Capítulo 4: Autômatos finitos e expressões regulares

– com Luiz Carlos Castro Guedes

4.1 - Introdução

Neste capítulo estudaremos uma máquina (um procedimento aceitador, ou reconhecedor), chamada *autômato finito (af)*. A palavra *finito* é incluída no nome para ressaltar que um af só pode conter uma quantidade finita e limitada de informação, a qualquer momento. Essa informação é representada por um estado da máquina, e só existe um número finito de estados.

Essa restrição faz com que o af seja severamente limitado na classe de linguagens que pode reconhecer, composta apenas pelas linguagens regulares, como mostraremos neste capítulo.

Duas versões do af são estudadas aqui: o *af determinístico (afd)* e o *af não determinístico (afnd).* Este capítulo mostra que uma linguagem regular pode ser definida de quatro formas:

- através de uma gramática regular (definição);
- através de um afd que reconhece a linguagem;
- idem, através de um afnd;
- através de uma *expressão regular*, um mecanismo a ser introduzido com essa expressa finalidade.

4.2 - Autômato finito determinístico

Como observado acima, a informação que um af guarda sobre a entrada (mais precisamente sobre a parte da entrada já lida) é representada por um *estado*, escolhido em um conjunto finito de estados. A definição formal de automato finito, na sua versão determinística é dada a seguir.

Definição. Um *Autômato Finito Determinístico (afd)* M, sobre um alfabeto Σ é um sistema $(K, \Sigma, \delta, i, F)$, onde

K é um *conjunto de estados* finito, não vazio;
Σ é um *alfabeto de entrada* (finito)
$\delta: K\times\Sigma \rightarrow K$ é a *função de transição*
$i \in K$ é o *estado inicial*
$F \subseteq K$ é o conjunto de *estados finais*.

O nome *determinístico* faz referência ao fato de que δ é uma função (também chamada *função próximo-estado*), que determina precisamente o próximo estado a ser assumido quando a máquina M se encontra no estado q e lê da entrada o símbolo a: o estado $\delta(q, a)$.

De forma simplificada, podemos dizer que um afd aceita uma cadeia se, partindo do *estado inicial*, e mudando de estado de acordo com a *função de transição*, o afd atinge um *estado final* ao terminar de ler a cadeia. Uma das maneiras de visualizar o funcionamento de um afd é através de um *controle finito* que lê símbolos de uma *fita* de entrada (onde se encontra a cadeia de entrada), sequencialmente, da esquerda para a direita. Os elementos do conjunto de estados K representam os estados possíveis do controle finito. A operação se inicia no estado inicial i, lendo o primeiro símbolo da fita de entrada. Por conveniência, considera-se que a cabeça de leitura se move sobre a fita, ao contrário do que seria de se esperar.

A Figura 4.1 representa um afd cujo controle está no estado q, e que está lendo o quarto símbolo da cadeia de entrada, um b.

Fig. 4.1 - Autômato Finito

Exemplo 1: Considere o afd M = (K, Σ, δ, i, F), onde temos

K = { q0, q1, q2, q3 }
Σ = { a, b }
i = q0
F = { q3 }

e onde a função de transição δ: { q0, q1, q2, q3 } × { a, b } → { q0, q1, q2, q3 } é dada pela tabela abaixo

δ	a	b
q0	q1	q2
q1	q0	q3
q2	q3	q0
q3	q2	q1

Alternativamente, podemos representar o afd M por um *diagrama de transições*, ou *diagrama de estados*, como o da Fig. 4.2. Note que o diagrama de transições determina completamente o automato M, através de algumas convenções:

- os estados são os nós do grafo, ou seja, K = { q0, q1, q2, q3 };
- o estado inicial é indicado pela seta, ou seja, i = q0;
- os estado finais são indicados pelo círculo duplo: q3 é o único estado final, ou seja, F = { q3 };

- as transições são as indicadas pelas arestas: δ(q0, a) = q1, δ(q0, b) = q2, δ(q1, a) = q0, etc, ou seja, δ é a mesma função representada pela tabela acima.

Cada estado de um af corresponde a uma determinada informação sobre a parte da cadeia de entrada já lida. No caso do exemplo, a informação pode ser descrita em frases curtas, mas isso nem sempre acontece. Para o estado q2, por exemplo, podemos dizer

"se o estado atingido é q2,
o número de símbolos a já lidos é par, e
o número de símbolos b já lidos é ímpar".

(Note que 0 é par.)

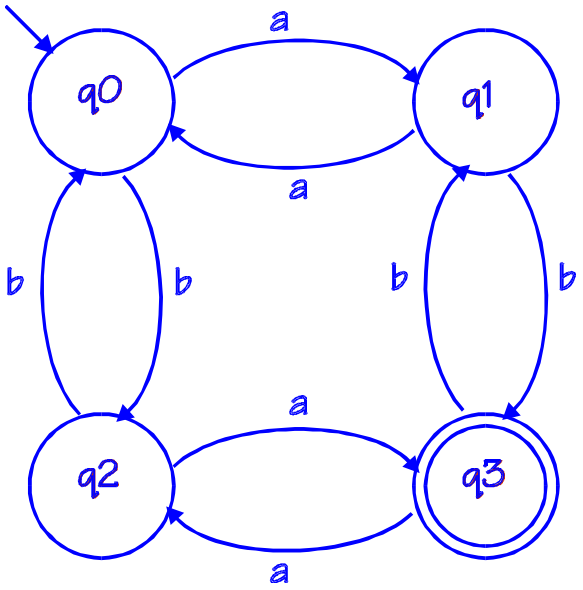


Fig. 4.2 - afd para o Exemplo 1

Em resumo, temos:

	número de a's	número de b's
q0	par	par
q1	ímpar	par
q2	par	ímpar
q3	ímpar	ímpar

A linguagem aceita ou reconhecida por M (ver definição abaixo) é a linguagem formada pelas cadeias em que os números de a's e de b's são ambos ímpares. Isso se deve ao fato de que o único estado final é q3.

Por exemplo, a cadeia abaa é da linguagem de M, porque, com essa cadeia, os seguintes estados são atingidos: q0, q1, q3, q2, q3. Como o último estado é final, a cadeia é aceita.

ð

A linguagem de um afd. Para definir a linguagem L(M), a linguagem das cadeias aceitas ou reconhecidas pelo afd M, podemos definir inicialmente uma *configuração* de M como sendo um par $(q, x) \in K \times \Sigma^*$, composto pelo estado corrente (o estado atual da máquina) e pela cadeia x, a parte da entrada que *ainda não foi lida*. Como observado, o

estado representa a parte já lida da cadeia de entrada. A configuração (i, x) é a *configuração inicial* de M para a cadeia x; qualquer configuração (q, ε) é uma *configuração final* se q ∈ F. A mudança de configuração é caracterizada pela relação |—, definida abaixo:

(q, ax) |— (p, x) se e somente se $\delta(q, a) = p$

ou seja, se tivermos $\delta(q, a) = p$, e se M estiver no estado q, lendo um símbolo a da cadeia de entrada, M moverá a cabeça de leitura uma posição para a direita e irá para o estado p. O símbolo a, depois de lido, não precisa mais aparecer na configuração. Podemos agora definir a linguagem L(M) por

$L(M) = \{ x \in \Sigma^* \mid (i, x) \vdash^* (f, \varepsilon), \text{ com } f \in F \}$

Como em casos anteriores, $\vdash^*$ indica o fechamento reflexivo-transitivo da relação, no caso a relação |— de mudança de configuração, indicando que a configuração final pode ser atingida em zero ou mais passos.

Exemplo 1: (continuação) Para mostrar que abaa ∈ L(M), basta observar que

(q0, abaa) |— (q1, baa) |— (q3, aa) |— (q2, a) |— (q3, ε),

porque q3 é final. Por outro lado, como

(q0, abab) |— (q1,bab) |— (q3, ab) |— (q2, b) |— (q0, ε),

abab não pertence a L(M).

ð

Uma caracterização alternativa de L(M) pode ser baseada em uma extensão $\hat{d}$ da função δ, feita de forma a aceitar cadeias no segundo argumento, isto é com domínio $K \times \Sigma^*$ em vez de $K \times \Sigma$. Pode-se definir a nova função $\hat{d} : K \times \Sigma^* \to K$ por

$$\hat{d}(q, e) = q, \quad \forall q \in K$$

$$\hat{d}(q, ax) = \hat{d}(d(q, a), x), \quad \forall q \in K, \forall x \in \Sigma^*, \forall a \in \Sigma.$$

Fato: A função $\hat{d}$ é realmente uma *extensão* de δ, isto é, sempre que δ é definida, $\hat{d}$ também é, e tem o mesmo valor. Ou seja, se q ∈ K e a ∈ Σ, $\hat{d}(q, a) = \delta(q, a)$.

Dem. Exercício.

Fato: A função $\hat{d}$ e a relação |— se relacionam por

$\hat{d}(q, x) = p$ se e somente se $(q, x) \vdash^* (p, \varepsilon)$

Portanto, temos

$L(M) = \{x \in \Sigma^* \mid \hat{d}(q, x) \in F \}$

Demonstração: Exercício.

Exemplo 1: (continuação) Para mostrar que abaa ∈ L(M), basta ver que

$\hat{d}$ (q0, abaa) = $\hat{d}$ (q1, baa) = $\hat{d}$ (q3, aa) = $\hat{d}$ (q2, a) = $\hat{d}$ (q3, ε) = q3 ∈ F

Como $\hat{d}$ (q0, abab) = q0 ∉ F, abab ∉ L(M).

Exercício 1: Mostre que o afd M do Exemplo 1 aceita a linguagem

L(M) = { x ∈ {a, b}* | os números de a's e de b's em x são ímpares }

Sugestão: indução no comprimento de x.

Exercício 2: Mostre que a definição anterior de $\hat{d}$ pode ser substituída pela equivalente

$$\hat{d}(q, e) = q, \quad \forall q \in K$$

$$\hat{d}(q, xa) = d(\hat{d}(q, x), a), \quad \forall q \in K, \forall x \in \Sigma^*, \forall a \in \Sigma.$$

Exercício 3: Modifique a definição do af M do Exemplo 1, fazendo F= { q1, q2 } Descreva a linguagem aceita por M assim modificado.

Exercício 4: Descreva a linguagem do afd M dado pelo diagrama de estados da Fig. 4.3.

