $\notin L$. Então, L é regular. Dividimos a prova em quatro partes.

Seja L_1 o conjunto de representações decimais de inteiros não-negativos. É fácil ver que

$$L_1 = 0 \cup \{1, 2, \ldots, 9\}\Sigma^*,$$

é regular, uma vez que está denotada através de uma expressão regular.

Seja L_2 o conjunto das representações decimais de números inteiros não-negativos divisíveis por 2. Então, L_2 é apenas o conjunto dos membros de L, terminados em 0, 2, 4, 6 ou 8; ou seja,

$$L_2 = L_1 \cap \Sigma^* \{0, 2, 4, 6, 8\}.$$

é regular pelo Teorema 2.3.1(e).

Seja L_3 o conjunto das representações decimais de números inteiros não-negativos divisíveis por 3. Lembre-se de que um número é divisível por 3, se, e somente se, a soma de seus dígitos é divisível por 3. Construímos um autômato finito que monitora em seu controle finito a soma módulo 3 de uma cadeia de dígitos. L_3, então, será a intersecção com L_1 da linguagem aceita por esse autômato finito, representado na Figura 2-18.

Por fim, $L = L_2 \cup L_3$, com certeza uma linguagem regular. ✧

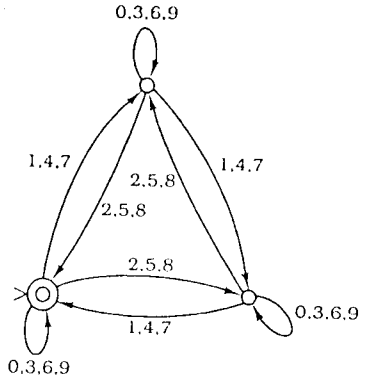


Figura 2-18

Embora agora já tenhamos visto uma série de técnicas poderosas para mostrar que uma linguagem é regular, ainda não dispomos de um método para demonstrar que uma linguagem é *não*-regular. Sabemos pelos princípios fundamentais, que existem linguagens não-regulares, visto que o conjunto de todas as expressões regulares (ou o conjunto dos autômatos finitos) é contável, enquanto o conjunto de todas as linguagens é incontável. Porém, demonstrar que qualquer linguagem particular não é regular requer ferramentas especiais.

Duas propriedades compartilhadas por todas as linguagens regulares, mas não por certas linguagens não-regulares, podem ser descritas intuitivamente como segue:

(1) Como uma cadeia é varrida da esquerda para a direita, a quantidade de memória exigida para determinar se a cadeia pertence ou não à linguagem deve ser limitada, fixada previamente e dependente da linguagem, não da particular cadeia de entrada. Por exemplo, esperamos que $\{a^n b^n: n \geq 0\}$ seja não-regular, uma vez que é difícil imaginar como se poderia construir um dispositivo de estados finitos que corretamente lembrasse, ao alcançar o limite entre os a's e os b's, quantos a's teria encontrado, para que tal número pudesse ser comparado com o número de b's.

(2) Linguagens regulares com um número infinito de cadeias são representadas por autômatos com ciclos, e por expressões regulares envolvendo a estrela de Kleene. Tais linguagens devem ter subconjuntos infinitos com uma certa estrutura simples e repetitiva associada à estrela de Kleene em uma expressão regular correspondente, ou a um circuito no diagrama de estados de um autômato finito. Isso nos leva a esperar, por exemplo, que $\{a^n: n \geq 1$ é um primo$\}$ não seja regular, uma vez que não há uma periodicidade simples no conjunto de números primos.

Essas idéias intuitivas são precisas, mas não o suficiente para serem utilizadas em provas formais. Devemos agora provar um teorema que captura essa intuição e identifica a não-regularidade de certas linguagens como conseqüência direta.

Teorema 2.4.1: *Seja L uma linguagem regular. Há um inteiro $n \geq 1$ tal que qualquer cadeia $w \in L$ com $|w| \geq n$ pode ser reescrito como $w = xyz$, tal que $y \neq \varepsilon$, $|xy| \leq n$, e $xy^iz \in L$ para cada $i \geq 0$.*

Prova: Como L é regular, é aceita por um autômato finito determinístico M. Seja n o número de estados de M, e w uma cadeia de comprimento n ou maior. Considere agora os primeiros n passos da computação de M sobre w:

$$(q_0, w_1 w_2 \ldots w_n) \vdash_M (q_1, w_2 \ldots w_n) \vdash_M \ldots \vdash_M (q_n, \varepsilon),$$

onde q_0 é o estado inicial de M e $w_1, \ldots w_n$ são os n primeiros símbolos de w. Como M tem somente n estados e há $n+1$ configurações $(q_i, w_{i+1} \ldots w_n)$ aparecendo na computação acima, *pelo princípio das casas de pombos existem* i e j, $0 \leq i < j \leq n$, tal que $q_i = q_j$. Em outras palavras, a cadeia $y = w_i w_{i+1} \ldots w_j$ guia M do estado q_i de volta para o estado q_i e essa cadeia é não-vazia visto que $i < j$. Contudo, essa cadeia poderia ser removida de w ou repetida qualquer número de vezes após o j-ésimo símbolo de w e M ainda aceitaria essa cadeia. Portanto, M reconhece $xy^iz \in L$ para cada $i \geq 0$, onde $x = w_1 \ldots w_i$ e $z = w_{j+1} \ldots w_m$. Note, por fim, que o comprimento de xy, o número acima definido como j, é, por definição, no máximo igual a n, como exigido pelo enunciado do teorema. ■

Esse teorema pertence a uma classe geral, chamada teoremas de bombeamento (*pumping theorems*), porque afirmam a existência de certos pontos em determinadas cadeias, nos quais uma subcadeia pode ser repetida-

mente inserida sem afetar a aceitabilidade da mesma. Em termos de sua estrutura matemática, o teorema do bombeamento acima é de longe o mais sofisticado teorema que já vimos até este ponto do livro, porque sua asserção, embora simples para provar e aplicar, envolve cinco quantificadores ao mesmo tempo. Considere novamente o que ele diz:

para cada linguagem regular L,
existe um $n \geq 1$, tal que,
para cada cadeia w em L mais longa que n,
existem cadeias x, y, z com $w = xyz$, $y \neq \varepsilon$, e $|xy| \leq n$, tal que
para cada $i \geq 0$ $xy^iz \in L$.

Aplicar o teorema corretamente pode ser sutil. Muitas vezes, é útil pensar na aplicação desse resultado como uma disputa entre você mesmo, que faz a prova, tentando mostrar que uma dada linguagem L não é regular, e um adversário, que insiste que L seja regular. O teorema afirma que, uma vez que L tenha sido escolhida, o adversário deve iniciar estabelecendo um número n; então, você apresenta uma cadeia w da linguagem, que seja mais longa que n; o adversário deve agora fornecer uma decomposição apropriada de w na forma xyz; até que você, triunfalmente, indique i, para o qual xy^iz não pertença à linguagem. Se você dispuser de uma estratégia que seja sempre vitoriosa, independentemente de quão brilhante seja o desempenho do seu adversário, então você provou que L não é regular.

Uma conseqüência desse teorema é que cada uma das duas linguagens mencionadas anteriormente nesta seção não é regular:

Exemplo 2.4.2: A linguagem $L = \{a^ib^i: i \geq 0\}$ não é regular, porque, se fosse, o Teorema 2.4.1 seria aplicável para algum inteiro n. Considere a string $w = a^nb^n \in L$. Pelo teorema, ela pode ser reescrita como $w = xyz$, tal que $|xy| \leq n$, e $y \neq \varepsilon$ – isto é, $y = a^i$ para algum $i > 0$. Mas então $xz = a^{n-i}b^n \notin L$, contradizendo o teorema. ✧

Exemplo 2.4.3: A linguagem $L = \{a^n: n$ é primo$\}$ não é regular. Supondo que ela fosse regular e que x, y e z fossem conforme especificados no Teorema 2.4.1, então, $x = a^p$, $y = a^q$ e $z = a^r$, onde $p, r \geq 0$ e $q > 0$. Pelo teorema, $xy^nz \in L$ para cada $n \geq 0$; isto é, $p + nq + r$ é primo para cada $n \geq 0$. Contudo, isso é impossível; supondo que $n = p+2q+r+2$; então $p+nq+r = (q+1) \cdot (p+2q+r)$, o qual, sendo um produto de dois números naturais, ambos maiores que 1, não pode ser primo. ✧

Exemplo 2.4.4: Às vezes vale a pena utilizar propriedades de fechamento para mostrar que uma linguagem não é regular. Seja, por exemplo,

$$L = \{w \in \{a, b\}^*: w \text{ tem um número igual de } a\text{'s e } b\text{'s}\}.$$

L não é regular, porque, se assim fosse também o seria a linguagem $L \cap a^*b^*$ – por fechamento das linguagens regulares em relação à intersecção; lembre-se do Teorema 2.3.1(e). Entretanto, essa última linguagem é precisamente $\{a^nb^n: n \geq 0\}$, a qual acabamos de demonstrar não ser uma linguagem regular. ✧

É interessante notar que o Teorema 2.4.1 pode de várias maneiras ser reescrito de forma substancialmente simplificada (veja uma delas no Problema 2.4.11).

Problemas para a Seção 2.4

2.4.1 **Uma progressão aritmética** é o conjunto $\{p + qn: n = 0, 1, 2, \ldots\}$ para algum $p, q \in N$.

(a) (Fácil) Mostre que se $L \subseteq \{a\}^*$ e $\{n: a^n \in L\}$ é uma progressão aritmética, então L é regular.

