CAPÍTULO 4

PROPRIEDADES DAS LINGUAGENS REGULARES

4.1. Introdução

Nos capítulos anteriores estudámos fundamentalmente o contexto regular para linguagens, gramáticas e autómatos. O esquema de equivalências a que se chegou define um universo de linguagens interessante e útil. Importa agora conhecer mais pormenorizadamente as propriedades dessa família de linguagens. Qual o seu grau de generalidade ? São fechadas em relação a certas operações sobre conjuntos? Uma linguagem é finita ou não? Outra importante questão é como se pode saber se uma dada linguagem é regular. Se conhecermos uma propriedade geral que todas as linguagens regulares têm obrigatoriamente, e se uma dada linguagem não a verifica, então ela não é regular. Veremos uma tal propriedade geral.

4.2. Propriedades de fecho das linguagens regulares

As propriedades de fecho das linguagens regulares sob diferentes operações têm um interesse teórico considerável. Dão uma perspectiva da natureza genérica das famílias de linguagens e ajudam na resposta a questões práticas. Um conjunto é fechado em relação a uma operação (sobre os seus elementos) se o resultado da aplicação dessa operação a quaisquer dos seus elementos resulta num seu elemento. A figura 4.44. seguinte ilustra esta propriedade para uma operação unária e uma operação binária (com dois argumentos): o resultado dá sempre um elemento do mesmo conjunto.

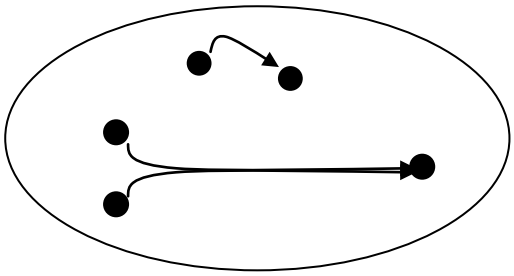


Figura 4.1.1. Fecho de um conjunto em relação a uma operação entre os seus elementos.

4.2.1. Fecho em relação a operações de conjuntos

Algumas conhecidas operações sobre conjuntos quando aplicadas a linguagens regulares produzem linguagens regulares.

Teorema 4.1. **Se L_1 e L_2 são linguagens regulares, então também o são**

(i) $L_1 \cup L_2$, propriedade de fecho em relação à união

(ii) $L_1 \cap L_2$, propriedade de fecho em relação à intersecção

(iii) $L_1 L_2$, propriedade de fecho em relação à concatenação,

(iv) $Compl(L_1)$, $Compl(L_2)$, fecho em relação à complementação

(v) L_1^*, L_2^*, fecho em relação ao fecho-estrela

(vi) $L_1 - L_2 = L_1 \cap Compl(L_2)$, fecho em relação à diferença

Demonstração:

A demonstração destas propriedades faz-se recorrendo ao que estudámos nos capítulos anteriores.

(i) (iii) (v):

Vimos que se L_1 e L_2 são regulares, existem expressões regulares r_1 e r_2 que lhes correspondem, isto é $L_1 = L_1(r_1)$, $L_2 = L_2(r_2)$.

Vimos também que por definição de expressões regulares, $r_1 + r_2$, $r_1 r_2$, r_1^* , r_2^* são expressões regulares denotando respectivamente as linguagens $L_1 \cup L_2$, $L_1 L_2$, L_1^*, L_2^*

Em consequência o fecho em relação à união(i), à concatenação (iii), ao fecho-estrela (v) é evidente e está demonstrado.

(iv)

Em relação à complementação (iv), não é assim tão imediato, mas já o vimos no Cap. 2.

Seja $M = (Q, \Sigma, \delta, q_0, F)$ o DFA que aceita L_1. Ele tem estados finais F. Se considerarmos agora o autómato que se obtém deste invertendo a qualidade aceitador/não aceitador de cada estado, teremos o DFA complementar M^c.

Então o DFA $M^c = (Q, \Sigma, \delta, q_0, Q - F)$ aceita $Compl(L_1)$.

De facto, num DFA a função de transição generalizada δ^* é uma função total, isto é, ela é definida para todas as cadeias de Σ^*. Portanto, dada uma cadeia w e o estado inicial q_0, $\delta^*(q_0, w)$ existe, e é ou um estado final (pertence e F, caso em que $w \in L_1$) ou um estado não aceitador e portanto pertencente a Q-F e neste caso $w \in Compl(L_1)$. Havendo um DFA que aceita $Compl(L_1)$, ela é regular e verifica-se o fecho em relação à complementação.

(ii)

Em relação à intersecção, faz-se uma prova construtiva: constrói-se um DFA especial a partir dos DFA de L_1 e L_2, respectivamente M_1 e M_2.

Sejam $M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$ e $M_2 = (P, \Sigma, \delta_2, p_0, F_2)$ os DFA de L_1 e L_2.

Vamos construir o DFA intersecção destes dois e chamemos-lhe M_{12}: ele aceitará uma cadeia se e só se essa cadeia for aceite por M_1 e por M_2 . Tomamos um carácter do alfabeto (que é o mesmo nas duas linguagens) e seguimo-lo num e noutro autómato. Se ele está num estado q_i de M_1 e num estado p_j de M_2, criamos um estado q_{ij} em M_{12}. Se ele transita em M_1 para q_k e em M_2 para p_l, cria-se o estado q_{kl} em M_{12}. Assim os estados de M_{12} correspondem a pares de estados de (Q, P).