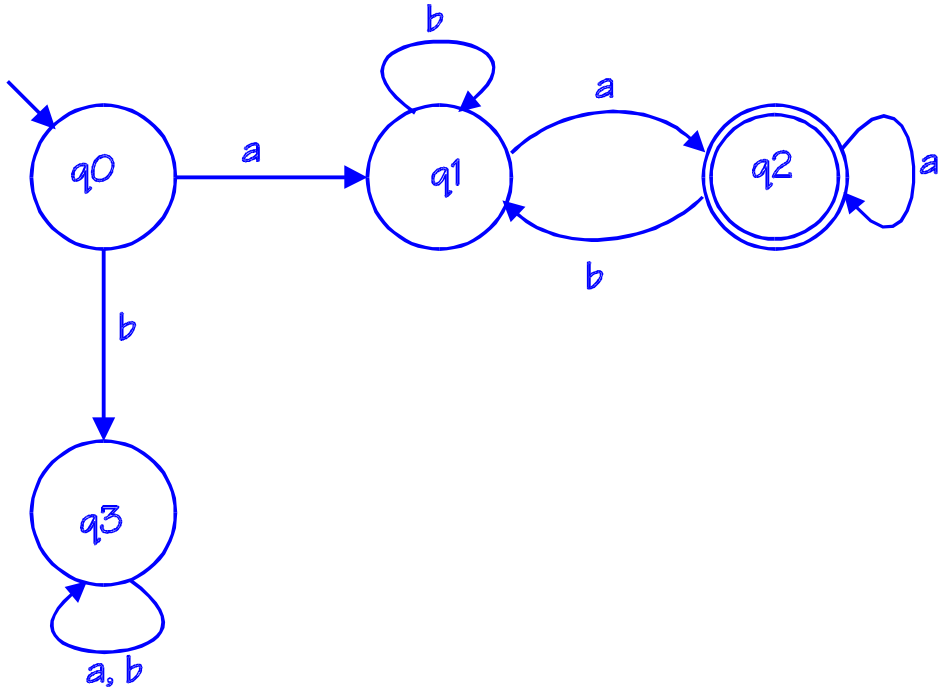


Fig. 4.3 - afd para o Exercício 4

Exercício 5: Descreva a linguagem do afd M = (K, Σ, δ, i, F), onde K={q0, q1, q2, q3}, Σ = { a, b }, i = q0, F = { q2 }, e δ dada pela tabela abaixo:

δ	a	b
q0	q1	q3
q1	q2	q0
q2	q3	q1
q3	q4	q2

ð

4.3 - Autômato finito não determinístico

Passaremos agora ao estudo do af não determinístico. Em oposição ao que acontece com o afd, a função de transição de um afnd não precisa determinar exatamente qual deve ser o próximo estado. Em vez disso, a função de transição fornece uma lista (um conjunto) de estados para os quais a transição poderia ser feita. Essa lista pode ser vazia, ou ter um número qualquer positivo de elementos.

Essa possibilidade de escolha entre vários caminhos a serem seguidos nos leva a modificar a definição de aceitação. Um afd aceita se "o último estado atingido é final"; mas um afnd aceita se "*existe uma sequência de escolhas tal que* o último estado atingido é final". Podemos alternativamente imaginar que o afnd "escolhe", "adivinha", o caminho certo para a aceitação, uma vez que a existência de escolhas erradas, que não levam a um estado final, é irrelevante.

Exemplo 2: Considere o afnd dado pelo diagrama da Fig. 4.4 e a cadeia de entrada ababa.

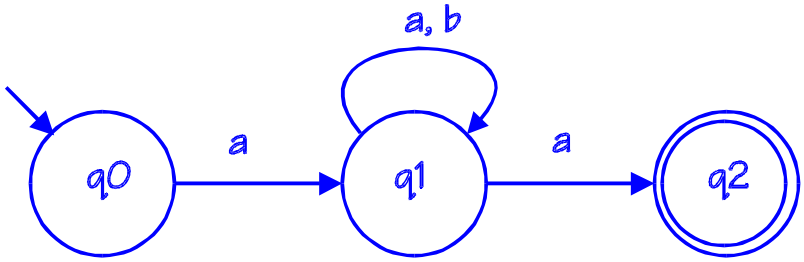


Fig. 4.4 - afnd para o Exemplo 2

A cadeia ababa é aceita, porque uma das possibilidades é a sequência de estados q0, q1, q1, q1, q1, q2. Naturalmente, com a mesma cadeia, poderíamos escolher a sequência q0, q1, q1, q1, q1, q1, que não leva a um estado final. Ou a sequência q0, q1, q1, q2, interrompida, porque q2 não prevê uma transição com o segundo b. Mas estes casos em que "o automato adivinhou errado" não criam problemas para a aceitação, porque "existe um caminho certo".

Este afnd aceita a linguagem das cadeias (de comprimento maior ou igual a 2), cujo primeiro e último símbolos são a, sendo os restantes quaisquer. (Compare este afnd com o afd de um dos exemplos anteriores, que aceita a mesma linguagem.)

ð

Definição. Formalmente, um *Autômato Finito não Determinístico (afnd)* M, sobre um alfabeto Σ é um sistema $(K, \Sigma, \delta, i, F)$, onde

K é um conjunto (finito, não vazio) de estados,
Σ é um *alfabeto de entrada* (finito),
$\delta: K\times(\Sigma \cup \{ \varepsilon \}) \rightarrow P(K)$ é a função de transição,
$i \in K$ é o *estado inicial*,
$F \subseteq K$ é o *conjunto de estados finais*.

A notação P(K) indica o conjunto "partes" de K (conjunto potência de K, ou, ainda, "powerset" de K), o conjunto de todos os subconjuntos de K.

Pela definição, portanto, δ é uma função que aceita como argumentos q e a, onde q é um estado e a pode ser um símbolo de Σ ou a cadeia vazia ε. Em qualquer caso, $\delta(q, a)$ é sempre um conjunto de estados, ou seja, um subconjunto de K.

Se tivermos $\delta(q, a) = \{p_1, p_2, ..., p_k\}$, entendemos que o autômato M, a partir do estado q, pode escolher um dos estados $p_1, p_2, ..., p_k$ para ser o próximo estado. Se $a = \varepsilon$, nenhum símbolo da entrada é lido; se $a \neq \varepsilon$, o símbolo a da entrada é lido. Podemos considerar o caso $a=\varepsilon$ como correspondendo a transições espontâneas: M muda de estado sem estímulo da entrada. Se tivermos $\delta(q, a) = \emptyset$, não há transições possíveis a partir do estado q com o símbolo a.

Definimos configurações para o caso do afnd da mesma forma que anteriormente. A mudança de configuração é dada pela relação |—, definida abaixo:

$(q, ax) \vdash (p, x)$ se e somente se $p \in \delta(q, a)$

Note que a pode ser a cadeia vazia, caso em que temos, particularizando,

$(q, x) \vdash (p, x)$ se e somente se $p \in \delta(q, \varepsilon)$

Podemos agora definir a linguagem L(M) por

$$L(M) = \{ x \in \Sigma^* \mid (i, x) \vdash^* (f, \varepsilon), \text{ com } f \in F \}$$

Exemplo 2: (continuação) Temos, para a mesma cadeia ababa de entrada,

$(q0, ababa) \vdash (q1, baba) \vdash (q1, aba) \vdash (q1, ba) \vdash (q1, a) \vdash (q2, \varepsilon)$

e, portanto, $ababa \in L(M)$. Temos também o "caminho errado"

$(q0, ababa) \vdash (q1, baba) \vdash (q1, aba) \vdash (q1, ba) \vdash (q1, a) \vdash (q1, \varepsilon)$

que leva à configuração não final $(q1, \varepsilon)$, e não permite nenhuma conclusão.

Cadeias como bab e abab não levam a configurações finais e não são aceitas. Da configuração (q0, bab) nenhuma configuração é atingível; para abab temos:

$(q0, abab) \vdash (q1, bab) \vdash (q1, ab) \vdash (q1, b) \vdash (q1, \varepsilon)$

Adicionalmente, temos um outro caminho

$(q0, abab) \vdash (q1, bab) \vdash (q1, ab) \vdash (q2, b)$

que também não atinge nenhuma configuração final. Assim, as cadeias bab e abab não são aceitas e não fazem parte de L(M).

ð

Exemplo 3: Considere o afnd M da Fig. 4.5. M aceita cadeias da forma c y c, onde c pode ser a ou b e y pode ser qualquer cadeia de a's e b's.

A cadeia ababa = c·y·c = a·bab·a é aceita por M, através da sequência de configurações abaixo, em que a primeira e a última transições são realizadas através de transições-ε.

(A, ababa) M lê ε e adivinha que c=a
|— (B, ababa) M lê a e confere que c=a
|— (C, baba) M lê b
|— (C, aba) M lê a e adivinha que este a faz parte de y
|— (C, ba) M lê b
|— (C, a) M lê a e adivinha que este a é o último c
|— (D, ε) M lê ε e adivinha que a cadeia acabou
|— (I, ε) M aceita

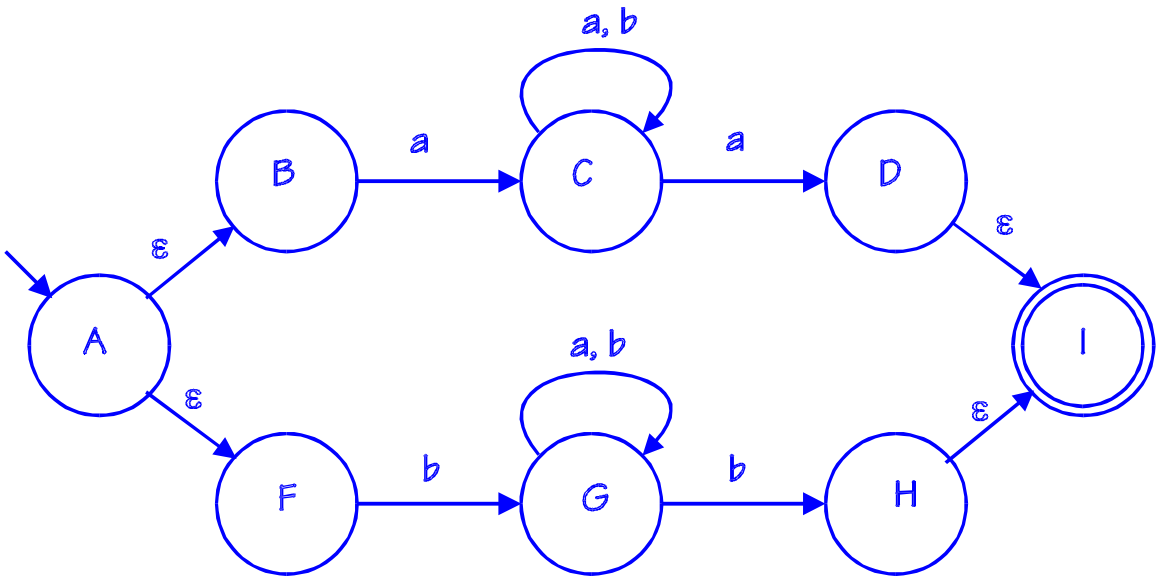


Fig. 4.5 - afnd para o Exemplo 3

Todas as configurações atingíveis (caminhos certos e errados) estão indicadas abaixo:

(A, ababa)
|— (B, ababa)
. |— (C, baba)
. |— (C, aba)
. |— (C, ba)
. . |— (C, a)
. . |— (C, ε) -- não aceita
. . |— (D, ε)
. . |— (I, ε) -- ok! aceita!
. |— (D, ba)
. |— (I, ba) -- bloqueado
|— (F, ababa) -- bloqueado

ð

Exercício 6: Considere a linguagem composta pelas cadeias no alfabeto {a, b} que contém a cadeia aaa ou a cadeia bb. Ou seja, a linguagem

L = { x y z | x, z ∈ {a, b}* e (y=aaa ou y=bb) }

Construa um afnd M que aceite L.