(b) Mostre que, se $L \subseteq \{a\}^*$ e $\{n: a^n \in L\}$ é uma união de um número finito de progressões aritméticas, então L é regular.

(c) (Difícil) Mostre que, se $L \subseteq \{a\}^*$ é regular, então $\{n: a^n \in L\}$ é uma união de um número finito de progressões aritméticas. (Essa é a volta da Parte (b).)

(d) Mostre que, se Σ é qualquer alfabeto e $L \subseteq \Sigma^*$ é regular, então $\{|w|: w \in L\}$ é uma união de um número finito de progressões aritméticas. (*Sugestão*: Utilizar a Parte (c).)

2.4.2 Sejam $D = \{0, 1\}$ e $T = D \times D \times D$. Uma correta adição de dois números, em notação binária, pode ser representada como uma cadeia sobre T^* se considerarmos os elementos de T como símbolos, representados em colunas verticais. Por exemplo,

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ +\ 0\ 1\ 1\ 0 \\ \hline 1\ 0\ 1\ 1 \end{array}$$

seria representado como a seguinte cadeia de símbolos formada por quatro elementos:

$$\begin{pmatrix}0\\0\\1\end{pmatrix}\begin{pmatrix}1\\1\\0\end{pmatrix}\begin{pmatrix}0\\1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix}$$

Mostre que o conjunto de todas as cadeias em T^* que representam adições corretas é uma linguagem regular.

2.4.3 Mostre que cada uma das seguintes linguagens é ou não uma linguagem regular. A notação decimal para um número é denotada, no modo usual, como uma cadeia sobre o alfabeto $\{0, 1, \ldots, 9\}$. Por exemplo, a notação decimal para o número 13 é uma cadeia de comprimento 2. Em **notação unária**, somente o símbolo I é utilizado; nessa notação, o número 5 seria representado como $IIIII$.

(a) $\{w: w$ é a notação unária para um número que seja múltiplo de 7$\}$

(b) $\{w: w$ é a notação decimal para um número que seja múltiplo de 7$\}$

(c) $\{w: w$ é a notação unária para um número n, tal que há um par de números primos que diferem por duas unidades, ambos maiores que $n\}$*

(d) $\{w: w$ é, para algum $n \geq 1$, a notação unária do número $10^n\}$

(e) $\{w: w$ é, para algum $n \geq 1$, a notação decimal do número $10^n\}$

(f) $\{w: w$ é uma seqüência de dígitos decimais que ocorre na expansão decimal infinita de $1/7\}$ (Por exemplo, 5714 é uma dessas seqüências, pois $1/7 = 0.14285714285714 \ldots$)

*N. de R. T. Dois números primos p e p+2 são conhecidos como "primos gêmeos" (em inglês, "twin primes")

2.4.4 Prove que $\{a^n b a^m b a^{m+n}: n, m \geq 1\}$ não é regular.

2.4.5 Utilizando o teorema do bombeamento e a propriedade do fechamento em relação à intersecção, mostre que as seguintes linguagens não são regulares.
(a) $\{www^R: w \in \{a, b\}^*\}$
(b) $\{www: w \in \{a, b\}^*\}$
(c) $\{w\overline{w}: w \in \{a, b\}^*\}$, onde $\overline{w}$ representa uma transcrição de w na qual cada ocorrência de a é substituída por b e vice-versa.

2.4.6 Uma cadeia x sobre o alfabeto $\{(,)\}$ é dita *equilibrada* caso se aplique o seguinte: (i) em qualquer prefixo de x o número de ('s não é menor que o número de)'s; (ii) o número de ('s em x é igual ao de)'s. Em outras palavras, x é equilibrado se ele pode ser obtido a partir de uma expressão aritmética válida, omitindo-se todas as variáveis, números e operações. (Veja, no próximo capítulo, uma definição menos complicada.) Mostre que o conjunto de todas as cadeias equilibradas sobre $\{(,)\}^*$ não é regular.

2.4.7 Mostre que, para qualquer autômato finito determinístico $M = (K, \Sigma, \delta, s, F)$, M reconhece uma linguagem infinita se, e somente se, M reconhece alguma cadeia de comprimento maior que, ou igual a $|K|$ e menor que $2|K|$.

2.4.8 As afirmações abaixo são verdadeiras ou falsas? Justifique suas respostas em cada caso. (Para todas elas, considere-se um alfabeto Σ fixo.)
(a) Cada subconjunto de uma linguagem regular é regular.
(b) Cada linguagem regular tem um subconjunto próprio regular.
(c) Se L é regular, então $\{xy: x \in L, \text{ e } y \notin L\}$ também o é.
(d) $\{w: w = w^R\}$ é regular.
(e) Se L é uma linguagem regular, então $\{w: w \in L \text{ e } w^R \in L\}$ também o é.
(f) Se C é qualquer conjunto de linguagens regulares, então $\bigcup C$ é uma linguagem regular.
(g) $\{xyx^R: x, y \in \Sigma^*\}$ é regular.

2.4.9 A noção de um autômato finito determinístico de duas fitas foi definida no Problema 2.1.5. Mostre que $\{(a^n b, a^m b^p): n, m, p > 0, n = m \text{ ou } n = p\}$ não é aceito por nenhum autômato finito determinístico de duas fitas. (*Sugestão*: Suponha que esse conjunto seja aceito por algum autômato finito determinístico de duas fitas M. Então, M reconhece $(a^n b, a^{n+1} b^n)$ para todo n. Mostre, por um argumento de bombeamento, que ele também reconhece $(a^n b, a^{n+1} b^{n+q})$ para algum $n \geq 0$ e $q > 0$, o que corresponde a uma contradição.) Pelo Problema 2.3.9, então, autômatos finitos não-determinísticos de duas fitas nem sempre podem ser convertidos em autômatos determinísticos, e pelo Problema 2.1.5, os conjuntos aceitos por autômatos finitos determinísticos de duas fitas não são fechados em relação à operação de união.

2.4.10 **Um autômato finito de dois cabeçotes** é um autômato finito com dois cabeçotes de leitura/gravação que podem mover-se independentemente, mas somente da esquerda para a direita, sobre a fita de entrada. Como em um autômato finito de duas fitas (Problema 2.1.5), o conjunto de estados é dividido em duas partes; cada parte corresponde a ler e mover um dos cabeçotes sobre a fita. Uma cadeia é aceita se ambos os cabeçotes se encontrarem juntos no final da cadeia, com o controle finito posicionado em um estado final. Autômatos finitos de dois cabeçotes podem ser determinísticos ou não-determinísticos. Utilizando uma notação de diagrama de estados à sua escolha, mostre que as seguintes linguagens são aceitas por autômatos finitos de dois cabeçotes.
(a) $\{ab: n \geq 0\}$
(b) $\{wcw: w \in \{a, b\}^*\}$
(c) $\{a^1 b a^2 b a^3 b \ldots b a^k b: k \geq 1\}$
Em quais casos você pode tornar determinísticas as suas máquinas?

2.4.11 Este problema estabelece uma versão mais forte do Teorema de bombeamento 2.4.1; o objetivo é fazer a parte "bombeada" a mais longa possível. Supondo que $M = (K, \Sigma, \delta, s, F)$ seja um autômato finito determinístico e que w seja qualquer cadeia em $L(M)$ de comprimento pelo menos $|K|$, mostre que há cadeias x, y e z, tal que $w = xyz$, $|y| \geq (|w| - |K| + 1)/|K|$, e $xy^n z \in L(M)$ para cada $n \geq 0$.

2.4.12 Sejam $D = \{0, 1\}$ e $T = D \times D \times D$. Uma *multiplicação* correta de dois números em notação binária pode também ser representada como uma cadeia sobre T^*. Assim, por exemplo, a multiplicação $10 \times 5 = 50$, ou

$$\begin{array}{r} 0\,0\,1\,0\,1\,0 \\ \underline{\times\, 0\,0\,0\,1\,0\,1} \\ 1\,1\,0\,0\,1\,0 \end{array}$$

seria representada pela seguinte cadeia de seis símbolos.

$$\begin{pmatrix}0\\0\\1\end{pmatrix}\begin{pmatrix}0\\0\\1\end{pmatrix}\begin{pmatrix}1\\0\\0\end{pmatrix}\begin{pmatrix}0\\1\\0\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}0\\1\\0\end{pmatrix}$$

Mostre que o conjunto de todas as cadeias sobre T^* que representa de modo correto as multiplicações não é uma linguagem regular. (*Sugestão*: Considere a multiplicação $(2^n + 1) \times (2^n + 1)$.)

2.4.13 Seja $L \subseteq \Sigma^*$ uma linguagem, e $L_n = \{x \in L: |x| \leq n\}$. A *densidade* de L é a função $d_L(n) = |L_n|$.
(a) Qual é a densidade de $(a \cup b)^*$?
(b) Qual é a densidade de $ab^*ab^* \; ab^*a$?[1]
(c) Qual é a densidade de $(ab \cup aab)^*$?
(d) Mostre que a densidade de qualquer linguagem regular é limitada superiormente por um polinômio, ou inferiormente por uma exponencial (a função na forma 2^{cn} para algum n). Em outras palavras, densidades de linguagens regulares não podem ser funções de taxas de crescimento intermediário, tais como $n^{\log n}$. (*Sugestão*: Considere um autômato finito determinístico que aceita L e todos os ciclos – caminhos fechados sem repetições de vérti-

[1] N. de R. T. Há um erro no original, em que o segundo * não aparece como expoente mas como símbolo normal.

ces – presentes no diagrama de estado desse autômato. O que ocorre se nenhum par de ciclos compartilhar um nó? O que ocorre se houver dois ciclos que compartilham um mesmo nó?)