Suponhamos o alfabeto Σ={a,b} e o carácter a. A Fig. 4.2.1 ilustra a demonstração.

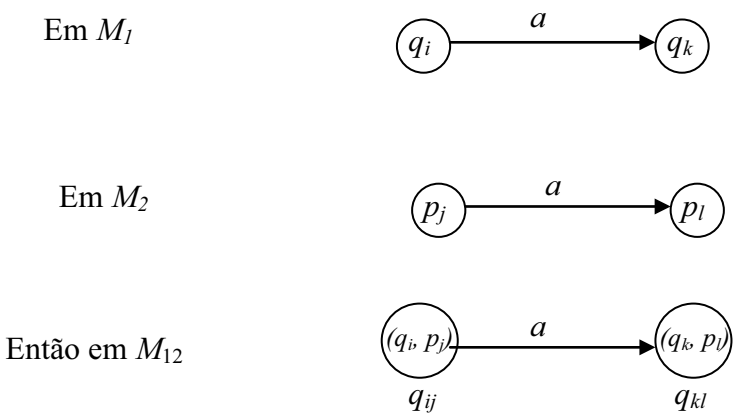


Figura 4.2.1 Autómato da intersecção de duas linguagens

De seguida vê-se para onde se transita com b, em q_i de M_1 e p_j de M_2, e cria-se em M_{12} o estado respectivo.

O conjunto dos estados de M_{12} é um subconjunto do produto cartesiano dos estados de M_1 com os de M_2, QxP, composto por pares (q_i, p_j), com $q_i \in Q_j$ e $p_j \in P$.

A função de transição do DFA M_{12}, $\underline{\delta}$, obtém-se sabendo que M_{12} está no estado (q_i, p_j) sempre que M_1 está no estado q_i e M_2 está no estão q_j, ou seja,

$$\underline{\delta}((q_i, p_j), a) = (q_k, p_l)$$

Sempre que

$$\delta_1(q_i, a) = q_k$$

e

$$\delta_2(p_j, a) = p_l$$

E quais serão os estados aceitadores em M_{12}?

Queremos obter o DFA que aceita a intersecção das linguagens L_1 e L_2, isto é, as cadeias que levam M_1 a um estado aceitador e ao mesmo tempo M_2 também a um estado aceitador. Para que isso aconteça, fazemos aceitadores em M_{12} os estados compostos por estados aceitadores, isto é, os estados (q_i, p_j) tais que q_i é aceitador em M_1 ($q_i \in F_1$) e p_j é aceitador em M_2 ($p_j \in F_2$).

Deste modo o DFA M_{12} aceita todas as cadeias, e só essas, $w \in L_1 \cap L_2$.

Alternativamente a esta demonstração construtiva, a usando os nossos conhecimentos de lógica, poderemos usar a Lei de DeMorgan

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

Se L_1 é regular, também o é o seu complemento. Se L_2 é regular, também o é o seu complemento. Se os dois complementos são regulares, também o é a sua união. Se a união é regular, também o é o seu complemento. Logo a intersecção é regular.

vi) Quanto à diferença, escrevamo-la de outra forma

$$L_1 - L_2 = L_1 \cap \overline{L_2}$$

Sendo L_2 regular, também o é o seu complemento. Sendo L_1 regular, a intersecção com outra regular dá uma regular, como acabámos de provar. Logo a diferença é regular (sendo L_1 e L_2 regulares, naturalmente).

Há uma outra operação interessante sobre uma linguagem - a reversão-, que se opera revertendo toda e qualquer cadeia da linguagem. Ora a reversa de uma linguagem regular é uma linguagem regular.

Teorema 4.2. A família das linguagens regulares é fechada em relação à reversão.

Para demonstrar, construirmos o DFA da linguagem e depois procuramos, a partir dele, um DFA que aceita a linguagem reversa. Se conseguirmos, está a prova feita.

Uma cadeia reverte-se lendo-a do fim para o princípio. Tendo um NFA que aceita uma cadeia, ele vai-a lendo, partindo do estado inicial, até chegar ao estado aceitador. Se agora andarmos para trás, partindo do estado final aceitador, percorrendo as arestas em sentido contrário, chegaremos ao estado inicial depois de lermos a cadeia ao contrário. Portanto para a reversão, faz-se uma prova construtiva através dos passos seguintes:

- constrói-se um NFA com um só estado final para a linguagem *L*.

- transforma-se o estado inicial em final e o final em inicial

- inverte-se o sentido das setas em todas as arestas do grafo do NFA.