Sugestão: M adivinha se a cadeia de entrada contém aaa ou bb, e apenas verifica esse fato.

ð

4.4 - Equivalência dos afd's e dos afnd's

Mostraremos nesta seção que uma linguagem é aceita por um af determinístico se e somente se ela é aceita por um af não determinístico. A classe de linguagens reconhecidas por afd's e afnd's é a classe das linguagens regulares (ver seção 4.6).

Teorema: Toda linguagem reconhecida por um afd é reconhecida por um afnd.

Demonstração: Exercício.
Sugestão: Basta definir um afnd em que a única transição possível em cada caso é aquela especificada no afd.

ð

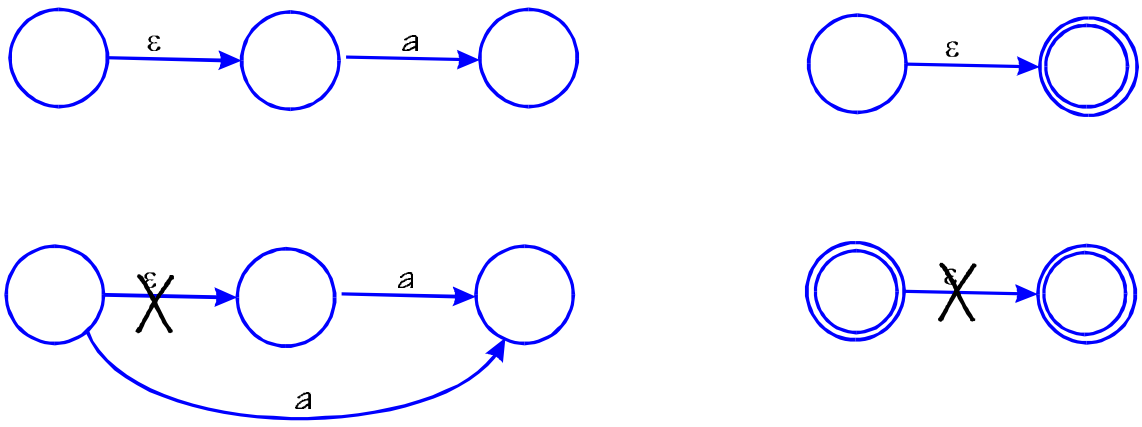
Teorema: Toda linguagem reconhecida por um afnd é reconhecida por um afd.
Demonstração: ver Lemas 1 e 2 abaixo.

ð

Lema 1: Toda linguagem reconhecida por um afnd é reconhecida por um afnd que não tem transições com ε.
Demonstração: Seja M = (K, Σ, δ, i, F) um afnd qualquer. Vamos construir um afnd M' = (K, Σ, δ', i, F') equivalente a M, isto é L(M') = L(M). Para isso vamos "simular" o efeito das transições com ε de duas maneiras:

- se tivermos a ∈ Σ, $\delta(p_1, \varepsilon) = p_2$, e $\delta(p_2, a) = q$, acrescentaremos a δ' uma transição de p_1 para q com a, ou seja, acrescentaremos q ao conjunto $\delta'(p_1, a)$;
- se tivermos $\delta(p_1, \varepsilon) = p_2$, e $p_2 \in F$, acrescentaremos p_1 a F.

(ver figura abaixo)



Isso deve ser feito repetidamente enquanto novas transições forem acrescentadas a δ', e enquanto novos estados forem acrescentados a F. Após isso, retiramos de δ as transições com ε, e chamamos os resultados de δ' e F'.

ð

Exemplo 4: Considere o afnd M do Exemplo 3 (Fig. 4.5). A construção descrita na prova do Lema 1 permite construir o afnd equivalente M' (Fig. 4.6), que não tem

transições com ε. Note que M' tem estados inúteis: B, F e I passaram a ser inacessíveis a partir do estado inicial.

Para aceitar a cadeia ababa, as configurações de M estão na tabela a seguir:

Configurações de M	Configurações de M'
(A, ababa)	(A, ababa)
(B, ababa)	---
(C, baba)	(C, baba)
(C, aba)	(C, aba)
(C, ba)	(C, ba)
(C, a)	(C, a)
(D, ε)	(D, ε) --- final
(I, ε) --- final	---

ð

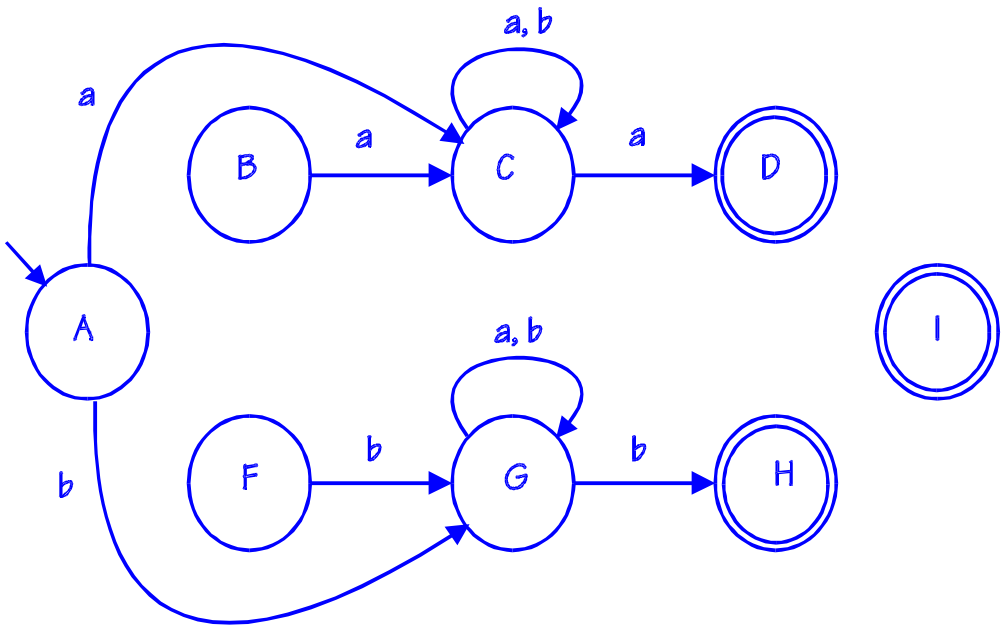


Fig. 4.6 - afnd sem transições-ε

Lema 2: Toda linguagem aceita por um afnd sem transições com ε é aceita por um afd.

Demonstração. Seja M = (K, Σ, δ, i, F) um afnd sem transições com ε. Vamos construir um afd M' = (K', Σ, δ', i', F'), equivalente a M, isto é, M' também aceita L. A idéia da demonstração é que os estados de M' são conjuntos de estados de M: a cada momento o estado de M' contém todos os estados em que M poderia se encontrar. Desta maneira, M' pode seguir ao mesmo tempo todos os caminhos percorridos por M. Temos:

K'= P(K); estados de M' são conjuntos de estados de M
i' = { i } o estado inicial de M' contém apenas o estado inicial de M
F' = { $Q \subseteq K \mid Q \cap F \neq \varnothing$ }
os estados finais de M' são os conjuntos de estados de M que contém pelo menos um estado final de M
$\delta': K' \times \Sigma \rightarrow K'$

A função δ' deve cobrir todas as possibilidades: δ'(Q, a) deve incluir todos os estados em todos os conjuntos δ(q, a), para cada q em Q, ou seja,

$$\delta'(Q,a) = \bigcup_{q \in Q} \delta(q,a)$$

para cada a ∈ Σ. A aceitação nas duas máquinas se dá de forma paralela:

M aceita x, e temos em M (i, x) |—* (f, ε), com f ∈ F
M' aceita x, e temos em M' (i', x) |—* (Q, ε), com $Q \cap F \neq \varnothing$

A ligação entre as duas sequências de configurações é feita pelo estado f ∈ Q.

O restante da demonstração consiste na prova de que dada uma das sequências de configurações, é possível construir a outra, e vice-versa.

Observamos que em geral não é necessário levar em consideração todos os estados de M', bastando apenas considerar aqueles que são acessíveis a partir de i'. Se M tem n estados, M' tem um máximo teórico de 2^n estados, mas em geral apenas uma fração desses estados é acessível a partir do estado inicial i'.

ð

Exemplo 4 (continuação): Podemos construir um afd M" a partir de M', como descrito na demonstração do Lema 2. M" será equivalente a M (Exemplo 3) e a M'.

Temos: i" = { A }. A tabela abaixo mostra a função δ". Note que os 251 estados não acessíveis a partir de { A } foram ignorados. O afd pode ser visto também na Fig. 4.7.

δ"	a	b
{ A }	{ C }	{ G }
{ C }	{ C, D }	{ C }
{ G }	{ G }	{ G, H }
{ C, D }	{ C, D }	{ C }
{ G, H }	{ G }	{ G, H }

Os estados finais de M" que precisam ser considerados são, portanto, {C, D} e {G, H}, que contém os estados finais D e H de M'. Para comparação, a tabela abaixo apresenta as configurações assumidas por M, M' e M" na aceitação de ababa.