2.5 MINIMIZAÇÃO DE ESTADOS

Na última seção, constatamos nossa suspeita de que autômatos finitos determinísticos constituem modelos rudimentares de computadores: computações baseadas em autômatos finitos não podem realizar tarefas computacionais simples como é o caso da comparação do número de *a*'s e de *b*'s em uma cadeia. Entretanto, autômatos finitos são úteis como *partes* básicas de computadores e algoritmos. Por isto, é importante sermos capazes de *minimizar o número de estados* de um dado autômato finito determinístico, isto é, determinar um autômato finito determinístico equivalente que tenha o menor número de estados possível. Em seguida, desenvolveremos os conceitos necessários e os resultados que conduzem a um *algoritmo de minimização de estados*.

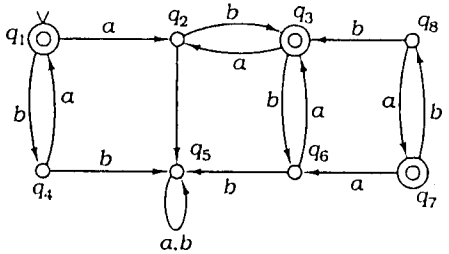


Figura 2-19

Dado um autômato finito determinístico, pode haver um modo fácil de eliminar alguns dos seus estados. Tomando, por exemplo, o autômato determinístico da Figura 2-19, que aceita a linguagem $L = (ab \cup ba)^*$ (o que não é muito difícil de verificar), consideremos o estado q_7. É fácil notar que esse estado é *inatingível*, uma vez que não há qualquer caminho do estado inicial para ele no diagrama de estados do autômato. Este é o tipo mais simples de simplificação que se pode fazer em qualquer autômato finito determinístico: remover todos os estados inatingíveis e todas as transições para dentro e para fora deles. De fato, essa simplificação era implícita em nossa conversão de um autômato finito não-determinístico para seu equivalente determinístico (lembre-se do Exemplo 2.2.4): desconsideramos todos os estados (conjuntos de estados do autômato original) que não são atingíveis a partir do estado inicial do autômato resultante.

Identificar os estados atingíveis é uma tarefa que pode ser executada em tempo polinomial, porque o conjunto de estados atingíveis pode ser definido como o fechamento de $\{s\}$ sob a relação $\{(p, q): \delta(p, a) = q$ para algum $a \in \Sigma\}$. Portanto, o conjunto de todos os estados atingíveis pode ser computado por meio do seguinte algoritmo:

$R := \{s\}$;
Enquanto existir um estado $p \in R$ e um correspondente símbolo $a \in \Sigma$, tal que $\delta(p, a) \notin R$, acrescentar $\delta(p, a)$ a R.

Entretanto, o autômato resultante da exclusão de todos os estados inatingíveis (Figura 2-20), ainda apresenta mais estados que não são realmente necessários, desta vez por razões mais sutis. Por exemplo, os estados q_4 e q_6 são *equivalentes* e, portanto, eles *podem ser convertidos em um único estado*. O que significa isso, exatamente? Intuitivamente, a razão de considerarmos esses estados como equivalentes é que, *a partir de qualquer um deles, precisamente as mesmas cadeias conduzem o autômato à aceitação*.

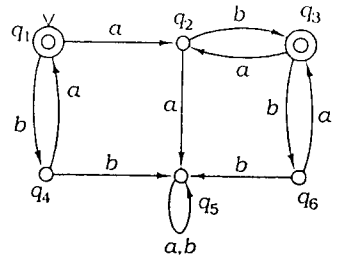


Figura 2-20

Nossa próxima definição captura uma relação similar entre cadeias que têm "um destino comum" com relação a uma linguagem.

Definição 2.5.1: Supondo que $L \subseteq \Sigma^*$ seja uma linguagem e que $x, y \in \Sigma^*$, dizemos que x e y são **equivalentes em relação a** L, denotado $x \approx_L y$, se, para todos os $z \in \Sigma^*$, o seguinte é verdadeiro: $xz \in L$ se e somente se $yz \in L$. Note que $\approx_L$ é uma relação de equivalência.

Em outras palavras, $x \approx_L y$ se ambas as cadeias pertencem a L ou se nenhuma das duas está em L; além disso, acrescentando qualquer cadeia fixa a ambos, x e y, resultam duas cadeias que ou estão ambas em L ou então ambas não estão em L.

Exemplo 2.5.1: Quando L é compreendida pelo contexto, denotamos por $[x]$ a classe de equivalência, em relação a L, à qual x pertence. Por exemplo, para a linguagem $L = (ab \cup ba)^*$, aceita pelo autômato na Figura 2-20, não é difícil ver que $\approx_L$ tem *quatro* classes de equivalência:

(1) $[\varepsilon] = L$,
(2) $[a] = La$,
(3) $[b] = Lb$,
(4) $[aa] = L(aa \cup bb)\Sigma^*$.

Em (1), para qualquer cadeia $x \in L$, incluindo $x = \varepsilon$, os z's que fazem $xz \in L$ são precisamente os membros de L. Em (2), qualquer $x \in La$ precisa de um z na forma bL a fim de que xz esteja em L. Analogamente, para (3) os z's estão na forma aL. Por fim, em (4) não há z que possa restaurar para L uma cadeia que tenha um prefixo em $L(aa \cup bb)$. Em outras palavras, todas as cadeias

no conjunto (1) têm *o mesmo destino*, quando estudadas em relação à inclusão em L; o mesmo ocorre para (2), (3) e (4). Finalmente, é fácil ver que essas quatro classes esgotam todas as cadeias de Σ^*. Portanto, essas são as classes de equivalência de $\approx_L$.✧

Note que $\approx_L$ refere-se a cadeias em termos de uma linguagem, não em termos de um autômato. Autômatos disponibilizam a relação descrita a seguir que não é tão fundamental:

Definição 2.5.2: Seja $M = (K, \Sigma, \delta, s, F)$ um autômato finito determinístico. Dizemos que duas cadeias $x, y \in \Sigma^*$ são **equivalentes em relação a** M, denotada $x \sim_M y$, se, intuitivamente, ambas levarem M de um estado s para um mesmo estado. Formalmente, $x \sim_M y$, se há um estado q, tal que $(s, x) \vdash_M^* (q, \varepsilon)$ e $(s, y) \vdash_M^* (q, \varepsilon)$.

Novamente, $\sim_M$ é uma relação de equivalência. Suas classes de equivalência podem ser identificadas pelos estados de M – mais precisamente, com aqueles estados que são alcançáveis a partir de s e que, portanto, têm pelo menos uma cadeia na correspondente classe de equivalência. Denotamos com E_q a classe de equivalência correspondente para o estado q de M.

Exemplo 2.5.1 (continuação): Por exemplo, para o autômato M da Figura 2-20, as classes de equivalência de $\sim_M$ são essas (onde $L = (ab \cup ba)^*$ é a linguagem aceita por M)

(1) $E_{q1} = (ba)^*$,
(2) $E_{q2} = La \cup a$,
(3) $E_{q3} = abL$,
(4) $E_{q4} = b(ab)^*$,
(5) $E_{q5} = L(bb \cup aa)\Sigma^*$,
(6) $E_{q6} = abLb$.

Novamente, elas formam uma partição de Σ^*.✧

Essas duas importantes relações de equivalência, uma associada à linguagem, outra ao autômato, estão relacionadas como segue:

Teorema 2.5.1: *Para qualquer autômato finito determinístico $M = (K, \Sigma, \delta, s, F)$ e quaisquer cadeias $x, y \in \Sigma^*$, se $x \sim_M y$, então $x \approx_{L(M)} y$.*

Prova: Para qualquer cadeia $x \in \Sigma^*$, seja $q(x) \in K$ o único estado tal que $(s, x) \vdash_M^* (q(x), \varepsilon)$. Note que, para qualquer $x, z \in \Sigma^*$, $xz \in L(M)$ se e somente se $(q(x), z) \vdash_M^* (f, \varepsilon)$ para algum $f \in F$. Agora, se $x \sim_M y$ então, pela definição de $\sim_M$, $q(x) = q(y)$ e, portanto, $x \sim_M y$ implica que o seguinte é verdadeiro:

$$xz \in L(M) \text{ se e somente se } yz \in L(M) \text{ para todo } z \in \Sigma^*,$$

o que é o mesmo que $x \approx_{L(M)} y$. ■

Um modo muito sugestivo de expressar o Teorema 2.5.1 é dizer que $\sim_M$ *é um* **refinamento** de $\approx_{L(M)}$. Em geral, dizemos que uma relação de equivalência $\sim$ é um refinamento de outra $\approx$ se para todo x, y $x \sim y$ implica $x \approx y$. Se $\sim$ é um refinamento de $\approx$, então cada classe de equivalência com relação a $\sim$ está contida em alguma classe de equivalência de $\approx$; em outras palavras, cada classe de equivalência de $\approx$ é a união de uma ou mais classes de equivalência de $\sim$. Por exemplo, a relação de equivalência que associa duas cidades quaisquer dos Estados Unidos que estão no mesmo *condado* (*county*) é um refinamento da relação de equivalência que liga duas cidades quaisquer do mesmo *estado* (*state*).