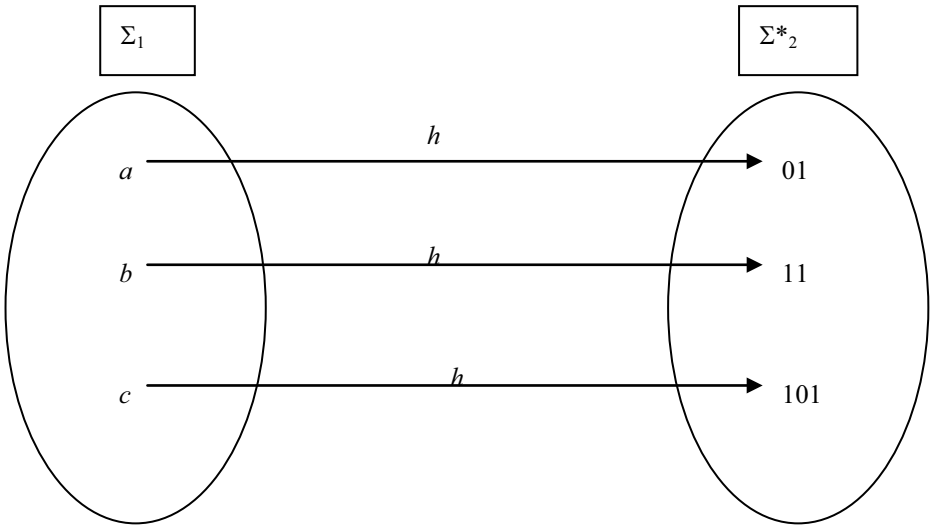
- o NFA resultante aceita L^R

Logo o NFA da reversa existe, porque acabámos de o construir, e por isso L^R é regular.

4.2.2. Fecho em relação a outras operações

4.2.2.1. Homomorfismo

Um homomorfismo é uma correspondência biunívoca entre dois conjuntos. Dois elementos são homomorfos (ou homomórficos) se são semelhantes na forma ou compostos por partes semelhantes entre si (*homo*-a mesma, *morfo*-forma).



$h\,(abc) = h(a)h(b)h(c) = (01)(11)(101)=0111101$

Figura 4.2.2. Ilustração do homomorfismo

Uma cadeia no conjunto origem é homomórfica de uma cadeia no conjunto de chegada. A relação de semelhança resulta da transformação *h*.

Teorema 4.3. Seja *h* um homomorfismo. Se *L* é uma linguagem regular, então a sua imagem homomórfica é também regular. A família das linguagens regulares é por isso fechada em relação a qualquer homomorfismo.

Podemos imaginar uma prova construtiva deste teorema. Seguindo o exemplo, construímos o NFA da linguagem original. Nele existem arestas com etiquetas *a*, *b* e *c*.

Substituamos cada aresta *a* por uma aresta 0 seguida de um estado seguido de uma aresta 1. Substituamos cada aresta *b* por uma aresta 1 seguida de um estado seguido de uma aresta 1. E substituamos cada aresta *c* por uma aresta 1 seguida de um estado seguido de uma aresta 0 seguida de um estado 1.

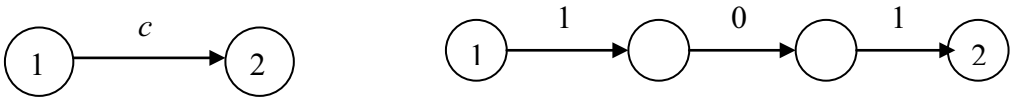


Figura 4.2.3. O autómato de um homomorfismo

O NFA obtido aceita a linguagem que resulta da original depois do homomorfismo *h*, que por isso mesmo é uma linguagem regular.

Pode-se também demonstrar a partir das expressões regulares. Escreve-se uma expressão regular da linguagem original *L*. Nela substituem-se os caracteres *a, b, c,* pelas suas imagens homomórficas 01, 11, 101. O que resulta é ainda uma expressão regular e portanto denota uma linguagem regular. Esta é a imagem homomórfica de *h*.

4.2.2.2 Quociente à direita

Considerem-se duas linguagens L_1 e L_2 no mesmo alfabeto. Tomemos agora todas as cadeias de L_1 que se podem decompor em duas partes: um sufixo *y* e um prefixo *x*, tal que o sufixo *y* constitui uma cadeia de L_2.

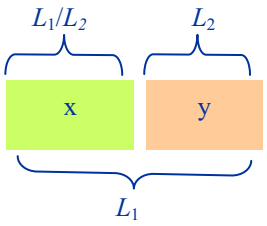


Figura 4.2.4. Quociente à direita

O conjunto dos prefixos *x* compõe a linguagem quociente de L_1 por L_2 à direita. Isto é,

$$L_1/L_2=\{x: xy \in L_1 \text{ para algum } y \in L_2\}$$

Teorema 4.4. Se L_1 e L_2 são linguagens regulares, então L_1/L_2 é também regular. A família das linguagens regulares é fechada em relação ao quociente à direita por uma linguagem regular.

Para demonstrar o teorema constrói-se o DFA do quociente. Ver por exemplo em Linz 107.

4.3. Propriedades de decisão de linguagens regulares

Há questões importantes sobre linguagens regulares às quais temos que saber responder. Não se trata naturalmente de analisar todas as cadeias de uma linguagem, tipicamente infinita, e ver se elas têm alguma característica especial. Essa resposta obtém-se analisando uma das formas de representação estudadas: um autómato finito (determinístico ou não determinístico), uma expressão regular ou uma gramática regular. Uma questão diz-se *decidível* se formos capazes de lhe dar uma resposta genérica, aplicável a todas as linguagens

(neste caso regulares), isto é, se existir um algoritmo para ela. Veremos de seguida algumas dessas questões fundamentais.

4.3.1. A questão da pertença

Dada uma linguagem qualquer, uma dada cadeia *w* pertence-lhe?

Teorema 4.5. Dada uma representação padrão de qualquer linguagem regular *L* em Σ e dada uma qualquer cadeia $w \in \Sigma^*$, existe um algoritmo para determinar se *w* pertence ou não a *L*.