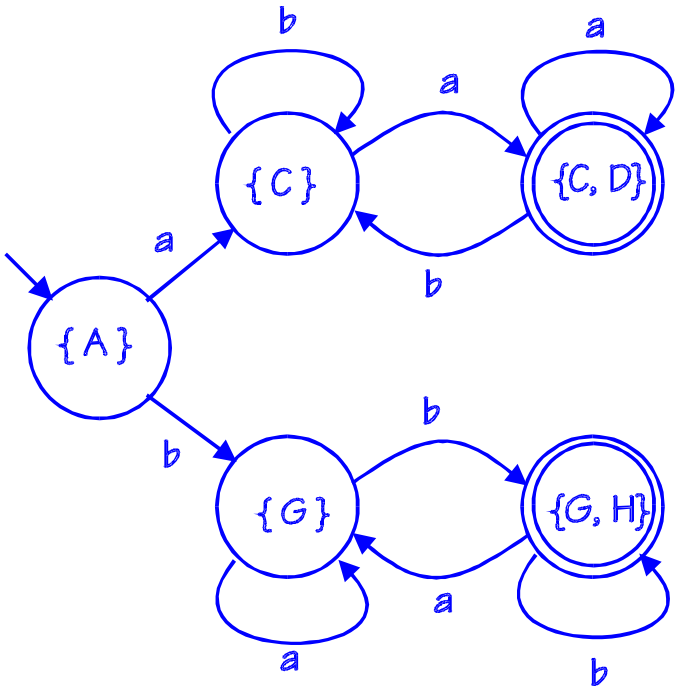


Fig. 4.7 - afd para o Exemplo 4

M	M'	M''
(A, ababa)	(A, ababa)	({A}, ababa)
(B, ababa)	---	---
(C, baba)	(C, baba)	({C}, baba)
(C, aba)	(C, aba)	({C}, aba)
(C, ba)	(C, ba)	({C, D}, ba)
(C, a)	(C, a)	({C}, a)
(D, ε)	(D, ε)	({C, D}, ε)
(I, ε)	---	---

ð

Exercício 7: Construa um afd equivalente ao afnd do Exercício 6.

ð

4.5 - Minimização de autômatos finitos

Para af determinísticos, é possível fazer uma minimização: dado um afd M, é possível construir um afd M', que é equivalente a M, e que tem o menor número de estados possível para todos os afd's que aceitam L(M). (Esta construção *não se aplica* a af não determinísticos.)

Uma propriedade adicional, que não demonstraremos aqui, é que o afd mínimo é único, exceto pelos nomes dos estados, que podem ser escolhidos arbitrariamente. Ou seja, o mesmo afd mínimo é obtido, a partir de qualquer afd que aceite a linguagem considerada.

A maneira de construir o afd mínimo é baseada na idéia de estados *equivalentes*, que podem ser *reunidos* em um só. Dois estados p e q são *equivalentes* quando as mesmas cadeias levam dos dois estados até a aceitação (até um estado final). Temos:

p ≡ q se e somente se
para toda cadeia x ∈ Σ*,
$\hat{d}$ (p, x) ∈ F se e somente se $\hat{d}$ (q, x) ∈ F

A última linha pode ser substituída por

ou $\hat{d}$ (p, x) e $\hat{d}$ (q, x) são ambos finais, ou são ambos não finais.

A relação ≡ é uma relação de equivalência. Portanto, trivialmente, para estados p, q e r quaisquer, temos

- p ≡ p (reflexividade)
- se p ≡ q, então q ≡ p (simetria)
- se p ≡ q e q ≡ r, então p ≡ r (transitividade)

(Demonstração: exercício.)

O algoritmo que vamos descrever aqui se baseia no fato de que é mais fácil provar que dois estados p e q não são equivalentes do que provar que são. Para mostrar que p e q não são equivalentes, basta achar *uma* cadeia x tal que $\hat{d}$ (p, x) é final e $\hat{d}$ (q, x) não é final, ou vice-versa. Dizemos que essa cadeia x *distingue* o estado p do estado q, e que p e q são distinguíveis.

As propriedades que vamos usar no algoritmo são:

Propriedade 1. *(Equivalência se propaga para a frente.)* Se p ≡ q, então para qualquer a ∈ Σ, os estados p' = δ(p, a) e q' = δ(q, a) são equivalentes.
Dem. Seja x ∈ Σ* uma cadeia qualquer. Devemos mostrar que p" = $\hat{d}$ (p', x) e q" = $\hat{d}$ (q', x) são ambos finais ou ambos não finais.

Seja y = ax. Temos

p" = $\hat{d}$ (p', x) = $\hat{d}$ (δ(p, a), x) = $\hat{d}$ (p, ax) = $\hat{d}$ (p, y)

e

q" = $\hat{d}$ (q', x) = $\hat{d}$ (δ(q, a), x) = $\hat{d}$ (q, ax) = $\hat{d}$ (q, y)

Como p ≡ q, p" e q" são ambos finais ou ambos não finais.

Propriedade 2. *(Distinguibilidade se propaga para trás.)* Para qualquer a ∈ Σ, se p' = δ(p, a) e q' = δ(q, a) não são equivalentes, então p e q também não são equivalentes.

Dem. Se p'e q' não são equivalentes, existe uma cadeia x que distingue p' e q'. Ou seja, chamando p" = $\hat{d}$ (p', x) e q" = $\hat{d}$ (q', x), temos que um deles é final e o outro não. Fazendo y = ax, temos

p" = $\hat{d}$ (p', x) = $\hat{d}$ (δ(p, a), x) = $\hat{d}$ (p, ax) = $\hat{d}$ (p, y)

e

q" = $\hat{d}$ (q', x) = $\hat{d}$ (δ(q, a), x) = $\hat{d}$ (q, ax) = $\hat{d}$ (q, y)

e vemos que y = ax distingue p e q.

ð

Propriedade 3. *(Iniciação)* Um estado final e um estado não final não podem ser equivalentes.

Demonstração: Sejam $p \in F$, e $q \notin F$. A cadeia vazia ε distingue p de q: $p = \hat{d}(p, \varepsilon) \in F$, e $q = \hat{d}(q, \varepsilon) \notin F$.

ð

Para minimizar um afd M, começamos por determinar quais são os pares de estados de M que são equivalentes, isto é, que podem ser reunidos em um único estado. Como é mais fácil descobrir quais são os pares de estados não equivalentes, consideramos que estados p e q são equivalentes se não conseguirmos mostrar que são distinguíveis (não equivalentes). As estruturas de dados usadas pelo algoritmo são:

para cada par (p, q) de estados distintos,

- um valor booleano marca(p, q), inicialmente com o valor false.
  Se marca(p,q) = true, p e q são distinguíveis.
- uma lista de pares de estados, inicialmente vazia.
  Se (r, s) está na lista de (p, q), isto significa que r e s serão distinguíveis, se p e q forem distinguíveis.

Se marca(p, q) = true, dizemos que o par (p, q) está marcado. Note que o par identificado como (p, q) é o mesmo par identificado como (q, p), e, portanto, tanto faz marcar (p, q), como marcar (q, p).

Note que o algoritmo que determina os pares de estados equivalentes é baseado nas propriedades vistas acima. As listas são usadas para evitar a necessidade de passar mais de uma vez por cada par de estados. Ao final da execução do algoritmo, os pares de estados equivalentes são os que *não* estão marcados. A prova de correção do algoritmo, pode ser encontrada, por exemplo, em [HopUll79][1].

Algoritmo. Determinação dos estados equivalentes em um afd M.

```
procedimento mark(p, q);
        se p ≠ q então
                marca(p, q) := true;
                para cada par (r,s) na lista de (p, q)
                        execute mark( r, s);
```

[1] **John E. Hopcroft, Jeffrey D. Ullman**, ***Introduction to Automata Theory, Languages and Computation***, Addison-Wesley, 1979 - **Sec. 3.4, p.68**

```
{parte principal do algoritmo}
        para cada p final,
                para cada q não final,
                        marca(p, q):= true;
        para cada par (p, q) não marcado,
                se existe um símbolo a tal que o par (δ(p,a), δ(q,a)) está marcado,
                        execute mark(p, q)
                senão, para cada símbolo a faça
                        p' := δ(p, a);
                        q' := δ(q, a);
                        se p' ≠ q' e (p, q) ≠ (p', q'),
                                acrescente (p, q) à lista de (p', q').
```

ð

O teste da penúltima linha não é realmente necessário, e pode ser considerado como uma otimização.

Dado M = (K, Σ, δ, i, F), usamos como estados as classes de equivalência de ≡, obtidas para M para a construção do afd mínimo M' = (K', Σ, δ', i', F'). Temos:

$K' = K/\equiv = \{ [q] \mid q \in K \}$
δ': $K' \times \Sigma \rightarrow K'$, dada por $\delta'([q], a) = [\delta(q, a)]$
$i' = [i]$
$F' = \{ [f] \mid f \in F \}$

Deixamos como exercício demonstrar a consistência da definição de δ', isto é, a demonstração de que o resultado da aplicação de δ' independe da escolha do representante q da classe de equivalência [q].

Exemplo 5: Seja M um afd com estados A, B, C, D, E e F, sendo A o estado inicial; C e F são os estados finais. Os símbolos de entrada são a e b, e δ como na tabela abaixo. M aceita as cadeias que tem um número de a's da forma 6n+2 ou 6n+5. Na realidade, bastaria exigir que o número de a's fosse da forma 3n+2, o que corresponde a um afd com apenas 3 estados, e, por essa razão, M não é mínimo, e deve ter estados equivalentes.

A tabela de transição de M é

δ	a	b
A	B	A
B	C	B
C	D	C
D	E	D
E	F	E
F	A	F

Os pares de estados (representados em ordem alfabética sem os parenteses) a serem considerados são AB, AC, AD, AE, AF, BC, BD, BE, BF, CD, CE, CF, DE, DF,

e EF. Não há necessidade de incluir pares como AA por causa da reflexividade, nem pares como BA por causa da simetria: basta incluir AB.

Vamos aplicar o algoritmo acima para determinar os pares de estados equivalentes.

(marcação dos pares final / não final)

marcamos AC, AF, BC, BF, CD, CE, DF e EF.