Exemplo 2.5.1 (continuação): Para um exemplo um pouco mais específico, as classes de equivalência de $\sim_M$ para o autômato M na Figura 2-20 "refina", nessa visão, as classes de equivalência de $\approx_{L(M)}$, exatamente como previsto pelo Teorema 2.5.1. Por exemplo, as classes E_{q5} e $[aa]$ coincidem, enquanto as classes E_{q1} e E_{q3} são subconjuntos de $[\varepsilon]$.✧

O teorema 2.5.1 tem implicações importantes sobre M e qualquer outro autômato M que aceite a mesma linguagem $L(M)$: seu número de estados deve ser *pelo menos tão grande* quanto o número de classes de equivalência de $L(M)$ sob $\approx$. Em outras palavras, o número de classes de equivalência de $L(M)$ *é* um limitante inferior natural do número de estados de qualquer autômato equivalente a M. Pode esse limitante ser alcançado? A seguir, mostramos que, de fato, pode.

Teorema 2.5.2 (Teorema de Myhill-Nerode): Seja $L \subseteq \Sigma^*$ uma linguagem regular. Então, há um autômato finito determinístico que reconhece L e tem precisamente o mesmo número de estados que o número de classes de equivalência em $\approx_L$.

Prova: Novamente, denotamos por $[x]$ a classe de equivalência da cadeia $x \in \Sigma^*$ na relação de equivalência $\approx_L$. Dado L, devemos construir um autômato finito determinístico (o **autômato-padrão para** L) $M = (K, \Sigma, \delta, s, F)$, tal que $L = L(M)$. M é definida como segue:

$K = \{[x]: x \in \Sigma^*\}$, o conjunto de classes de equivalência em relação a $\approx_L$.
$s = [\varepsilon]$, a classe de equivalência de ε em relação a $\approx_L$
$F = \{[x]: x \in L\}$.
Por fim, para qualquer $[x] \in K$ e qualquer $a \in \Sigma$, defina $\delta([x], a) = [xa]$.

Como saber se o conjunto K é finito, isto é, se $\approx_L$ tem um número finito de classes de equivalência? L é regular, sendo portanto com certeza aceito por algum autômato finito determinístico M'. Pelo teorema anterior, $\sim_{M'}$ é um refinamento de $\approx_L$ e, deste modo, há menos classes de equivalência em L que em $\sim_{M'}$ – ou seja, estados de M'. Portanto, K é um conjunto finito. Também temos o argumento de que δ é bem definida, isto é, $\delta([x], a) = [xa]$ para qualquer cadeia $x \in [x]$. Isso é de fácil verificação porque $x \approx_L x'$ se e somente se $xa \approx_L x'a$. A seguir, demostraremos que $L = L(M)$. Para isso, mostraremos inicialmente que, para todos os $x, y \in \Sigma^*$, temos

$$([x], y) \vdash_M^* ([xy], \varepsilon). \qquad (1)$$

Este resultado pode ser estabelecido por indução em $|y|$. É trivial quando $y = \varepsilon$, e, se isso se aplica a todos os y's de comprimento até n e $y = y'a$, então, por indução, $([x], y'a) \vdash^*_M ([xy'], a) \vdash^*_M ([xy], \varepsilon)$.

Agora (1) completa a prova: para todo $x \in \Sigma^*$, temos que $x \in L(M)$, se, e somente se, $([\varepsilon], x) \vdash^*_M (q, \varepsilon)$ para algum $q \in F$, o que é por (1) o mesmo que dizer $[x] \in F$, ou, pela definição de F, $[x \in L]$. ■

Exemplo 2.5.1 (continuação): O autômato padrão correspondente à linguagem $L = (ab \cup ba)^*$, aceita pelo autômato finito determinístico de seis estados mostrado na Figura 2-20, é apresentado na Figura 2-21. Ele tem *quatro* estados. Naturalmente, este é o menor autômato finito determinístico que reconhece essa linguagem. ✧

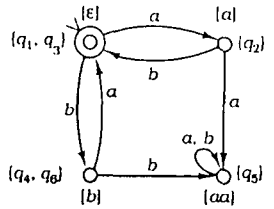


Figura 2-21

O Teorema 2.5.2, por vezes denominado Teorema de Myhill-Nerode, implica a seguinte caracterização de linguagens regulares:

Corolário: *Uma linguagem L é regular se e somente se $\approx_L$ tiver um número finito de classes de equivalência.*

Prova: Se L é regular, então $L = L(M)$ para algum autômato finito determinístico M, e M tem pelo menos o mesmo número de estados que o número de classes de equivalência de $\approx_L$. Existirá portanto, ao menos, um número finito de classes de equivalência em $\approx_L$.

Inversamente, se $\approx_L$ apresenta um número finito de classes de equivalência, então o autômato finito padrão determinístico M_L (lembre da prova do Teorema 2.5.2) reconhece L. ■

Exemplo 2.5.2: O corolário que acabamos de provar é uma interessante alternativa para a caracterização do significado de uma linguagem L ser regular. Além disso, proporciona uma outra maneira de provar que uma linguagem não é regular – além do Teorema do Bombeamento.

Como exemplo, eis uma prova alternativa de que $L = \{a^n b^n : n \geq 1\}$ não é regular: não existem duas cadeias a^i e a^j, com $i \neq j$, que sejam equivalentes sob $\approx_L$, simplesmente porque há uma cadeia (no caso, b^i) que, quando concatenada com a^i, resulta em uma cadeia em L, mas quando anexada a a^j, produz uma cadeia que não pertence a L. Daí $\approx_L$ terá um número infinito de classes de equivalência $[\varepsilon]$, $[a]$, $[aa]$, $[aaa]$, ... e, portanto, pelo corolário, L não é regular. ✧

Para qualquer linguagem regular L, o autômato construído na prova do Teorema 2.5.2 é o autômato determinístico, com o número mínimo de estados, que reconhece L – um resultado de óbvia importância prática. Infelizmente, esse autômato é definido em termos das classes de equivalência de $\approx_L$, e não é óbvia a maneira como tais classes de equivalência podem ser identificadas para qualquer dada linguagem regular L – especialmente se L for definida em termos de um autômato determinístico M. A seguir, desenvolveremos um *algoritmo* para construir esse autômato mínimo, partindo de *qualquer autômato finito determinístico* M, tal que $L = L(M)$.

Seja $M = (K, \Sigma, \delta, s, F)$ um autômato finito determinístico. Definamos a relação $A_M \subseteq K \times \Sigma^*$ como segue: $(q, w) \in A_M$ se, e somente se, $(q, w) \vdash^*_M (f, \varepsilon)$ para algum $f \in F$, isto é, $(q, w) \in A_M$ significa que w leva M do estado q para um estado de aceitação. Diremos que dois estados $q, p \in K$ são **equivalentes**, denotado $q \equiv p$, se, e somente se, $(p, z) \in A_M$ para todos os $z \in \Sigma^*$: $(q, z) \in A_M$. Portanto, se dois estados forem equivalentes, então as correspondentes classes de equivalência de $\sim_M$ são subconjuntos da mesma classe de equivalência de $\approx_L$. Em outras palavras, as classes de equivalência de $\equiv$ são precisamente os conjuntos de estados de M que devem ser agrupados a fim de se obter o autômato padrão de $L(M)$[†].

Desenvolveremos a seguir um algoritmo para computar as classes de equivalência de $\equiv$.

Nosso algoritmo irá computar $\equiv$ como o *limite* de uma seqüência de relações de equivalência $\equiv_0$, $\equiv_1$, $\equiv_2$,..., definidas a seguir. Para dois estados $q, p \in K$ diz-se que $q \equiv_n p$, quando $(q, z) \in A_M$ se, e somente se, $(p, z) \in A_M$ para todas as cadeias z, tais que $|z| \leq n$. Em outras palavras, $\equiv_n$ é uma relação de equivalência mais grosseira do que $\equiv$, somente requerendo que estados q e p se comportem da mesma maneira quanto à aceitação de cadeias *de comprimento não superior a n*.

Obviamente, cada relação de equivalência $\equiv_0$, $\equiv_1$, $\equiv_2$,... é refinamento da anterior. Além disso, $q \equiv_0 p$ vale se q e p forem ambos estados finais ou ambos não-finais, isto é, há precisamente duas classes de equivalência de $\equiv_0$: F e $K - F$ (considerando ambos não-vazios). Resta investigar de que maneira $\equiv_{n+1}$ depende de $\equiv_n$:

Lema 2.5.1: *Para quaisquer dois estados $q, p \in K$ e qualquer inteiro $n \geq 1$, $q \equiv_n p$ se, e somente se, (a) $q \equiv_{n-1} p$ e (b) para todo $a \in \Sigma$, $\delta(q, a) \equiv_n \delta(p, a)$.*

Prova: Por definição de $\equiv_n$, $q \equiv_n p$ se e somente se $q \equiv_{n-1} p$ e também qualquer cadeia $w = av$ de comprimento n leva dos estados q e p para um estado de aceitação ou para um estado de não-aceitação. Entretanto, a segunda condição equivale a afirmar que $\delta(q, a) \equiv_n \delta(p, a)$ para qualquer $a \in \Sigma$. ■

[†] A relação $\equiv$ pode ser chamada de *quociente* de $\sim_M$ por $\approx_L$. Se $\sim$ é um refinamento de $\approx$, então o quociente de $\approx$ por $\sim$, denotada $\approx / \sim$, é *uma relação de equivalência* sobre as classes de equivalência de $\sim$. Duas classes $[x]$ e $[y]$ de $\sim$ são relacionadas por $\approx / \sim$ se $x \approx y$ – é fácil ver que isso é de fato uma relação de equivalência. Para retornar a nosso exemplo geográfico, o quociente da relação "mesmo estado" pela relação "mesmo condado" relaciona quaisquer dois condados (cada um com pelo menos uma cidade nele) que pertencem ao mesmo estado.