Demonstração (construtiva):

Representa-se a linguagem por um DFA, e depois testa-se se o DFA aceita ou não *w*. Este é o algoritmo.

4.3.2. A questão de finitude ou infinitude de uma linguagem.

Como saber se uma linguagem é finita ou infinita?

Teorema 4.6. Existe um algoritmo para verificar se uma linguagem, dada numa forma padrão, é **vazia**, **finita** ou **infinita**.

A demonstração é construtiva:

1º Representa-se a linguagem por um grafo de transição de um DFA, o que é sempre possível por ser regular.

2º Analisam-se os caminhos do grafo.

Se existe um caminho do estado inicial ao estado final**,** a linguagem não é **vazia**.

3º Procuram-se todos os vértices que são base de um ciclo.

Se algum desses vértices base está num caminho do estado inicial ao estado final, a linguagem é **infinita.** Se não, é **finita**.

De facto poderemos dar um número infinito de voltas ao ciclo, e por cada volta aceita-se uma cadeia; logo o número de cadeias aceites é infinito.

4.3.3. A questão da igualdade de linguagens

Como verificar se duas linguagens são ou não iguais?

Teorema 4.7. Dadas duas linguagens regulares L_1 e L_2 numa forma padrão, existe um algoritmo para determinar se $L_1=L_2$.

Demonstração construtiva:

Constrói-se o DFA para a união das diferenças das linguagens

$$L = (L_1 - L_2) \cup (L_2 - L_1)$$

e testa-se aí se a linguagem resultante é vazia. Se sim, as linguagens são iguais, se não, são diferentes. De facto se $L_1=L_2$ então $L_1 - L_2=\varnothing$ e $L_2 - L_1=\varnothing$ e $\varnothing \cup \varnothing=\varnothing$. Por outro lado se L não é vazia, então ou $L_1 - L_2$ ou $L_2 - L_1$ é não-vazia e por isso $L_1 \neq L_2$.

4.4. Identificação de linguagens não regulares: lema da bombagem

A questão básica sobre uma linguagem- será ela regular ? – é a de resposta mais difícil. Sabemos que se formos capazes de construir um NFA para ela, ela é regular. E se não formos ? Não será por falta de engenho e arte? Haverá um algoritmo para saber se uma linguagem **não é** regular? Disso trata o tão estranho (numa primeira abordagem) lema da bombagem.

4.4.1. O princípio do pombal (*pigeonhole*)

O princípio do pombal é usado pelos matemáticos para se referirem à seguinte situação: se dispusermos n objectos em m caixas (gaiolas no pombal) e se $n>m$, então pelo menos uma caixa tem que conter mais do que um objecto.

A analogia com os autómatos finitos reside no facto de estes, por terem um número finito de estados, terem memória limitada. Assim sendo eles não são capazes de distinguir prefixos (de cadeias) de comprimentos arbitrários.

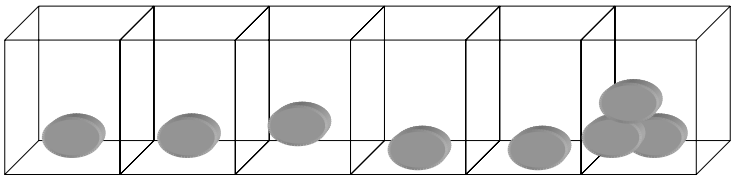


Figura 4.4.1. O princípio do pombal.

Por exemplo, num grafo de transição com *n* vértices, qualquer caminho de comprimento igual ou superior a *n* tem que repetir algum vértice, isto é, tem que conter um ciclo. Se cada estado for considerado uma gaiola, ele terá mais de uma passagem.

Exemplo 4.4.1. Seja *L=L*(*aba*b*)

O seu NFA é fácil de desenhar (Figura 4.4.2) :

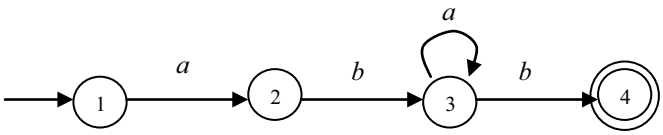


Figura 4.4.2. Exemplo 4.4.1

A cadeia *abb* leva o DFA do estado inicial ao final aceitador, passando uma vez em cada estado. O caminho *abaab* passa duas vezes no estado 3. Qualquer caminho aceitador maior passará mais vezes em 3. Temos aqui uma ilustração do princípio do pombal.

Considere-se a cadeia *abab* de *L*. Poderemos decompô-la em três partes - *x, y e z*- da seguinte forma:

Tabela 4.4.1

ab	*a*	*b*
x	*y*	*z*

Se tivermos agora a cadeia *abaab* também a poderemos decompor em três partes mas de modo ligeiramente diferente,

Tabela 4.4.2

ab	*aa*	*b*
x	y^2	*z*

e *abaaab*

Tabela 4.4.3

ab	*aaa*	*b*
x	y^3	*z*

E assim sucessivamente, qualquer cadeia aceite se pode decompor em três partes xy^iz como na Tabela 4.4.4 seguinte:

Tabela 4.4.4

i	$w=xy^iz$
0	*abb*
1	*abab*
2	*abaab*
3	*abaaab*
4	*abaaaab*
…	...