(exame de cada par não marcado)

AB: Temos $\delta(A, a)=B$, $\delta(B, a)=C$, e BC está marcado. Logo, marcamos AB.
AD: Temos $\delta(A, a)=B$, $\delta(D, a)=E$, e $\delta(A, b)=A$, $\delta(D, b)=D$. Como BE não está marcado, incluímos AD na lista de BE. (Note que não há necessidade de incluir AD na lista de AD.)
AE: Temos $\delta(A, a)=B$, $\delta(E, a)=F$, e BF está marcado. Logo, marcamos AE.
BD: Temos $\delta(B, a)=C$, $\delta(D, a)=E$ e CE está marcado. Logo, marcamos BD.
BE: Temos $\delta(B, a)=C$, $\delta(E, a)=F$, e $\delta(B, b)=B$, $\delta(E, b)=E$. Como CF não está marcado, incluímos BE na lista de CF.
CF: Temos $\delta(C, a)=D$, $\delta(F, a)=A$, e $\delta(C, b)=C$, $\delta(F, b)=F$. Como AD não está marcado, incluímos CF na lista de AD.
DE: Temos $\delta(D, a)=E$, $\delta(E, a)=F$ e EF está marcado. Logo, marcamos DE.

(os pares restantes são equivalentes)

Os pares marcados aparecem na tabela abaixo:

A						
B	X					
C	X	X				
D		X	X			
E	X		X	X		
F	X	X		X	X	
	A	B	C	D	E	F

Os pares restantes (não marcados) são AD, BE, CF. Logo, $A \equiv D$, $B \equiv E$ e $C \equiv F$. Naturalmente, além disso, $A \equiv A$, $D \equiv A$, etc. Note que as cadeias que distinguem os pares de estados não equivalentes podem ser deduzidas da ordem de marcação: para os pares final/não final, a cadeia é ε. Para os demais pares, neste exemplo, a cadeia é a. Por exemplo, marcamos AB porque BC estava marcado, e porque de A e B passamos com o símbolo a para B e C. A cadeia correspondente a AB é portanto $a \cdot \varepsilon = a$.

Podemos agora construir o afd mínimo: o conjunto de estados é o das classes de equivalência. Como previsto, tem apenas 3 estados. Temos:

K' = { [A], [B], [C], [D], [E], [F] } = { {A, D}, {B, E}, {C, F} }
i' = [A] = {A, D}
F' = { [C], [F] }= {C, F}

Para calcular as transições, escolhemos representantes das classes. Por exemplo, como [A] = [D] = {A, D}, $\delta'(\{A, D\}, a)$ pode ser calculada como $\delta'([A], a) = [\delta(A, a)] = [B] = \{B, E\}$ ou como $\delta'([D], a) = [\delta(D, a)] = [E] = \{B, E\}$. Qualquer que seja o

representante escolhido, o resultado será o mesmo, porque, como vimos na propriedade 1, "a equivalência se propaga para a frente".

A função de transição pode ser vista na tabela abaixo:

δ'	a	b
{A, D}	{B, E}	{A, D}
{B, E}	{C, F}	{B, E}
{C, F}	{A, D}	{C, F}

ð

Um resultado interessante, cuja demonstração pode ser encontrada na referência citada, é o de que o afd mínimo que aceita uma dada linguagem é único, exceto por isomorfismo. Neste contexto, isomorfismo quer dizer simplesmente re-nomeação de estados: dados dois afd's M_1 e M_2 que aceitam a mesma linguagem L, se construirmos os afd's mínimos associados ao dois afd's, encontraremos dois afd's M_1' e M_2' que são, na prática, idênticos: M_1' e M_2' só se distinguem pelos nomes de seus estados. Ou seja, é a linguagem L que define o afd mínimo que a aceita, e o resultado é sempre o mesmo, independente do afd aceitador de L do qual partimos.

Isso pode ser usado para resolver dois problemas interessantes. Primeiro, se quisermos determinar se dois afd's M_1 e M_2 são equivalentes, basta construir os afd's M_1' e M_2' mínimos correspondentes. Se M_1' e M_2' forem isomorfos, M_1 e M_2 são equivalentes. Segundo, se quisermos mostrar que um afd M dado é mínimo, basta aplicar a M o processo de minimização, e verificar que o resultado M' é isomorfo de M. Isto é feito no Exemplo 6, onde, adicionalmente, para cada par de estados (p, q) distintos de M, deduzimos exemplos de cadeias que os distinguem.

Exemplo 6: Vamos verificar que um afd é mínimo, aplicando a ele o processo de minimização, e mostrando que o resultado final é isomorfo do afd inicial. Seja M o afd com estados A, B, C, D, E e F, sendo A o estado inicial; e F o único estado final. Os símbolos de entrada são a e b. A tabela de transição de M é

δ	a	b
A	B	A
B	C	B
C	D	C
D	E	D
E	F	E
F	A	F

Aplicando o processo de minimização, temos:

(marcação dos pares final / não final)

marcamos AF, BF, CF, DF, EF.

(exame de cada par não marcado)

AB: incluímos AB na lista de BC;
AC: incluímos AC na lista de BD;

AD: incluímos AD na lista de BE;
AE: como BF está marcado, marcamos AE;
BC: incluímos BC na lista de CD;
BD: incluímos BD na lista de CE
BE: como CF está marcado, marcamos BE; portanto, marcamos AD (da lista de BE);
CD: incluímos CD na lista de DE;
CE: como DF está marcado, marcamos CE; portanto, marcamos BD (da lista de CE) e AC (da lista de BD);
DE: como EF está marcado, marcamos DE; portanto, marcamos CD (da lista de DE), BC (da lista de CD) e AB (da lista de BC).

Os pares marcados aparecem na tabela abaixo:

	A	B	C	D	E	F
A						
B	X					
C	X	X				
D	X	X	X			
E	X	X	X	X		
F	X	X	X	X	X	

(os pares restantes são equivalentes)

Não há pares de estados distintos restantes. Ou seja, cada estado é equivalente apenas a ele mesmo. O afd mínimo é idêntico a M, apenas tem estados {A}, {B}, {C}, {D}, {E}, {F}. As cadeias d(XY) que distinguem os pares de estados XY são:

d(AF) = d(BF) = d(CF) = d(DF) = d(EF) = ε.
d(AE) = a · d(BF) = a · ε = a
d(BE) = a · d(CF) = a · ε = a
d(AD) = a · d(BE) = a · a = aa
d(CE) = a · d(DF) = a · ε = a
d(BD) = a · d(CEF) = a · a = aa
d(AC) = a · d(BD) = a · aa = aaa
d(DE) = a · d(EF) = a · ε = a
d(CD) = a · d(DE) = a · a = aa
d(BC) = a · d(CD) = a · aa = aaa
d(AB) = a · d(BC) = a · aaa = aaaa

Naturalmente, as cadeias d(XY) podem também ser obtidas por inspeção, sem executar o algoritmo.

ð

Exercício 8: Construa um afd mínimo que aceite a linguagem L no alfabeto Σ = {a, b}, com L ={ cdxcd | c, d ∈ Σ, x ∈ Σ* }

ð

Exercício 9: Construa um afd mínimo que aceite o complemento da linguagem L do Exercício 8.

ð

4.6 - Equivalência entre autômatos finitos e gramáticas regulares

Um dos resultados que devemos estabelecer neste capítulo é que a classe de linguagens reconhecidas por automatos finitos é a classe das linguagens regulares. Já sabemos, da seção anterior, que a classe das linguagens aceitas por af determinísticos é exatamente a mesma classe das linguagens aceitas por af não determinísticos. Trata-se, portanto de estabelecer dois resultados simples, expressos através dos Teoremas 4.6 e 4.7, a seguir.

Teorema 4.6: Toda linguagem regular é aceita por um afnd.

Demonstração: Seja L uma linguagem regular. Portanto, L = L(G), para alguma gramática regular G = (N, Σ, P, S). Vamos construir um afnd M = (K, Σ, δ, i, F) que aceita a linguagem L(G). Temos: K = N U { f }, i = S, F = { f }. Ou seja, os estados de M serão os não terminais de M, mais um estado f criado para ser o único estado final. O estado inicial é o símbolo inicial de G. (Note que f deve ser um símbolo novo, para não causar confusão.)

As transições de M são dadas pelas regras de G:

Inicialmente, faça δ(A, a)= ∅, para todos os nãoterminais A e para todos os símbolos a, e para a = ε. A seguir, para todas as regras de G,

- se G tem uma regra A → a B, acrescente uma transição de A para B com o símbolo a, ou seja, acrescente B a δ(A, a).
- se G tem uma regra A → a, acrescente uma transição de A para f com o símbolo a, ou seja, acrescente f a δ(A, a).
- se G tem uma regra A → ε, acrescente uma transição de A para f com ε, ou seja, acrescente f a δ(A, ε).

Devemos mostrar que, para qualquer x ∈ Σ*, M aceita x sse x ∈ L(G). A demonstração se completa pela verificação de que a sequência de configurações (S, x) |—* (f, ε) em M corresponde exatamente à sequência de passos da derivação S ⇒* x em G.

ð

Exemplo 7: Seja a gramática regular G, dada por suas regras:

S → a A | b B
A → a A | b A | a
B → a B | b B | b

que gera a linguagem { c x c | c ∈ {a, b} e x ∈ {a, b}*}. O afnd descrito na prova do teorema anterior é M = ({ S, A, B, f }, { a, b }, δ, S, { f }), com δ dada pela tabela abaixo.

δ	ε	a	b
S	∅	{ A }	{ B }
A	∅	{ A, f }	{ A }
B	∅	{ B }	{ B, f }
f	∅	∅	∅

Seja a cadeia x=ababa. A cadeia x pertence à linguagem, como se pode ver pela derivação

$S \Rightarrow aA \Rightarrow abA \Rightarrow abaA \Rightarrow ababA \Rightarrow ababa.$

A cadeia x também é aceita por M, como se pode ver pela sequência de configurações

$(S, ababa) \vdash (A, baba) \vdash (A, aba) \vdash (A, ba) \vdash (A, a) \vdash (f, \varepsilon)$

Note que os estados e os símbolos não terminais aparecem na mesma ordem, exceto por f, que não aparece na derivação. Os símbolos terminais, entretanto, tem tratamento diverso: são gerados na derivação, e aparecem desde sua introdução até a cadeia final, e são consumidos nas transições do afnd, aparecendo desde a configuração inicial até o momento de sua leitura.