O Lema 2.5.1 sugere que podemos computar $\equiv$ e, a partir disso, o autômato padrão para L, pelo seguinte algoritmo:

Inicialmente as classes de equivalência de $\equiv_n$ são F e $K - F$;
Repetir para n:= 1, 2...
calcule as classes de equivalência de $\equiv_n$ a partir das $\equiv_{n-1}$
até $\equiv_n$ ser igual a $\equiv_{n-1}$.

Cada iteração pode ser realizada aplicando o Lema 2.5.1: para cada par de estados de M, testamos se as condições do lema se aplicam e, em caso afirmativo, colocamos ambos na mesma classe de equivalência de $\equiv_n$. Mas como saber se esse procedimento *é um algoritmo*, cuja execução sempre irá terminar? A resposta é simples: para toda iteração para a qual a condição de término não seja satisfeita, $\equiv_n \neq \equiv_{n-1}$, $\equiv_n$ é um *refinamento próprio* de $\equiv_{n-1}$ e, portanto, $\equiv_n$ tem pelo menos uma classe equivalência a mais do que $\equiv_{n-1}$. Como o número de classes de equivalência não pode tornar-se maior que o número de estados de M, o algoritmo terminará após, no máximo, $|K| - 1$ iterações.

Quando o algoritmo termina, por exemplo, na n-ésima iteração, tendo computado $\equiv_n = \equiv_{n-1}$, então o lema implica $\equiv_n = \equiv_{n+1} = \equiv_{n+2} = \equiv_{n+3} = \ldots$. Portanto, a relação computada será precisamente $\equiv$. Na próxima seção, ofereceremos uma análise mais cuidadosa da complexidade desse importante algoritmo, determinando que ele é polinomial.

Exemplo 2.5.3: Vamos aplicar o algoritmo de minimização de estados para o autômato finito determinístico M na Figura 2-20 (naturalmente, pelo exemplo anterior, já conhecemos o resultado esperado: o autômato padrão de quatro estados para $L(M)$). Nas várias iterações, devemos ter classes de equivalência do correspondente $\equiv_i$:

Inicialmente, as classes de equivalência de $\equiv_0$ são $\{q_1, q_3\}$ e $\{q_2, q_4, q_5, q_6\}$. Após a primeira iteração, as classes de $\equiv_1$ são $\{q_1, q_3\}$, $\{q_2\}$, $\{q_4, q_6\}$ e $\{q_5\}$. O refinamento ocorreu porque $\delta(q_2,b) \not\equiv_0 \delta(q_4,b)$, $\delta(q_5,b)$ e $\delta(q_4, a) \not\equiv_0 \delta(q_5, a)$. Após a segunda iteração, não há novos refinamentos de classes. O algoritmo, portanto, termina, e o autômato minimizado resultante é mostrado na Figura 2-22.

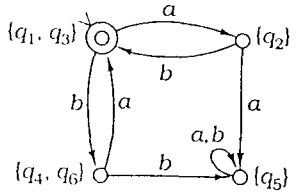


Figura 2-22

Como esperado, ele é isomórfico ao autômato-padrão mostrado na Figura 2-21.✧

Exemplo 2.5.4: Lembre-se da linguagem $L \subseteq \{a_1, \ldots, a_n\}^*$ contendo todas as cadeias que não apresentam ocorrências de todos os n símbolos, e o autômato finito determinístico correspondente, com 2^n estados (Exemplo 2.2.5).

Podemos agora mostrar que esses são todos os estados necessários. A razão é que $\approx_L$ tem 2^n classes de equivalência: para cada subconjunto A de Σ, seja L_A o conjunto de todas as cadeias que contêm ocorrências de todos os símbolos de A e de não-símbolos de $\Sigma - A$ (por exemplo, $L_\emptyset = \{\varepsilon\}$). Então, não é difícil ver que os 2^n conjuntos L_A são precisamente as classes de equivalência de $\approx_L$. Em outras palavras, se $x \in L_A$ e $y \in L_B$ para dois subconjuntos distintos A e B de Σ então, para qualquer $z \in L_{\Sigma-B}$, $xz \in L$ e $yz \notin L$ (aqui consideramos que B não esteja contido em A; se isto não for verdade, invertam-se os papéis dos conjuntos A e B).✧

Lembre-se de que há um *autômato finito não-determinístico* com $n + 1$ estados que reconhece a mesma linguagem (Exemplo 2.2.2). Embora saibamos que os autômatos determinísticos são exatamente tão poderosos quanto os não-determinísticos, o preço do determinismo reflete-se no número de estados, que é, no pior caso, exponencial. Antecipando importantes questões da *complexidade computacional* discutidas nos Capítulos 6 e 7, podemos dizer, observando de outro ponto de vista, que quando o número de estados é priorizado, *o não-determinismo é exponencialmente mais poderoso que o determinismo no caso dos autômatos finitos.*

Problemas para a Seção 2.5

2.5.1 (a) Forneça as classes de equivalência em relação à $\approx_L$ para as seguintes linguagens:
(i) $L = (aab \cup ab)^*$.
(ii) $L = \{x: x$ contém uma ocorrência de $aababa\}$.
(iii) $L = \{xx^R: x \in \{a, b\}^*\}$.
(iv) $L = \{xx: x \in \{a, b\}^*\}$.
(v) $L_n = \{a, b\}a\{a, b\}^n$, onde $n > 0$ é um número inteiro fixo.
(vi) A linguagem dos parênteses equilibrados (Problema 2.4.6).
(b) Com relação às linguagens em (a), para as quais a resposta é finita, apresente um autômato finito determinístico com o menor número possível de estados que reconheça a linguagem correspondente.

2.5.2 Uma cadeia $x \in \Sigma^*$ é dita *square-free* se ela não puder ser escrita na forma $x = uvvw$ para algum u, v, $w \in \Sigma^*$, $v \neq \varepsilon$. Por exemplo, *lewis* e *christos* são square-free, mas *harry* e *papadimitriou* não o são. Mostre que, se $|\Sigma| > 1$, então o conjunto de todas as cadeias *square-free* em Σ^* não é regular.

2.5.3 Para cada um dos autômatos finitos (determinístico ou não-determinístico) considerados nos Problemas 2.1.2 e 2.2.9, determine o autômato finito determinístico equivalente que apresenta um número mínimo de estados.

2.5.4 Um **autômato finito de mão dupla** é similar a um autômato finito determinístico, exceto que o cabeçote de leitura pode avançar e retroceder sobre a fita de entrada. Se ela tentar retroceder a partir da extremidade esquerda da fita, o autômato pára de operar, sem aceitar a entrada. Formalmente, um autômato finito de mão dupla M é

uma quíntupla $(K, \Sigma, \delta, s, F)$, onde K, Σ, s e F mantêm a definição adotada para autômatos finitos determinísticos, e δ é uma função de $K \times \Sigma$ para $K \times \{\leftarrow, \rightarrow\}$, onde $\leftarrow$ ou $\rightarrow$ indicam a direção do movimento do cabeçote. Uma configuração é um membro de $K \times \Sigma^* \times \Sigma^*$; a configuração (p, u, v) indica que a máquina está no estado p com o cabeçote posicionado no primeiro símbolo de v e com u à esquerda do cabeçote. Se $v = \varepsilon$, a configuração (p, u, ε) indica que M completou sua operação no estado p, tendo acabado de ler u.

Escrevemos $(p_1, u_1, v_1) \vdash_M (p_2, u_2, v_2)$ se, e somente se, $v_1 = \sigma v$ para algum $\sigma \in \Sigma$, $\delta(p_1, \sigma) = (p_2, p)$ e/ou

1. $p = \rightarrow$ e $u_2 = u_1\sigma$, $v_2 = v$, ou
2. $p = \leftarrow$, $u_1 = u\sigma'$ para algum $u \in \Sigma^*$ e $v_2 = \sigma' v_1$.

Como sempre, $\vdash^*_M$ é o fechamento transitivo reflexivo de $\vdash_M$. M reconhece w, se e somente se $(s, \varepsilon, w) \vdash^*_M (f, w, \varepsilon)$ para algum $f \in F$. Nesse problema você irá utilizar o Teorema de Myhill-Nerode (o Teorema 2.5.2 e seu corolário) para mostrar que uma linguagem aceita por um autômato finito é aceita por um autômato finito de mão dupla, e portanto, o aparente poder adicional conferido pela movimentação do cabeçote para a esquerda não aumenta o poder dos autômatos finitos.

Seja M um autômato finito de mão dupla, conforme acabamos de definir.

(a) Seja $q \in K$ e $w \in \Sigma^*$. Mostre que há, no máximo, um $p \in K$, tal que $(q, \varepsilon, w) \vdash^*_M (p, w, \varepsilon)$.

(b) Seja t algum elemento fixo *não* pertencente ao conjunto K. Para qualquer $w \in \Sigma^*$, defina uma função χ_w: $K \rightarrow K \cup \{t\}$, como segue.

$$\chi_w(q) = \begin{cases} p, \text{ se } (q, \varepsilon, w) \vdash^*_M (p, w, \varepsilon) \\ t, \text{ caso contrário} \end{cases}$$

Pela Parte (a), χ_w é bem definida. Além disso, para qualquer $w \in \Sigma^*$, define-se θ_w: $K \times \Sigma \rightarrow K \cup \{t\}$, como segue:

$$\theta_w(q, a) = \begin{cases} p, & \text{se } (q, w, a) \vdash^+_M (p, w, a), \text{ mas não é verdade que} \\ & (q, w, a) \vdash^+_M (r, w, a) \vdash^+_M (p, w, a) \text{ para qualquer } r \neq p, \\ t, & \text{se não há } p \in K, \text{ tal que } (q, w, a) \vdash^+_M (p, w, a) \end{cases}$$

(Aqui $\vdash^+_M$ corresponde ao fechamento reflexivo (não-transitivo) de $\vdash_M$, isto é, a relação "produz resultado em um ou mais passos" sobre o conjunto das configurações.) Agora sejam $w, v \in \Sigma^*$, $\chi_w = \chi_v$ e $\theta w = \theta v$. Mostre que, para qualquer $u \in \Sigma^*$, M aceita wu, se, e somente se, M aceita vu.