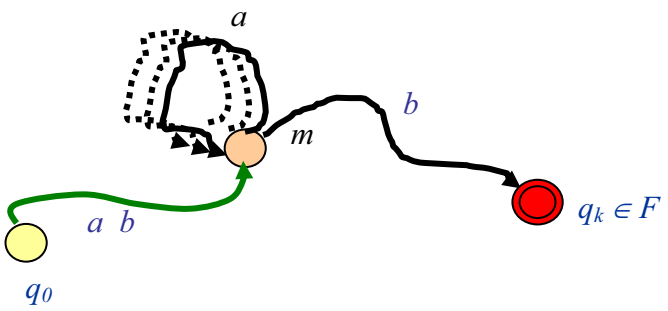


Figura 4.4.3. A decomposição para o exemplo 1. Nesta decomposição $x=ab$, $z=b$.

Podem-se desenhar os caminhos correspondentes, como na figura 4.4.3: partindo de q_0, com o segmento *ab* chega-se ao ponto (estado) *m*; aí podem-se tirar *a*'s da cartola, tantos quantos se quiser, ou, dito de outro modo, podem-se bombear *a*'s de um poço sem fundo, até que decidimos prosseguir, percorrendo o segmento *b* até ao fim. Todas as cadeias bombeadas (isto é, no caso com *a*'s bombeados) pertencem à linguagem, dado que levam ao estado aceitador

Analisemos em detalhe a decomposição *xyz* das cadeias desta linguagem. Ela é possível para qualquer cadeia da linguagem de comprimento igual ou maior do que 4 (*abab, abaab*, …). A decomposição verifica as condições seguintes:

$$w=xyz$$

com

$$|xy|=aba = 3$$

e

$$|y| = 1$$

Se bombearmos *y* obtemos

$$w_i = xy^i z$$

a cadeia w_i pertence a *L* para todo o *i=0, 1, 2, ...*

Se *m* for o comprimento da menor cadeia que contém uma vez o ciclo, neste caso *m*=4 e $|xy| = 3 \leq m$.

Pode-se fazer uma construção semelhante para uma qualquer linguagem regular **infinita.** Como é representada por um autómato finito, para ser infinita tem que ter cadeias arbitrariamente grandes, e portanto tem que ter pelo menos um ciclo que o permita.

Exemplo 4.4.2.

Seja a linguagem *L=abb(baa)*bb* regular e infinita.

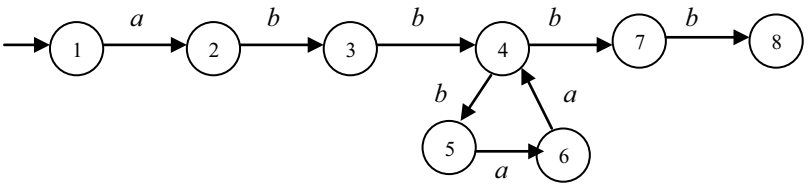


Figura 4.4.4. Autómato de *L=abb(baa)*bb*

Se olharmos atentamente para o NFA verificamos que o único ciclo existente é 4-5-6. Será aí que se faz a bombagem. Bombeia-se *bab* um número arbitrário de vezes. A decomposição *xyz* neste caso tem *y=bab*. As outras partes serão *x=abb*, *z=bb.* Por outro lado a menor cadeia que contém *xy* é *abbbaabb* e portanto *m*= 8.

Existe assim uma decomposição que verifica as condições seguintes:

$$w = xyz$$

com

$$|xy|=abbbaa = 6 \leq m$$

e

$$|y| = 3$$

Se bombearmos *y* obtemos

$$w_i=xy^iz$$

a cadeia w_i pertence a *L* para todo o *i=0, 1, 2, ...*

Exemplo 4.4.3

No caso da linguagem regular *L=a(abb)*bb(baa)*bb*, cujo grafo do NFA é o da Fig 4.4.5,

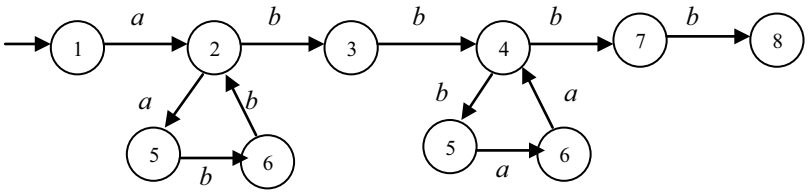


Figura 4.4.5. Autómato de *L=a(abb)*bb(baa)*bb*

existem dois ciclos que fazem a linguagem infinita. Se procurarmos uma decomposição *xyz* das cadeias da linguagem onde possamos fazer a bombagem, encontraremos duas possibilidades, conforme o ciclo que escolhermos:

- escolhendo o 1º ciclo, será *xyz= a(abb)bbbb* ou *a(abb)bbbaabb* ou $a(abb)bb(baa)^2bb$

Tabela 4.4.6

x	*y*	*z*
a	*abb*	*bbb*
a	*abb*	*bbbaabb*
a	*abb*	*bbbaabaabb*
a	*abb*	*...*

temos que *x* e *y* são bem definidos mas *z* pode ter muitas composições.

- escolhendo o 2º ciclo, será *xyz= abb(baa)bb* ou *a(abb)bb(baa)bb* ou ...

Tabela 4.4.7

x	*y*	*z*
abb	*baa*	*bb*
aabbbb	*baa*	*bb*
aabbabbbb	*baa*	*bb*
aabbabbabbbb	*baa*	...

temos agora que *y* e *z* são bem definidos mas *x* pode ter muitas composições.