ð

Exemplo 8: Seja a gramática regular G, dada por suas regras:

$S \rightarrow a\ A \mid b\ A \mid \varepsilon$
$A \rightarrow a\ S \mid b\ S$

Temos $L(G) = \{ x \in \{a, b\}^* \mid |x| \text{ é par } \}$. O afnd descrito na prova do teorema anterior é $M = (\{ S, A, f \}, \{ a, b \}, \delta, S, \{ f \})$, com δ dada pela tabela abaixo.

δ	ε	a	b
S	{ f }	{ A }	{ A }
A	$\varnothing$	{ S }	{ S }
f	$\varnothing$	$\varnothing$	$\varnothing$

Seja a cadeia x = abba, de comprimento par. Temos:

$S \Rightarrow aA \Rightarrow abS \Rightarrow abbA \Rightarrow abbaS \Rightarrow abba.$

em G, e

$(S, abba) \vdash (A, bba) \vdash (S, ba) \vdash (A, a) \vdash (S, \varepsilon) \vdash (f, \varepsilon)$

em M.

ð

Teorema 4.7: Se L é aceita por um automato finito, então L é regular.

demonstração: Podemos supor que L é aceita por um afd $M = (K, \Sigma, \delta, i, F)$. Vamos construir uma gramática regular G para L. A gramática $G = (K, \Sigma, P, i)$ tem como símbolos não terminais os estados de M, e como símbolo inicial o estado inicial i de M. As regras de G são dadas pelas transições e pelos estados finais de M:

se $p = \delta(q, a)$ em M, G tem uma regra $q \rightarrow ap$ em P.
se q é final ($q \in F$), G tem uma regra $q \rightarrow \varepsilon$ em P

A demonstração é semelhante a anterior: devemos mostrar que, para qualquer $x \in \Sigma^*$, M aceita x sse $x \in L(G)$.

ð

Exemplo 9: Seja o afd $M = (\{q_0, q_1\}, \{a, b\}, \delta, q_0, \{q_1\})$, com δ dada pela tabela abaixo.

δ	a	b
q_0	q_1	q_1
q_1	q_0	q_0

M aceita as cadeias de {a, b}* que tem comprimento ímpar.

A gramática G correspondente, de acordo com o teroema acima, é

$q_0 \rightarrow a\ q_1 \mid b\ q_1$
$q_1 \rightarrow a\ q_0 \mid b\ q_0 \mid \varepsilon$

Para a cadeia x= ababa, temos

$(q_0, ababa) \vdash (q_1, baba) \vdash (q_0, aba) \vdash (q_1, ba) \vdash (q_0, a) \vdash (q_1, \varepsilon)$

e

$q_0 \Rightarrow aq_1 \Rightarrow abq_0 \Rightarrow abaq_1 \Rightarrow ababq_0 \Rightarrow ababaq_1 \Rightarrow ababa$

ð

4.7 - Expressões regulares

Vamos agora definir expressão regular. A expressão regular é a maneira mais compacta e mais simples de descrever conjuntos regulares, e é usada com essa finalidade em construção de compiladores, editores, sistemas operacionais, protocolos, etc. A definição abaixo é uma definição recursiva, e será usada como base para outras definições, e para as demonstrações.

Definição. Definimos uma *expressão regular* (*er*) em um alfabeto Σ através de *ER1 ¼ ER6* abaixo:

ER1. ∅ é uma er.
ER2. ε é uma er.
ER3. para cada a ∈ Σ, a é uma er.
ER4. Se α e β são er's, então (α ∨ β) é uma er.
ER5. Se α e β são er's, então (α ∘ β) é uma er.
ER6. Se α é uma er, então (α*) é uma er.

Naturalmente, α é uma er se e somente se isso pode ser provado a partir de *ER1 ¼ ER6*.

Usualmente, são omitidos os parenteses de er's, de acordo com a ordem de precedência

* → ∨ → ∘

e considerando os operadores como associativos à esquerda. Além disso, o símbolo ∘ é frequentemente omitido.

Exemplo 10: Seja Σ = {a, b} e seja a expressão regular α = (a∨b)* a b b, ou seja, com todos os parênteses, α = (((((a∨b)*)∘a)∘b)∘b). Mostramos que α é uma er, mostrando sucessivamente que são er's as expressões a seguir:

1.	a	de *ER3*
2.	b	de *ER3*
3.	(a∨b)	de 1, 2 e *ER4*
4.	(a∨b)*	de 3 e *ER6*

5. $(a \vee b)^* \circ a$ de 4, 1 e *ER5*
6. $(a \vee b)^* \circ a \circ b$ de 5, 2 e *ER5*
7. $(a \vee b)^* \circ a \circ b \circ b$ de 6, 2 e *ER5*.

ð

Definição. A linguagem $L[\alpha]$ associada a uma er (ou denotada pela er) é definida de forma recursiva, seguindo a definição de er:

ER1. $L[\varnothing] = \varnothing$;
ER2. $L[\varepsilon] = \{\varepsilon\}$;
ER3. para cada $a \in \Sigma$, $L[a] = \{a\}$;
ER4. $L[(\alpha \vee \beta)] = L[\alpha] \cup L[\beta]$;
ER5. $L[(\alpha \circ \beta)] = L[\alpha] \circ L[\beta]$;
ER6. $L[(\alpha^*)] = (L[\alpha])^*$.

Exemplo 11: Seja $\alpha = (a \vee b)^* \circ a \circ b \circ b$, como acima. Podemos determinar a linguagem $L[\alpha]$ seguindo o mesmo caminho usado para provar que α é uma er.

1. $L[a] = \{a\}$ de *ER3*
2. $L[b] = \{b\}$ de *ER3*
3. $L[a \vee b] = L[a] \cup L[b] = \{a\} \cup \{b\} = \{a, b\}$ de 1, 2 e *ER4*
4. $L[(a \vee b)^*] = (L[a \vee b])^* = \{a, b\}^*$ de 3 e *ER6*
5. $L[(a \vee b)^* \circ a] = L[(a \vee b)^*] \circ L[a] = \{a, b\}^* \circ \{a\}$ de 4, 1 e *ER5*
6. $L[(a \vee b)^* \circ a \circ b] = L[(a \vee b)^* \circ a] \circ L[b] =$
   $= \{a, b\}^* \circ \{a\} \circ \{b\} = \{a, b\}^* \circ \{ab\}$ de 5, 2 e *ER5*
7. $L[(a \vee b)^* \circ a \circ b \circ b] = L[(a \vee b)^* \circ a \circ b] \circ L[b] =$
   $= \{a, b\}^* \circ \{ab\} \circ \{b\} = \{a, b\}^* \circ \{abb\}$ de 6, 2 e *ER5*

Assim, $L[\alpha]$ é a linguagem das cadeias que terminam em abb.

ð

Uma outra forma de indicar as mesmas propriedades de pertinência vistas acima, mais adequada para provar a pertinência em casos isolados é:

ER1. Não existe x tal que $x \in L[\varnothing]$.
ER2. Se $x \in L[\varepsilon]$, então $x = \varepsilon$
ER3. Se $x \in L[a]$ (para $a \in \Sigma$), então $x = a$.
ER4. Se $x \in L[\alpha \vee \beta]$, então ou $x \in L[\alpha]$, ou $x \in L[\beta]$.
ER5. Se $x \in L[\alpha \circ \beta]$, então $x = yz$, com $y \in L[\alpha]$ e $z \in L[\beta]$.
ER6. Se $x \in L[\alpha^*]$, então ou $x = \varepsilon$, ou $x = yz$, com $y \in L[\alpha]$ e $z \in L[\alpha^*]$.

Os casos 1..5 são autoexplicativos; para o caso 6, basta observar a propriedade apresentada no Exercício 10.

Exercício 10: Mostrar que, para qualquer linguagem L , $L^* = \{\varepsilon\} \cup (L \circ L^*)$.

Exemplo 12: Suponhamos que desejamos provar que $x = abaabb \in L[\alpha]$, para a er $\alpha = (a \vee b)^* \circ a \circ b \circ b$, usando as propriedades acima. Os passos intermediários da prova estão indicados abaixo:

1. $a \in L[a]$
2. $a \in L[a \vee b]$ de 1
3. $b \in L[b]$
4. $b \in L[a \vee b]$ de 3

5.	$\varepsilon \in L[(a \vee b)^*]$	
6.	$a \in L[(a \vee b)^*]$	de 2 e 5
7.	$ba \in L[(a \vee b)^*]$	de 4 e 6
8.	$aba \in L[(a \vee b)^*]$	de 2 e 7
9.	$abaa \in L[(a \vee b)^* \circ a]$	de 8 e 1
10.	$abaab \in L[(a \vee b)^* \circ a \circ b]$	de 9 e 3
11.	$abaabb \in L[(a \vee b)^* \circ a \circ b \circ b]$	de 10 e 3

Note que cada ocorrência de um símbolo (a ou b) em x fica associada a uma ocorrência do mesmo símbolo em α. No fundo a construção da prova de que $x \in L[\alpha]$ consiste exatamente na descoberta de uma associação adequada. No exemplo acima, a única associação possível está indicada abaixo, pela numeração das ocorrências de símbolos, na er e na cadeia considerada:

$$\alpha = (a_1 \vee b_2)^* \, a_3 \circ b_4 \circ b_5 \, , \; x = a_1 \, b_2 \, a_1 \, a_3 \, b_4 \, b_5$$

Em outros casos, podem ser possíveis várias associações. Por exemplo, considere o alfabeto $\Sigma = \{a, b, c\}$, a er $\beta = (a \vee b)^* \circ (bvc)^*$ e a cadeia y = bb. Neste caso, temos

$$\beta = (a_1 \vee b_2)^* \; \circ \; (b_3 \vee c_4)^*$$

e podemos ter $y = b_2 \, b_2$, ou $y = b_2 \, b_3$, ou ainda, $y = b_3 \, b_3$.

ð

Definição. Dizemos que duas er's α e β são *equivalentes* se as linguagens a elas associadas são iguais: $L[\alpha] = L[\beta]$. A equivalência é indicada por $\alpha \equiv \beta$.

Exercício 11: Mostre que, dada uma er α, é sempre possível construir uma er β equivalente a α, de forma que ou $\beta = \varnothing$, ou β não contém o símbolo $\varnothing$.