(c) Mostre que, se $L(M)$ é a linguagem aceita por um autômato determinístico de mão dupla, então $L(M)$ é aceito por algum autômato finito determinístico convencional (mão única). (*Sugestão*: Utilize (b) acima para demonstrar que $\equiv_{L(M)}$ tem um número finito de classes de equivalência.)

(d) Prove que há um algoritmo exponencial para o qual, dado um autômato determinístico de dois modos M, constrói um autômato finito determinístico equivalente. (*Sugestão*: Quantas funções χ_w e θ_w diferentes podem existir como uma função de $|K|$ e $|\Sigma|$?)

(e) Projete um autômato finito determinístico de mão dupla com $O(n)$ estados, que aceite a linguagem $L_n = \{a, b\}^* a \{a, b\}^n$ (lembre-se do Problema 2.5.1(a)(v)). Comparando com o Problema 2.5.1(b)(v), prove que o crescimento exponencial em (d) acima é inevitável.

(f) Os argumentos e a construção utilizados nesse problema podem ser estendidos para autômatos finitos *não-determinísticos* de mão dupla?

2.6 ALGORITMOS PARA AUTÔMATOS FINITOS

Muitos dos resultados obtidos neste capítulo tratam de diferentes modos de representação de linguagens regulares como linguagens aceitas por autômatos finitos, determinísticos ou não, ou então como linguagens geradas por expressões regulares. Na seção anterior, vimos como, dado um autômato finito determinístico qualquer, podemos obter o autômato finito determinístico equivalente, que apresente o menor número possível de estados. Todos esses resultados são construtivos no sentido de que suas provas imediatamente sugerem *algoritmos* para os quais a partir de uma representação de um certo tipo, produzem uma representação equivalente de qualquer um dos outros tipos. Nesta subseção explicaremos esses algoritmos, e faremos uma análise grosseira da sua complexidade.

Partiremos do algoritmo de conversão de um autômato finito não-determinístico para um equivalente determinístico (Teorema 2.2.1); agora focalizando sua complexidade. As entradas do algoritmo definem um autômato finito não-determinístico $M = (K, \Sigma, \Delta, s, F)$. Assim devemos calcular a complexidade deste algoritmo como uma função das cardinalidades de K, Σ e Δ. O desafio básico é computar a função de transição do autômato determinístico, ou seja, aquela que, para cada $Q \subseteq K$ e cada $a \in \Sigma$, determina

$$\delta(Q, a) = \bigcup \{E(p) : p \in K \text{ e } (q, a, p) \in \Delta \text{ para algum } q \in Q\}.$$

É mais prático *pré-computar* a forma geral de todos os $E(p)$'s de uma vez por todas, utilizando o algoritmo de fechamento apresentado nessa prova. Isso pode ser feito em tempo total $O(|K|^3)$ para todos os $E(p)$'s. Uma vez obtidos os $E(p)$'s, a computação de $\delta(Q, a)$ pode ser realizada colecionando-se inicialmente todos os estados p, de tal modo que $(q, a, p) \in \Delta$ e, então, obtendo-se a união de todos os $E(p)$'s – o total $O(|\Delta||K|^2)$ das operações elementares tais como adicionar um elemento a um subconjunto de K. A complexidade do algoritmo total é portanto, $O(2^{|K|}|K|^3|\Sigma||\Delta||K|^2)$. Não é surpresa que nossa estimativa de complexidade seja uma função exponencial do comprimento da entrada (evidenciado pelo fator $2^{|K|}$); assim, a obtenção do algoritmo (o autômato finito determinístico equivalente) pode ser, no pior caso, exponencial.

Converter uma expressão regular R em um *autômato finito não-determinístico* equivalente (Teorema 2.3.2) é relativamente mais eficiente: é fácil mostrar, por indução no comprimento de R, que o autômato resultante não tem mais que $2|R|$ estados e, portanto, não mais que $4|R|^2$ transições – ou seja, um polinômio.

Transformar um autômato finito $M = (K, \Sigma, \Delta, s, F)$ (determinístico ou não) em uma expressão regular gerando a mesma linguagem (Teorema 2.3.2) envolve computar $|K|^3$ expressões regulares $R(i, j, K)$. Entretanto, o *comprimento*

dessas expressões é, no pior caso, exponencial: durante cada iteração no índice K, o comprimento de cada expressão regular é grosseiramente *triplicado*, como ocorre no caso da concatenação das três expressões regulares da iteração anterior. As expressões regulares resultantes podem ter comprimento da ordem de $3^{|K|}$ – uma função exponencial do tamanho do autômato.

O algoritmo de minimização da seção anterior que, dado qualquer autômato determinístico $M = (K, \Sigma, \delta, s, F)$, computa um autômato finito determinístico equivalente com o menor número de estados, é *polinomial*. Ele é finalizado em, no máximo, $|K|-1$ iterações, cada qual determinando para cada par de estados, se eles estão relacionados por $\equiv_n$; esse teste gasta somente $O(|\Sigma|)$ operações elementares, tais como testar se dois estados pertencem ou não a uma relação de equivalência $\equiv_{n-1}$ previamente calculada. Portanto, a complexidade total do algoritmo de minimização será $O(|\Sigma|\,|K|^3)$ – um polinômio.

Dados dois geradores de linguagens ou dois aceitadores de linguagem, uma questão natural e interessante é determinar se eles são ou não *equivalentes*, isto é, se eles geram (ou aceitam) a mesma linguagem ou não. Se os dois aceitadores forem autômatos finitos determinísticos, o algoritmo de minimização de estados também proporciona uma solução para o problema da equivalência: dois autômatos finitos determinísticos são equivalentes se, e somente se, *seus autômatos padrão forem idênticos*. Isso ocorre porque o autômato padrão depende somente da linguagem aceita, constituindo, portanto, uma padronização útil para o teste de equivalências. Verificar se dois autômatos determinísticos são idênticos não é um problema difícil de isomorfismo, pois seus estados podem ser sucessivamente identificados, partindo-se dos iniciais, com a ajuda dos rótulos encontrados nas transições.

Em contraste, o único modo de afirmar que dois *autômatos não-determinísticos* ou duas expressões regulares são equivalentes é convertê-los em dois autômatos finitos determinísticos e, então, testar estes quanto à equivalência. Este algoritmo é, naturalmente, exponencial.

Resumimos abaixo nossa discussão dos problemas algorítmicos relacionados às linguagens regulares e suas representações:

Teorema 2.6.1:

(a) Há um algoritmo exponencial que, dado um autômato finito não-determinístico, constrói um autômato finito determinístico equivalente.

(b) Há um algoritmo polinomial que, dada uma expressão regular, constrói um autômato finito não-determinístico equivalente.

(c) Há um algoritmo exponencial que, dado um autômato finito não-determinístico, constrói uma expressão regular equivalente.

(d) Há um algoritmo polinomial que, dado um autômato finito determinístico, constrói um autômato finito determinístico equivalente com o menor número possível de estados.

(e) Há um algoritmo polinomial que, dados dois autômatos finitos determinísticos, decide se eles são equivalentes.

(f) Há um algoritmo exponencial que, dados dois autômatos finitos não-determinísticos, decide se eles são equivalentes; o mesmo ocorre em relação à equivalência de duas expressões regulares.

Sabemos que a complexidade exponencial em (a) e (c) acima é inevitável, porque, como o indicam o Exemplo 2.2.5 e o Problema 2.3.8, o algoritmo

(em (a), o autômato determinístico; em (c), a expressão regular equivalente) pode ser exponencial. Há, porém, três importantes questões que permanecem não-resolvidas no Teorema 2.6.1:

(1) Há um algoritmo polinomial para determinar se dois autômatos finitos não-determinísticos são equivalentes ou então se a complexidade exponencial em (f) é inerente?

(2) Podemos determinar, em tempo polinomial, o autômato não-determinístico, que apresenta o mínimo número de estados, que é equivalente a um dado autômato não-determinístico? Podemos certamente fazer isto em tempo exponencial testando todos os possíveis autômatos não-determinísticos com menos estados que o autômato dado, quanto à equivalência em cada caso, utilizando o algoritmo exponencial em (f).

(3) Mais intrigantemente, suponha que nos seja dado um autômato finito não-determinístico e que desejemos determinar o autômato finito determinístico equivalente que tenha o mínimo número de estados. Isso pode ser realizado combinando os algoritmos apresentados em (a) e (d) no Teorema 2.6.1. Entretanto, o número de passos pode ser exponencial sobre o tamanho do autômato não-determinístico dado, mesmo que o resultado final possa ser pequeno – simplesmente porque o resultado intermediário, o autômato determinístico não-otimizado, produzido pela construção dos subconjuntos, pode ter um número de estados exponencialmente maior que o necessário. Há um algoritmo que produz *diretamente* o autômato determinístico equivalente, com o número mínimo de estados, em tempo limitado por um polinômio sobre a entrada e a saída final?