Para todas as cadeias da linguagem de comprimento maior do que 8 existe pelo menos um ciclo. Para que uma cadeia contenha os dois ciclos pelo menos uma vez, terá um comprimento de pelo menos 11. Podemos assim afirmar que para todas as cadeias maiores do que 11 existe **pelo menos uma** decomposição *xyz* que verifica as propriedades

$$w=xyz$$

com

$$|xy| \leq 11$$

e

$$|y| \geq 1$$

Se bombearmos *y* obtemos

$$w_i=xy^iz$$

a cadeia w_i pertence a *L* para todo o *i=0, 1, 2, ...*

Estamos agora em condições de enunciar esse tão famoso quanto incompreendido lema da bombagem para linguagens regulares.

4.4.2. O Lema da bombagem (*pumping lema*)

A propriedade das linguagens regulares que acabámos de analisar é formalizada pelo lema da bombagem.

Teorema 4.8. Seja L uma linguagem regular infinita. Então existe algum inteiro positivo m tal que qualquer cadeia $w \in L$ com $|w| \geq m$ se pode decompor em

$$w=xyz$$

com

$$|xy| \leq m$$

e

$$|y| \geq 1$$

tal que

$$w_i=xy^iz$$

também pertence a L para todo o $i=0, 1, 2, ...$

Quer dizer que qualquer cadeia suficientemente longa de L se pode partir em três partes de tal modo que um número arbitrário de repetições da parte do meio produz outra cadeia de L. Pode-se dizer por isso que a sub-cadeia do meio é "bombeada", e daí o nome do lema.

Qual o interesse do lema ? Provar que uma linguagem é regular ? Será que só as linguagens regulares o verificam ? O lema não diz que se aplica só a elas.

Exemplo 4.4.4.

Consideremos a linguagem $L = \{aa^pab^qa^q, p \geq 0, q \geq 1\}$

Existe um $m = 5$ tal que para toda a cadeia $|w| \geq 5$ é possível encontrar uma decomposição $w=xyz$ tal que $|xy| \leq 4$, $|y| \geq 1$, e a cadeia $w_i=xy^iz$ pertence à linguagem para todo o $i \geq 0$.

Demonstração:

Seja a cadeia de L é *aaaba*. Nela podemos fazer a decomposição

$xyz = |aa|a|ba|$ obedecendo às restrições de tamanho. Bombeando agora y obtém-se

Tabela 4.4.8

i	$w=xy^iz$
0	*aaba*
1	*aaba*
2	*aaaba*
3	*aaaaba*
4	*aaaaaba*
…	...

cadeias que pertencem todas a L.

Para cadeias de L maiores do que 5, por exemplo *aaaaba*, *aabababa*, pode-se sempre fazer a decomposição com $x=a$, $y=a$, de tal modo que bombeando o y se obtêm cadeias da linguagem para todo o valor de i.

No entanto esta linguagem é livre de contexto não regular, como veremos posteriormente. O leitor pode tentar desenhar um autómato finito, ou escrever uma expressão regular para ela que não conseguirá fazê-lo devido ao termo a^qb^q que tem forçosamente um número de a's igual ao número de b's . Por isso as linguagens regulares são um subconjunto das linguagens que verificam o Lema da Bombagem.

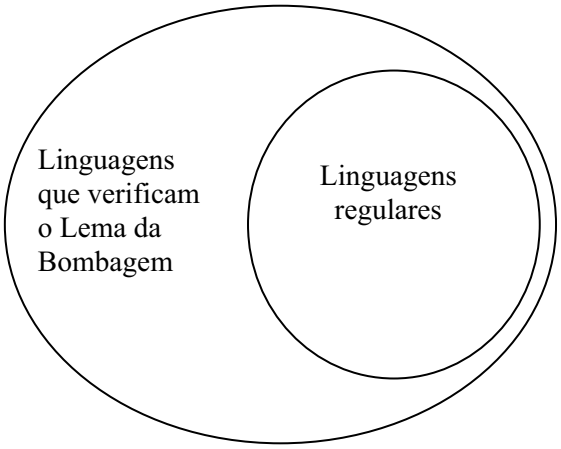


Figura 4.4.6. As linguagens regulares verificam o Lema da Bombagem. Mas há outras, não regulares, que também o verificam.

O Lema diz-nos que se a linguagem é regular (proposição A) então existe um *m* apropriado (proposição B),

$$A \rightarrow B \text{ (implicação)}$$

mas isto não nos autoriza a dizer que se existe um tal *m* então a linguagem é regular, isto é não é verdade que

$$B \rightarrow A$$

O que sabemos com toda a certeza é que

$$\textbf{Se } (A \rightarrow B) \textbf{ então } (\text{não}B \rightarrow \text{não}A)$$

isto é, se não existe um tal *m*, a linguagem não é regular, com toda a certeza.

O Lema usa-se precisamente com **não**$B \rightarrow$**não**A: procura-se demonstrar com ele que uma linguagem **não é** regular.

Para isso temos que provar que é impossível a existência de um *m* apropriado. Como fazer a prova ? Por contradição: supõe-se que existe e tenta-se reduzir ao absurdo.

Vejamos a lógica do Lema e do seu contrário:

Tabela 4.4.9.