Sugestão: considere as equivalências

$\varnothing \vee \alpha \equiv \alpha \vee \varnothing \equiv \alpha$
$\varnothing \circ \alpha \equiv \alpha \circ \varnothing \equiv \varnothing$
$\varnothing^* \equiv \varepsilon$

ð

Exercício 12: Mostre que, dada uma er α, é sempre possível construir uma er β equivalente a α, de forma que ou $\beta = \varepsilon \vee \gamma$, ou $\beta = \gamma$, e γ não contém o símbolo ε.

Sugestão: considere as equivalências

$\varepsilon \circ \alpha \equiv \alpha$
$(\varepsilon \vee \alpha)^* \equiv \alpha^*$
$(\varepsilon \vee \alpha) \circ \beta \equiv \beta \vee (\alpha \circ \beta)$
$(\varepsilon \vee \alpha) \vee \beta \equiv \varepsilon \vee (\alpha \vee \beta)$

ð

Exercício 13: Suponha a seguinte definição: uma er α é *ambígua* se para algum $x \in L[\alpha]$, existe mais de uma associação possível entre as ocorrências de símbolos em x e em α. Sejam as expressões

$\beta = (a \vee b)^* \circ (b \vee c)^*$
$\gamma = (a \vee b)^* \circ (a \circ a \vee b \circ b) \circ (a \vee b)^*$

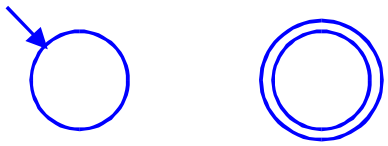
- Mostre que β e γ são ambíguas, de acordo com a definição dada.
- Construa er's equivalentes a β e γ que não sejam ambíguas.

Teorema 4.8: Toda linguagem denotada por uma expressão regular é regular.

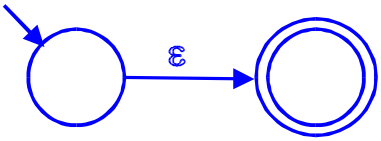
Demonstração. Seja α uma er qualquer. Vamos mostrar que $L(\alpha)$ é regular construindo um afnd $M(\alpha)$ que aceita $L(\alpha)$, preparando para uma demonstração por indução na estrutura de α.

Por simplicidade, vamos construir todos os afnd $M(\alpha)$ considerados nesta demonstração com exatamente um estado final, distinto do estado inicial. Para uma er α não elementar, o afnd $M(\alpha)$ será construído a partir dos afnd's das er's componentes. Para evitar a necessidade de nomear cada estado de cada uma dessas máquinas, vamos indicar a forma de composição graficamente. Por convenção, sempre representaremos o estado inicial à esquerda, e o estado final à direita.

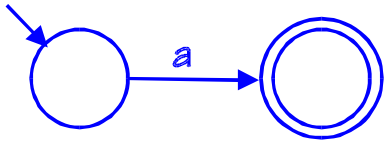
ER1. O afnd $M(\varnothing)$ que aceita $\varnothing$ é



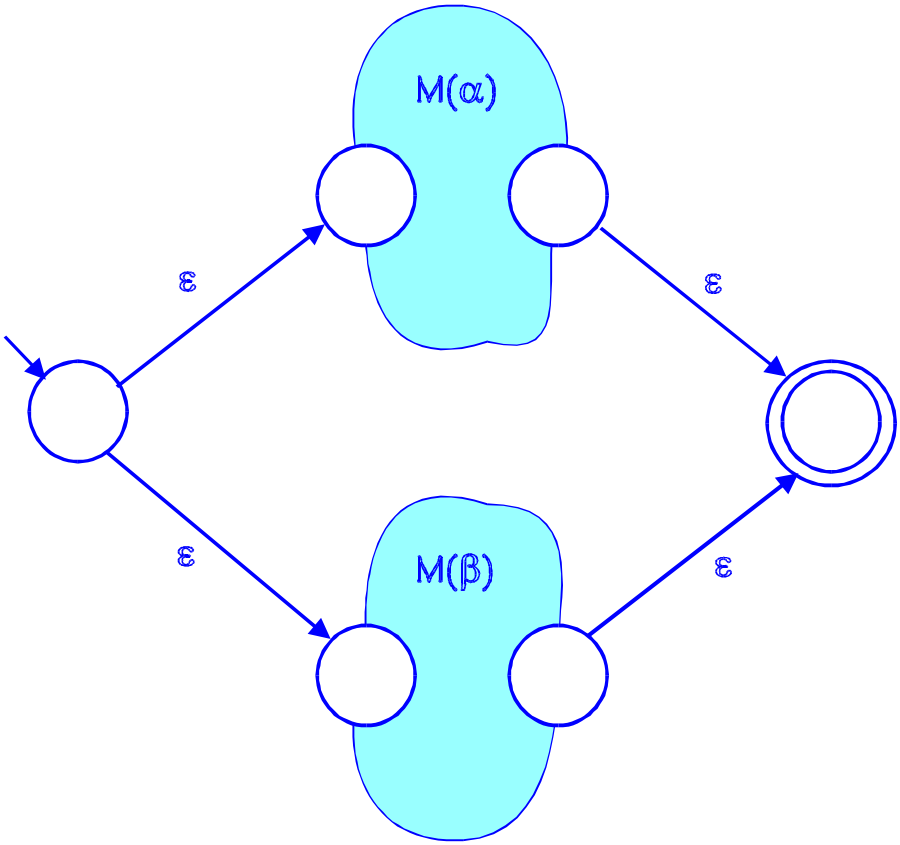
ER2. O afnd $M(\varepsilon)$ que aceita $L[\varepsilon]$ é



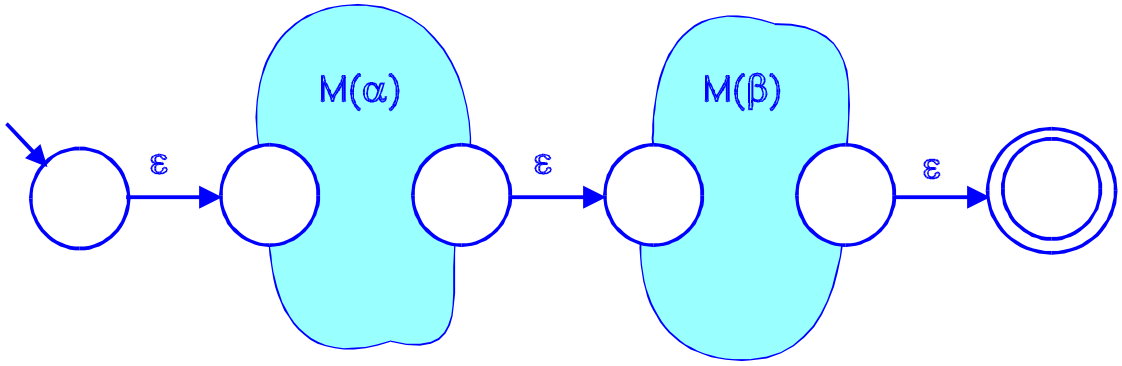
ER3. O afnd M(a) que aceita L[a], $a \in \Sigma$ é



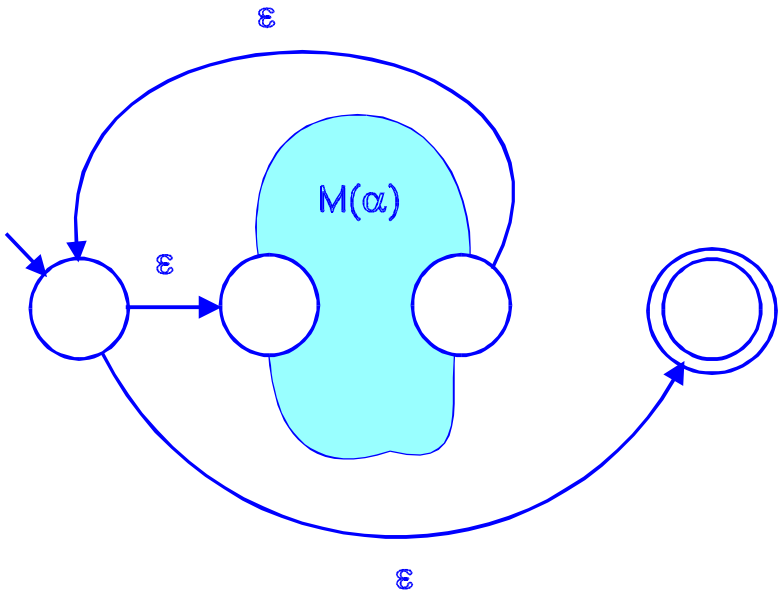
ER4. Se α e β são er's, podemos supor (pela Hipótese de Indução) que já estão construídos $M(\alpha)$ e $M(\beta)$. O afnd $M(\alpha \vee \beta)$ que aceita $L[\alpha \vee \beta]$ é obtido acrescentando um estado inicial e um final novos a $M(\alpha)$ e $M(\beta)$. As novas transições são transições com entrada ε do estado inicial novo para os antigos estados iniciais de $M(\alpha)$ e de $M(\beta)$, e dos antigos estados finais de $M(\alpha)$ e de $M(\beta)$ para o novo estado final. O resultado é



ER5. Se α e β são er's, podemos supor (pela Hipótese de Indução) que já estão construídos M(α) e M(β). O afnd M(α ∘ β) que aceita L[α ∘ β] é obtido acrescentando um estado inicial e um final novos a M(α) e M(β). As novas transições são transições com entrada ε do estado inicial novo para o antigo estado inicial de M(α), do antigo estado final de M(α) para o antigo estado inicial de M(β), e do antigo estado final de M(β) para o novo estado final. O resultado é



ER6. Se α é uma er, podemos supor (pela Hipótese de Indução) que já está construído M(α). O afnd M(α*) que aceita α* é obtido acrescentando um estado inicial e um final novos a M(α). As novas transições são transições com entrada ε do estado inicial novo para o antigo estado inicial de M(α) e para o novo estado final e do antigo estado final de M(α) para o novo estado inicial.



O restante da demonstração é deixado como exercício.