Como deveremos ver no Capítulo 7 (sobre *completude* $\mathcal{NP}$), suspeitamos fortemente que essas três questões têm respostas negativas, *embora não se saiba ainda como prová-lo.*

Autômatos finitos como algoritmos

É possível apresentar uma relação muito básica entre autômatos finitos determinísticos e algoritmos: um autômato finito determinístico M representa um algoritmo eficiente para decidir se uma dada cadeia pertence ou não a $L(M)$. Por exemplo, o autômato finito determinístico da Figura 2-23 pode ser representado como o seguinte algoritmo:

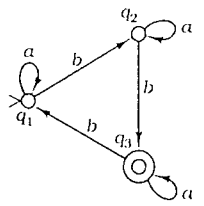


Figura 2-23

q_1: seja σ := próximo símbolo;
se σ = fim de arquivo então rejeite;
se não, se $\sigma = a$, então vá para q_1;
se não, se $\sigma = b$, então vá para q_2;
q_2: seja σ := próximo-símbolo;
se σ = fim de arquivo, então rejeite;
se não, se $\sigma = a$ então vá para q_2;
se não, se $\sigma = b$ então vá para q_3;
q_3: seja σ := próximo-símbolo;
se σ = fim de arquivo, então aceite;
se não, se $\sigma = a$ então vá para q_3;
se não, se $\sigma = b$ então vá para q_1;

Para representar um autômato finito determinístico $M = (K, \Sigma, \delta, s, F)$ na forma de um algoritmo, para cada estado em K temos $|\Sigma| + 2$ instruções, das quais a primeira obtém o próximo símbolo de entrada e cada uma das outras é responsável por realizar a ação correta para um valor particular do símbolo de entrada – ou, para o caso no qual alcançamos o fim da cadeia de entrada, um evento que chamamos "fim de arquivo". Isso pode ser expresso formalmente como segue:

Teorema 2.6.2: *Se L é uma linguagem regular, então há um algoritmo que, dado a cadeia $w \in \Sigma^*$, testa se ela está em L, em tempo $O(|w|)$.*

Quanto aos autômatos finitos não-determinísticos, eles constituem definitivamente uma poderosa simplificação notacional, e a forma mais natural e direta de representar expressões regulares como autômatos (lembre-se das construções usadas na demonstração do Teorema 2.3.1), mas eles não correspondem obviamente a algoritmos. Naturalmente, podemos sempre transformar um dado autômato finito não-determinístico no seu equivalente determinístico pela construção dos subconjuntos utilizados na prova do Teorema 2.2.1, mas a construção em si (e o autômato resultante) pode ser exponencial. A questão que surge, então, é: *podemos "executar" diretamente o autômato finito não-determinístico*, de modo semelhante àquele como executamos os determinísticos? Mostraremos a seguir que isso é possível com uma modesta perda em velocidade.

Lembre-se da idéia subjacente à construção dos subconjuntos: após ter lido parte da entrada, um autômato não-determinístico pode estar em qualquer dos *conjuntos* de *estados*. A construção dos subconjuntos computa todos os possíveis conjuntos de estados. Entretanto, quando estamos interessados apenas em passar uma cadeia através do autômato, talvez a melhor idéia seja *calcular os conjuntos de estados* apenas quando necessário, de acordo com a cadeia de entrada.

Concretamente, seja $M = (K, \Sigma, \Delta, s, F)$ um autômato finito não-determinístico, e considere o seguinte algoritmo:

$S_0 := E(s)$, $n := 0$;
repita o seguinte
  faça $n := n + 1$ e seja σ o n-ésimo símbolo de entrada;
  se $\sigma \neq$ fim de arquivo, então

    $S_n := \cup\{E(q)$: para algum $p \in S_{n-1}$, $(p, \sigma, q) \in \Delta\}$
  até σ = fim de arquivo
  se $S_{n-1} \cap F \neq \emptyset$ então aceite. Se não, rejeite a cadeia

Aqui $E(q)$ significa o conjunto $\{p: (q, \varepsilon) \vdash^*_M (p, \varepsilon)\}$, como na construção dos subconjuntos.

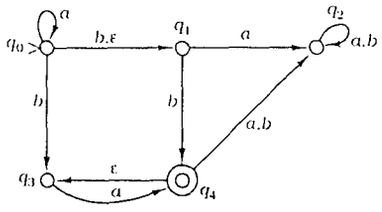


Figura 2-24

Exemplo 2.6.1: Vamos "executar", utilizando esse algoritmo, o autômato finito não-determinístico da Figura 2-24 sobre a cadeia de entrada *aaaba*. Os diversos valores, para o conjunto S_n, são mostrados abaixo.

$S_0 = \{q_0, q_1\}$.
$S_1 = \{q_0, q_1, q_2\}$.
$S_2 = \{q_0, q_1, q_2\}$.
$S_3 = \{q_0, q_1, q_2\}$.
$S_4 = \{q_2, q_3, q_4\}$.
$S_5 = \{q_2, q_3, q_4\}$.

A máquina acaba aceitando a entrada *aaaba*, porque S_5 contém um estado final. ✧

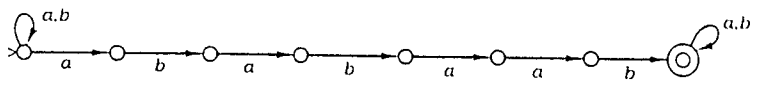
É fácil provar, por indução no comprimento da cadeia de entrada, que esse algoritmo mantém no conjunto S_n todos os estados de M que poderiam ser alcançados pela leitura dos primeiros n símbolos da entrada. Em outras palavras, ele simula o autômato finito determinístico equivalente *sem jamais construí-lo*. Seus requisitos de tempo são modestos, como afirma o seguinte teorema (para uma prova do limite de tempo, bem como para um aprimoramento desse estudo, veja o Problema 2.6.2).

Teorema 2.6.3: *Se $M = (K, \Sigma, \Delta, s, F)$ é um autômato finito não-determinístico, então há um algoritmo que, dada a cadeia $w \in \Sigma^*$, testa se ela está em $L(M)$, em tempo $O(|K|^2 |w|)$.*

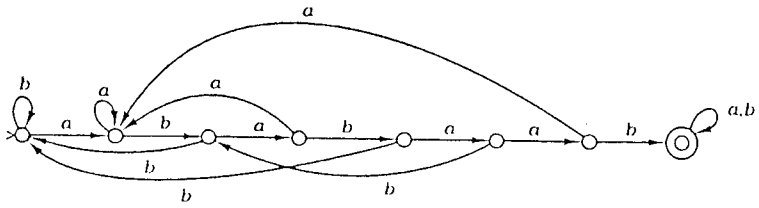
Seja dada uma *expressão regular* α sobre o alfabeto Σ, e suponha que desejamos determinar se uma certa cadeia $w \in \Sigma^*$ está ou não em $L(\alpha)$, a linguagem gerada por α. Como α pode ser facilmente transformado em um autômato finito não-determinístico M equivalente, de tamanho comparável ao de α (lembre-se das construções utilizadas no Teorema 2.3.1), o algoritmo acima também poderá ser utilizado para responder a essas questões.†

† Para o leitor familiarizado com o sistema operacional Unix, esse algoritmo baseia-se nos mesmos princípios utilizados nos comandos *grep* e *egrep*.

Um problema computacional relacionado, que também pode ser resolvido por métodos baseados em autômatos finitos, é o da **correspondência de cadeia**, questão muito importante nos sistemas computacionais e suas aplicações. Seja $x \in \Sigma^*$ uma cadeia a que chamamos *cadeia de referência*. Desejamos implementar um algoritmo eficiente que, dada qualquer cadeia w, o texto (presumivelmente muito mais longo que x) determina se x ocorre ou não como subcadeia de w. Não estamos interessados em um algoritmo que tome x e w como entradas e nos informa se x ocorre em w. Ao contrário, desejamos que tal algoritmo (A_x), se especialize em descobrir todas as cadeias mais longas em que x ocorra. Nossa estratégia consiste em projetar um autômato finito que reconhece a linguagem $L_x = \{w \in \Sigma^*: x \text{ é uma subcadeia de } w\}$.



(a)



(b)

Figura 2-25

É trivial projetar um autômato finito não-determinístico capaz de aceitar L_x. Por exemplo, se $\Sigma = \{a, b\}$ e $x = ababaab$, o correspondente autômato finito não-determinístico é mostrado na Figura 2-25(a). Porém, para transformar esse autômato em um algoritmo útil, deveríamos recorrer à simulação direta do Teorema 2.6.3 – com seu tempo de execução $O(|x|^2|w|)$, polinomial, mas muito lento para essa aplicação – ou convertê-lo em um autômato determinístico equivalente – uma construção que sabemos ser, potencialmente, exponencial. Felizmente, no caso de o autômato não-determinístico surgir nas aplicações de correspondência de cadeias, a construção dos subconjuntos é *sempre eficiente*, e o autômato determinístico M_x resultante tem exatamente o mesmo número de estados que o autômato não-determinístico original (veja a Figura 2-25(b)). Ele é claramente o seu autômato mínimo equivalente. Esse autômato M_x representa, portanto, um algoritmo para testar se $w \in L_x$ em tempo $O(|w|)$, para qualquer cadeia $w \in \Sigma^*$.