O Lema pela afirmativa	**O Lema pela negativa**
- existe **um** *m* tal que para	- para **qualquer** valor de *m* sou capaz de encontrar
- **qualquer** cadeia $\|w\| \geq m$ pertencente a *L* existe	- (pelo menos) **uma** cadeia $\|w\| \geq m$ pertencente a L em que
- **uma** decomposição *xyz* que produz, pela bombagem de *y*, $\|xy\| \leq m$	- **qualquer** decomposição *xyz* produz, pela bombagem de *y,* $\|xy\| \leq m$
- **todas** cadeias pertencentes a *L* $\forall i \geq 0, xy^i z \in L$	- (pelo menos) **uma** cadeia que não pertence à linguagem $\exists\, i \geq 0, xy^i z \notin L$

O que temos então que provar, para uma linguagem qualquer, a fim de podermos concluir que não é regular, é tudo o que se encontra na segunda coluna desta Tabela 4.4.9.

A prova pode transformar-se num jogo entre duas pessoas, tal como na Tabela 4.4.10.

Tabela 4.4.10. O jogo do lema da bombagem

Jogador 1	Jogador 2
Define a linguagem L	
	Escolhe um *m*
Indica uma cadeia de L	
	Define uma decomposição $\|xy\| \leq m$
Indica um *i* que bombeie para fora	

O jogador 1 tenta provar que não é regular e o jogador 2 o contrário (procurando o *m*).

O jogo só é conclusivo se o jogador 1 ganhar, concluindo-se que a linguagem **não é** regular. A vitória do jogador 2 não permite qualquer conclusão (trata-se de um jogo pouco cavalheiresco …).

O jogador 1 tem que ter uma estratégia para a escolha da cadeia concreta para um qualquer valor de *m* que o jogador 2 coloque sobre a mesa, e aqui reside o cerne da prova.

Para um qualquer *m* que o meu adversário proponha, qual a estrutura da cadeia que devo escolher de modo que ele fique impossibilitado de aí definir uma decomposição apropriada? Como o posso "encostá-lo à parede"?

Esta é o desafio do jogador 1.

Exemplos de aplicação do Lema da Bombagem.

Exemplo 4.4.5. Provar que L=$\{a^n b^n\}$ não é regular

Já encontrámos esta linguagem em diversas ocasiões. Para que um autómato finito a pudesse reconhecer, teria que ser capaz de memorizar o número de *a*'s que já entraram e depois contar um igual número de *b*'s até aceitar a cadeia. Só se o autómato pudesse ter um número arbitrariamente grande de estados, até ao infinito, ele será capaz de reconhecer esta linguagem.

Por aqui é legítima a suspeita de que a linguagem é não regular. Mas provemo-lo formalmente com o lema da bombagem

Tem que se provar que: para qualquer valor de *m,* por muito grande que este seja, existe uma cadeia em que qualquer decomposição $|xy| \leq m$ bombeia para fora da linguagem.

Se me dão um *m*=10, por exemplo, eu apresento a cadeia $a^{10}b^{10}$. A decomposição *xy* tem que ser tal que todos elementos de *y* são *a*'s, pelo facto de $|xy| \leq 10$. Logo bombeando *y* por exemplo duas vezes, aumenta o número de *a*'s, que fica assim maior do que o número de *b*'s e por conseguinte a cadeia não pertence à linguagem.

Se me dão *m*=20, escolho $a^{20}b^{20}$ e acontece coisa semelhante.

E para um m qualquer eu escolho a cadeia $a^m b^m$. Qualquer decomposição xyz bombeia para fora da linguagem.

Exemplo 4.4.6.

Seja o alfabeto $\Sigma=\{0\}$ e a linguagem composta por cadeias apenas de zeros e com um número de zeros igual à sequência dos quadrados perfeitos, isto é 0, 1, 4, 9, 16, …, ou seja, $L= \{0^{i^2},\ i \in N\}$.

Vejamos outra perspectiva da prova, que não recorre directamente ao Lema da Bombagem, mas ainda assim prova por contradição.

Admitamos que a linguagem é regular. Existe por isso um NFA que a aceita. Seja n o seu número (finito) de estados. Ora este NFA tem que aceitar a cadeia com n^2 zeros. Mas como tem apenas n estados, para ler esta cadeia toda tem que passar pelo menos duas vezes por um estado, isto é, tem que ter um percurso fechado, um anel, que passa pelo menos em um estado. Assim pelo facto indiscutível de que $n^2>n$, há pelo menos dois estados iguais desde o incial q_0 até ao final aceitador q_{n2}. Sejam eles q_i e q_j, definindo-se $i-j= p \leq n$. O anel da figura tem comprimento p. Mas se dermos duas voltas ao anel, a cadeia resultante também será aceite pela DFA. Essa cadeia terá (n^2+p) zeros. Será este um quadrado perfeito ? Se for bem faz o autómato em a aceitar. Mas se não for ela não pertence à linguagem.

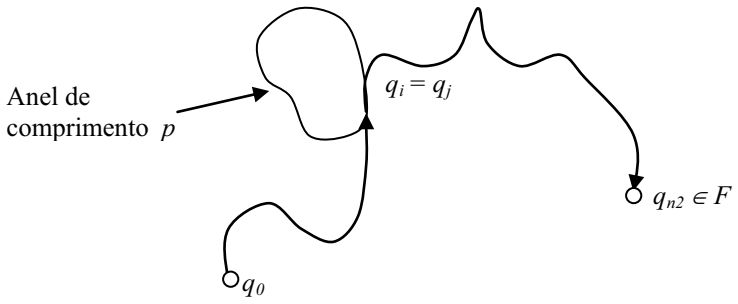


Figura 4.4.7.Ilustração do exemplo 4.4.6

Depois de n^2, o próximo quadrado perfeito é $(n+1)^2=n^2+2n+1$. Ora $n^2+p < n^2+2n+1$ porque $p \leq n$ e por isso n^2+p não é um quadrado perfeito e portanto não pertence à linguagem.

Logo não pode existir um tal NFA, ou seja, a linguagem não pode ser regular.

Esta prova "geométrica" é bastante intuitiva. Se quisermos uma prova baseada no lema da bombagem, poderemo-nos inspirar na prova gráfica.

A contraprova aqui tem que procurar destruir a característica essencial da linguagem: os quadrados perfeitos dos seus zeros.

Se me dás um certo m, apresento-te a cadeia com um número de zeros igual ao quadrado perfeito m^2. Fazendo agora uma decomposição xyz, tal que $|xy|\leq m$, bombeando y, obtêm-se sempre, qualquer que seja y, cadeias com um número de zeros que não é um quadrado perfeito. Isto é, bombeia-se para fora (embora para algumas bombagens se possa eventualmente fazer para dentro). De facto, o y ficará sempre nos primeiros m zeros, e terá no máximo m zeros (nesta caso x é vazia). Analise-se este caso, em que y fica com os m zeros. Bombeando uma vez, o número total de zeros será m^2+m. Ora m^2+m não é um quadrado perfeito; se fosse m^2+2m+1 sê-lo-ia porque seria $(m+1)^2$. Isto basta para negar o Lema da Bombagem e concluir que a linguagem é não regular.

Exemplo 4.4.7

Seja a linguagem dos números primos de zeros $L= \{0^p$, em que p é um número primo$\}$

Vamos ao jogo !

Dás-me um m qualquer e eu apresento-te a cadeia com p zeros, sendo p o número primo igual ou imediatamente superior a m. Isto é possível porque a quantidade de números primos é infinita, e portanto por muito grande que seja m posso encontrar sempre um primo superior. Agora a decomposição xyz coloca y nos primeiros m zeros. Seja r o comprimento de y. Se bombear y um número de vezes igual a p, resulta uma cadeia com $p+rp$ zeros, ou seja $p(1+r)$ zeros, que não é um número primo (porquê? Note-se que $r\geq 1$).

Também se pode fazer uma prova gráfica, por contradição, admitindo que existe um DFA com n estados que aceita a linguagem. Tomamos um número primo p superior a n e concluímos que o DFA terá que conter um anel que fecha no estado $q_i=q_j$. Fazendo agora $j-i=r$ e bombeando p vezes obtém-se um número de zeros igual a $p+rp$ que não é primo e portanto a cadeia bombeada não pertence à linguagem.

Exercício 4.4.1.

Provar que a linguagem $L=\{ww, w \in \Sigma, \ \forall \Sigma\}$ é não regular.

O Lema da Bombagem é por alguns considerado um dos mais interessantes tópicos em Teoria da Computação. O Professor Harry Mairson, da Universidade de Brandeis (http://www.cs.brandeis.edu/~mairson/poems/node1.html) dedica-lhe mesmo um inspirado poema, que aqui se reproduz com autorização.

The Pumping Lemma *by Harry Mairson*

Any regular language *L* has a magic number *p*
And any long-enough word in *L* has the following property:
Amongst its first *p* symbols is a segment you can find
Whose repetition or omission leaves *x* amongst its kind.

So if you find a language *L* which fails this acid test,
And some long word you pump becomes distinct from all the rest,
By contradiction you have shown that language *L* is not
A regular guy, resilient to the damage you have wrought.

But if, upon the other hand, *x* stays within its *L*,
Then either *L* is regular, or else you chose not well.
For *w* is *xyz*, and *y* cannot be null,
And *y* must come before *p* symbols have been read in full.

As mathematical postscript, an addendum to the wise:
The basic proof we outlined here does certainly generalize.
So there is a pumping lemma for all languages context-free,
Although we do not have the same for those that are r.e.

Outros exemplos

0^n1^m, $n>m$

Para qualquer m que me dês apresento a cadeia $0^{m+1}1^m$, que pertence à linguagem e é maior do que *m*. A decomposição *xyz* tem que ser feita de tal modo que o *y* é composto apenas por zeros (pelo menos um zero). Bombeando com $i=0$ apaga-se um 0 e ficando $n=m$ e destruindo-se assim a condição $n>m$

Normalmente em problemas deste tipo, com um desigualdade, segue-se esta estratégia: escolhe-se uma cadeia legítima no limite da desigualdade e depois apaga-se uma parte, bombeando y^0 de modo a destruir a desigualdade.

Bibliografia

An Introduction to Formal Languages and Automata, Peter Linz, 3rd Ed., Jones and Bartelett Computer Science, 2001

Introduction to Automata Theory, Languages and Computation, 2nd Ed., John Hopcroft, Rajeev Motwani, Jeffrey Ullman, Addison Wesley, 2001.

Elements for the Theory of Computation, Harry Lewis and Christos Papadimitriou, 2nd Ed., Prentice Hall, 1998.

Introduction to the Theory of Computation, Michael Sipser, PWS Publishing Co, 1997.