ð

Exemplo 13: Seja a er $\alpha = (a \vee b)^* a$. Vamos construir um afnd que aceita $L(\alpha)$, seguindo a construção indicada na demonstração acima. Os passos intermediários e os resultados estão indicados nas tabelas a seguir

M(a)		ε	a	b
inicial:	A	∅	{B}	∅
final:	B	∅	∅	∅

M(b)		ε	a	b
inicial:	C	∅	∅	{D}
final:	D	∅	∅	∅

M(a ∨ b)		ε	a	b
inicial:	E	{A, C}	∅	∅
	A	∅	{B}	∅
	B	{F}	∅	∅
	C	∅	∅	{D}
	D	{F}	∅	∅
final:	F	∅	∅	∅

M((a ∨ b)*)

		ε	a	b
inicial:	G	{E, H}	∅	∅
	E	{A, C}	∅	∅
	A	∅	{B}	∅
	B	{F}	∅	∅
	C	∅	∅	{D}
	D	{F}	∅	∅
	F	{G}	∅	∅
final:	H	∅	∅	∅

M((a ∨ b)*∘a)

		ε	a	b
inicial:	I	{G}	∅	∅
	G	{E, H}	∅	∅
	E	{A, C}	∅	∅
	A	∅	{B}	∅
	B	{F}	∅	∅
	C	∅	∅	{D}
	D	{F}	∅	∅
	F	{G}	∅	∅
	H	{A'}	∅	∅
	A'	∅	{B'}	∅
	B'	{J}	∅	∅
final:	J	∅	∅	∅

Note que a sub-expressão a ocorre duas vezes em M(α), e porisso foi necessário incluir duas vezes M(a); para a segunda vez renomeamos os estados, que passaram a ser A' e B'.

ð

Exercício 14: Construa um afd mínimo para a linguagem denotada pela er α do Exemplo 13, a partir do afnd M(α) construído no exemplo.

ð

Exercício 15: Construa afd's mínimos que aceitem as linguagens denotadas pelas expressões regulares do Exercício 13,

β = (a ∨ b)* ∘ (b ∨ c)*

γ = (a ∨ b)* ∘ (a∘a ∨ b∘b) ∘ (a ∨ b)*

ð

Teorema 4.9: Toda linguagem regular é denotada por uma expressão regular.
Demonstração: ver referência citada.

ð

A demonstração do Teorema 4.9 constrói a expressão regular que representa a linguagem aceita por um afd examinando os caminhos entre o estado inicial e os estados finais do afd. O operador ∨ é usado para tratar caminhos alternativos, o operador ∘ para tratar caminhos de comprimento maior que 1, e o operador * para tratar laços. Embora a construção seja interessante, na prática o uso normalmente feito de er's é para

especificação de linguagens regulares, e é muito mais frequente a construção de afd's a partir de er's, do que a construção inversa.

4.8 - O Lema do Bombeamento para linguagens regulares

Como mencionado anteriormente, precisamos de um resultado que nos permita demonstrar que algumas linguagens não são regulares. Este resultado é o "Lema do Bombeamento", ou "Pumping Lemma", que nos diz que qualquer cadeia suficientemente longa z de uma linguagem regular pode ser decomposta em três partes: z = uvw, de maneira que podemos construir outras cadeias da linguagem pela repetição da parte central v: todas as cadeias da forma $u\ v^i\ w$ são também da linguagem. Ou seja, podemos acionar a *bomba* quantas vezes quisermos, para criar quantas sentenças novas da linguagem desejarmos: uw, uvvw, uvvvw, ...

Para mostrar que uma linguagem não é regular, mostramos que não há como decompor uma cadeia (qualquer, arbitrariamente longa) da linguagem de forma que seja possível *bombear* e continuar na linguagem.

Teorema 4.10: (Lema do Bombeamento) Seja L uma linguagem regular. Então, existe um natural n tal que qualquer cadeia z de L com comprimento maior ou igual a n pode ser decomposta em três cadeias u, v e w (z = uvw) de forma que

$|uv| \leq n$
$v \neq \varepsilon$
para qualquer $i \geq 0$, $u\ v^i\ w \in L$

Demonstração: A demonstração se baseia no fato de que para as cadeias *longas* z é necessário usar pelo menos um *loop* de estados num afd que aceite a linguagem. Assim, os símbolos de u são usados para chegarmos a um estado q do loop; os símbolos de v são usados para dar a volta no loop, de volta ao estado q; os símbolos de w são usados para ir de q até um estado final. Portanto, podemos dar quantas voltas no loop quisermos, e repetir v um número qualquer i de vezes: $u\ v^i\ w$.

As cadeias *curtas* (comprimento < n) devem ser excluídas porque podem ser aceitas sem passar por nenhum loop.

A demonstração completa pode ser encontrada em [HopUll79].

ð

Exemplo 14: Seja a linguagem regular $L = L[\alpha] = L[\beta]$, com $\alpha = 1(01)^*$ e $\beta = (10)^*1$. Considere a cadeia z=10101, pertencente a L. Podemos decompor a cadeia, da forma descrita no teorema acima, de diversas formas. Por exemplo:

u = 1	v = 01	w = 01
u = 10	v = 10	w = 1
u = ε	v = 1010	w = 1

Note que todas as cadeias das formas $1(01)^i01$, $10(10)^i1$, $(1010)^i1$ pertencem a L.

ð

Exercício 16: (baseado no Exemplo 14)

- Mostre que α e β são equivalentes.
- Estime o valor de n (Teorema 4.10) para L.
- Mostre todas as formas de decomposição de z que satisfazem o Teorema.

ð

Exemplo 15: A linguagem $L = \{ a^i b^i \mid i \geq 0\}$ não é regular. (Já vimos que L é uma llc.) Vamos mostrar aqui que L não é regular. A demonstração é por contradição. Suponha que L é regular. Se L é regular, o Teorema acima se aplica, e existe n tal que a decomposição descrita pode ser realizada para qualquer cadeia de comprimento igual ou maior que n.

Seja k=n+1. Considere a cadeia $z=a^k b^k$. Qualquer decomposição z=uvw deve ter em v o mesmo número de a's e de b's, para que a propriedade de que o número de a's é igual ao de b's se mantenha nas cadeias u v^i w. Se isso não acontecer, quando acrescentarmos mais um v (aumentando i de 1), obteremos uma cadeia fora da linguagem. Portanto, v deve ser da forma $a^j b^j$, com j>0, já que v não pode ser vazia. Mas nesse caso, u v^2 w conterá a cadeia $a^j b^j a^j b^j$, com pelo menos um a depois de um b, o que não pode acontecer na linguagem.

Ou seja, nenhuma decomposição é possível, contrariando o Teorema, e podemos concluir que L não é regular.

ð

Exercício 17: Mostre que a linguagem $L = \{ x\, x^R \mid x \in \{0, 1\}^* \}$ não é regular.

ð

4.9 - Propriedades de fechamento das linguagens regulares

Vamos ver agora algumas propriedades de fechamento da classe das linguagens regulares. A maioria das provas pode ser feita usando mais de um dos formalismos usados para definir linguagens regulares: gramáticas regulares, afd's, afnd's e expressões regulares — veja o Exercício 18.

Teorema 4.11: A classe das linguagens regulares no alfabeto Σ é fechada para as operações de *(a)* união, *(b)* interseção, *(c)* complemento em relação a Σ^*, *(d)* concatenação e *(e)* fechamento transitivo (estrela).

Demonstração:
(a) Vamos mostrar que se L_1 e L_2 são linguagens regulares, então $L_1 \cup L_2$ é uma linguagem regular. Sejam α e β er's tais que $L(\alpha)=L_1$ e $L(\beta) = L_2$.
Portanto, $L_1 \cup L_2 = L(\alpha \vee \beta)$ também é regular.

(b) Vamos mostrar que se L_1 e L_2 são linguagens regulares, então $L_1 \cap L_2$ é uma linguagem regular. Sejam $M_1 = (K_1, \Sigma, \delta_1, i_1, F_1)$ e $M_2 = (K_2, \Sigma, \delta_2, i_2, F_2)$, afd's tais que $L(M_1)=L_1$ e $L(M_2) = L_2$.
Seja $M = (K_1 \times K_2, \Sigma, \delta, (i_1, i_2), F_1 \times F_2)$, com δ definida por

$$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$$

M aceita $L_1 \cap L_2$, porque

$((i_1, i_2), x) \vdash\!\!\!-^* ((f_1, f_2), \varepsilon)$ em M sse
$(i_1, x) \vdash\!\!\!-^* (f_1, \varepsilon)$ em M_1
e $(i_2, x) \vdash\!\!\!-^* (f_2, \varepsilon)$ em M_2,

e

(f_1, f_2) é final em M sse
f_1 é final em M_1
e f_2 é final em M_2.

L é aceita por M, e portanto, é regular.

(c) Vamos mostrar que se L é regular, o complemento $\overline{L} = \Sigma^* - L$ também é regular. Seja $M = (K, \Sigma, \delta, i, F)$ um afd que aceita L. Então $M' = (K, \Sigma, \delta, i, K-F)$ aceita $\overline{L}$. Temos:

M' aceita $x \Leftrightarrow \hat{\delta}(i, x) \in K - F \Leftrightarrow \hat{\delta}(i, x) \notin F \Leftrightarrow$ M não aceita x
$\Leftrightarrow x \notin L \Leftrightarrow x \in \overline{L}$.

(d) Vamos mostrar que, se L_1 e L_2 são regulares, $L_1 \circ L_2$ é regular. Sejam α e β er's tais que $L(\alpha)=L_1$ e $L(\beta) = L_2$. Portanto, $L_1 \circ L_2 = L(\alpha \circ \beta)$ também é regular.

(e) Vamos mostrar que se L é regular, o fechamento L* também é regular. Seja a uma er tal que $L(\alpha) = L$. Então $L(\alpha^*)^* = (L(\alpha))^* = L^*$ também é regular.

ð

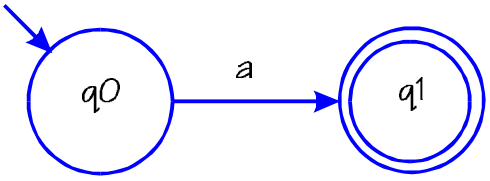
Exercício 18: Considere o Teorema 4.11. Construa demonstrações alternativas para os casos indicados:

(a) construindo uma gramática regular para a união de L_1 e L_2, a partir de gramáticas regulares de L_1 e L_2.
(b) usando (a), (c), e a propriedade de conjuntos (de Morgan) que diz que $X \cap Y = \overline{\overline{X} \cup \overline{Y}}$
(d) construindo uma gramática regular para a concatenação de L_1 e L_2, a partir de gramáticas regulares de L_1 e L_2.
(e) construindo uma gramática regular para o fechamento de L, a partir de uma gramática regular de L.

ð

Exercício 19: Mostre que, na construção usada na demonstração do Teor. 4.11 parte (c), não pode ser usado um afnd.

Sugestão: Considere o afnd



(revisão de 27fev97)