Infelizmente, esse algoritmo tem uma desvantagem que o torna inadequado às diversas aplicações práticas da correspondência de cadeias. Em aplicações reais, o alfabeto Σ subjacente tem várias dezenas, freqüentemente centenas de símbolos. Um autômato finito determinístico que representa um algoritmo deve executar, para cada símbolo de entrada, uma longa seqüência de comandos **if**, uma para cada símbolo do alfabeto (lembre-se do primeiro algoritmo desta subseção). Em outras palavras, a notação O, relativa ao tempo de execução $O(|w|)$ do algoritmo, "oculta" uma constante potencialmente grande: o tempo de execução é de fato $O(|\Sigma||w|)$. Uma maneira mais inteligente de contornar essa situação pode ser explorada no Problema 2.6.3.

Problemas para a Seção 2.6

2.6.1 Mostre que as expressões regulares seguintes não representam a mesma linguagem: $aa(a \cup b)^* \cup (bb)^*a^*$ e $(ab \cup ba \cup a)^*$ Para isso:
(a) submeta-as a um algoritmo geral; e
(b) encontre uma cadeia que seja gerada por uma delas, mas não pela outra.

2.6.2 (a) qual é a seqüência de S_i's produzida no caso de ser apresentada ao finito não-determinístico do Exemplo 2.6.1 a entrada *bbabbabba*?
(b) Prove que o algoritmo que executa um autômato finito não-determinístico com m estados sobre uma entrada de comprimento n opera em tempo $O(m^2 n)$.
(c) Seja dado um autômato finito não-determinístico que tenha, no máximo, p transições de cada estado. Mostre que é possível construir para ele um algoritmo $O(mnp)$.

2.6.3 Seja Σ um alfabeto, $x = a_1 \ldots a_n \in \Sigma^*$. Considerando o autômato finito não-determinístico $M_x = (K, \Sigma, \Delta, s, F)$, onde $K = \{q_0, q_1, \ldots, q_n\}$, $\Delta = \{(q_{i-1}, a_i, q_i): i = 1, \ldots, n\} \cup \{(q_i, a, q_j): a \in \Sigma, i \in \{0, n\}\}$ (lembre-se da Figura 2-25),
(a) mostre que $L(M_x) = \{w \in \Sigma^*: x \text{ é uma subcadeia de } w\}$.
(b) mostre que o autômato finito determinístico M'_x, que tem o mínimo número de estados e é equivalente a M_x, também tem $n + 1$ estados. Qual é o tempo exigido (em função de n), no pior caso, para sua construção?
(c) mostre que há um autômato finito não-determinístico M''_x equivalente a M_x, também com $n + 1$ estados $\{q_0, q_1, \ldots, q_n\}$ e com a seguinte importante propriedade: *cada estado, exceto q_0 e q_n, tem exatamente duas transições fora dele, uma das quais é em vazio.* (*Sugestão:* Substitua cada transição para trás no autômato finito determinístico da Figura 2-25 por uma transição em vazio apropriada; generalize.)
(d) Argumente que M''_x contorna o problema da constante oculta $|\Sigma|$ discutida no último parágrafo da seção 2.6.
(e) Apresente um algoritmo para construir M''_x de x. Qual é a complexidade desse algoritmo, em função de n?
(f) Conceba um algoritmo $O(n)$ para o problema em (e) acima. (*Sugestão:* Suponha que as transições em vazio de M''_x são $(q_i, \varepsilon, q_{f(i)})$, $i = 1, \ldots, n - 1$. Mostre como computar $f(i)$ com base nos valores $f(j)$ para $j < i$. Uma inteligente análise "amortizada" dessa computação fornece o limite $O(n)$.)
(g) Sejam $\Sigma = \{a, b\}$ e $x = aabbaab$. Construa M_x, M'_x e M''_x. Execute cada um desses autômatos usando como entrada a cadeia *aababbaaabbaaabbaabb*.

REFERÊNCIAS

Algumas das primeiras publicações sobre autômatos finitos foram:

- G. H. Mealy "A method for synthesizing sequential circuits", *Bell System Technical Journal*, 34, 5, pp. 1045-1079, 1955, e
- E. F. Moore "Gedanken experiments on sequential machines", *Automata Studies*, ed. C. E. Shannon and J. McCarthy, pp. 129-53, Princeton: Princeton University Press, 1956.

A obra clássica sobre autômatos finitos (contendo o Teorema 2.2. 1) é:

- M. O. Rabin and D. Scott "Finite automata and their decision problems", *IBM Journal of Research and Development*, 3, pp. 114-25, 1959.

O teorema 2.3.2, afirmando que autômatos finitos aceitam linguagens regulares, é de Kleene:

- S. C. Kleene "Representation of events by nerve nets", in *Automata Studies*, ed. C. E. Shannon and J. McCarthy, pp. 3-42, Princeton: Princeton University Press, 1956.

Nossa prova desse teorema segue a obra:

- R. McNaughton and H. Yamada "Regular expressions and state graphs for automata", *IEEE Transactions on Electronic Computers*, EC-9, 1 pp. 39-47, 1960.

O teorema 2.4.1 (o "lema do bombeamento") é da autoria de:

- V. Bar-Hillel, M. Perls, and E. Shamir "On formal properties of simple phrase structure grammars", Zeitschrift fur Phonetik, Sprachwissenschaft, und Kommunikationsforschung, 14, pp. 143-172, 1961.

Os transdutores de estados finitos (Problema 2.1.4) foram apresentados pela primeira vez em:

- S. Ginsburg "Examples of abstract machines", IEEE Transactions on Electronic Computers, EC-11, 2, pp. 132-135, 1962.

Os autômatos de estados finitos de duas fitas (Problemas 2.1.5 e 2.4.7) foram examinados em:

- M. Bird "The equivalence problem for deterministic two-tape automata", *Journal of Computer and Systems Sciences*, 7, pp. 218-236, 1973.

O teorema de Myhill-Nerode (Teorema 2.5.2) é da autoria de:

- A. Nerode "Linear automaton transformations", *Proc. AMS*, 9, pp.541-544, 1958.

O algoritmo para minimizar autômatos finitos provém do trabalho de Moore, citado acima. Um algoritmo mais eficiente é encontrado em:

- J. E. Hopcroft "An $n \log n$ algorithm for minimizing the states in a finite automaton", in *The Theory of Machines and Computations*, ed. Z. Kohavi, New York: Academic Press, 1971.

A simulação de autômatos não-determinísticos (Teorema 2.6.3) baseia-se em:

- K. Thompson "Regular Expression search algorithms", *Communications of the ACM*, 11, 6, pp. 419-422, 1968.

O algoritmo rápido de correspondência de padrões, do Problema 2.6.3, encontra-se em:

- D. E. Knuth, J. H. Morris, Jr, V. R. Pratt "Fast pattern matching in strings", *SIAM J. on Computing*, 6, 2, pp. 323-350, 1976.

A equivalência dos autômatos finitos de mão única com os de mão dupla (Problema 2.5 - 4) é mostrada em:

- J. C. Shepherdson "The reduction of two-way automata to one-way automata", *IBM Journal of Research and Development*, 3, pp. 198-200, 1959.

3 Linguagens livres de contexto

3.1 GRAMÁTICAS LIVRES DE CONTEXTO

Imagine-se um processador de linguagem. Você poderia reconhecer uma sentença válida em português, ao ouvir: "o gato está no chapéu", você entenderia que essa frase está no mínimo sintaticamente correta (seja ou não verdade aquilo que ela diz), mas "chapéu o no está gato" não teria sentido. De fato, você utiliza um idioma para fazer isso e pode dizer, imediatamente, ao ler uma sentença, se ela foi ou não formada de acordo com as regras geralmente aceitas para estruturar sentenças. Sob esse enfoque, você está atuando como um **reconhecedor de linguagem**: um dispositivo que reconhece cadeias válidas. Os autômatos finitos do capítulo anterior são formalizados como uma possível forma de um reconhecedor de linguagem.

Mas você também é capaz de produzir sentenças válidas da língua portuguesa. Novamente, a razão por que você iria querer fazer isso e a forma como observa um idioma para poder fazê-lo não são do nosso interesse; mas o fato é que você de fato fala ou escreve sentenças, e, em geral, estas estão sintaticamente corretas (mesmo quando falsas). Sob esse aspecto, você está atuando como um **gerador de linguagem**. Nesta seção, iremos estudar alguns tipos de geradores de linguagens formais. Um dispositivo dessa natureza começa a operar ao receber algum sinal "inicia", e passa então a construir uma cadeia da linguagem. Sua operação não fica completamente determinada *a priori*, mas é, não obstante, condicionada por um conjunto fixo de regras. Ao final, esse processo pára, tendo o dispositivo produzido uma cadeia completa. A linguagem definida pelo dispositivo é o conjunto de todas as possíveis cadeias que ele pode produzir.

Um reconhecedor ou um gerador para uma linguagem natural qualquer, como, por exemplo, o idioma inglês, não é uma coisa fácil de construir; de fato, projetar esses dispositivos para grandes subconjuntos de linguagens naturais tem sido um permanente desafio na pesquisa de ponta há várias décadas. Contudo, a idéia de uma linguagem geradora tem alguma força explicativa em uma tentativa de discutir a linguagem humana. Mais importante para nós, entretanto, é a teoria dos geradores de linguagens "artificiais" formais, incluindo as linguagens regulares e a importante classe das linguagens "livres de contexto" a serem apresentadas adiante. Essa teoria complementará elegantemente o estudo dos autômatos, que reconhece linguagens e é também de valor prático na especificação e na análise das linguagens de computador.

Expressões regulares podem ser vistas como geradores de linguagem. Por exemplo, considere-se a expressão regular $a(a^* \cup b^*)b$. Uma descrição verbal de como gerar uma cadeia, de acordo com essa expressão, seria a seguinte: