MATERIAL LINGUAGENS FORMAIS E AUTÔMATOS-BCC/FACOM/UFU

PROFA. DRA. RITA MARIA DA SILVA JULIA

BIBLIOGRAFIA DE BASE:

- Introduction à La Calculabilité, Pierre Wolper, Dunod, Paris, 3ª Edição
- Introdução à Teoria de Autômatos, Linguagens e Computação, John E. Hopcroft, Jeffrey D. Ullman, Rajeev Motwani (Tradução da 2ª Edição Americana)
- Linguagens Formais e Autômatos, Paulo Blauth Menezes, 4ª Edição.

1. Introdução

1.1 Motivação

No domínio da Ciência da Computação, o que se deve fazer quando um programa está a um passo de funcionar mas ainda apresenta erros:

a) Tentar retificá-lo de modo a corrigir o erro?
b) Rever sua concepção e recomeçar com um método alternativo de resolução?
c) Concluir com surpresa, a partir da análise do problema, que o mesmo é insolúvel, qualquer que seja o programa proposto, qualquer que seja a tecnologia disponível?

O objetivo central do presente curso é introduzir fundamentos teóricos que servirão de base para cursos futuros em que serão feitas análises mais aprofundadas dos casos que recaem nesta terceira categoria (no presente curso, tais fundamentos serão utilizados para introduzir noções preliminares relativas a tais casos). Assim sendo, a abordagem a ser seguida baseia-se em princípios fundamentais que são independentes da tecnologia (situação similar a dos limites de rendimento dos motores térmicos deduzidos a partir dos princípios da Termodinâmica, os quais independem da tecnologia utilizada para construir os motores). Tais princípios fundamentais vêm sendo desenvolvidos desde 1930 (antes da aparição dos computadores), no quadro da Lógica Matemática, com o objetivo de prover uma definição precisa de prova formal.

Se os limites a serem estudados no presente curso são independentes dos programas e da tecnologia utilizada para construir as máquinas, é natural que se pergunte se eles também se aplicam ao cérebro humano. A resposta a tal pergunta depende do fato de o cérebro poder ou não fazer coisas que um computador não pode e não poderá jamais fazer. Caso se considere que o cérebro é um conjunto

extremamente complexo de neurônios interconectados e que cada neurônio tem um comportamento simples e modelável, não há motivos para não se aceitar que tais limites se apliquem ao cérebro. Caso contrário, cuja análise envolveria profundas considerações filosóficas, tais limites não podem ser imputados ao cérebro.

1.2 Problemas e Programas

A fim de definir os problemas que são solúveis por um programa executado por um computador (soluções chamadas ***procedimentos efetivos***, ou, ***effective procedures***) , duas noções precisam ser estabelecidas:

- A noção de problema;
- A noção de programa executado por um computador

1.2.1 A Noção de Problema

Tal noção será introduzida com base em exemplos.

Exemplo 1: Determinar se um número natural é par ou ímpar é um problema.

Tal exemplo evidencia diversas características da noção de problema:

- Um problema é uma questão genérica, ou seja, aplica-se a um conjunto de elementos (no caso, ao números naturais);
- Cada instância do problema produz uma resposta (35 é par? Não.).
- As noções de programa e problema são independentes. De fato, um programa que resolve um problema não o define. Além disso, vários programas permitem resolver um mesmo problema. A resolução de um problema por um programa requer a definição de uma representação das instâncias do problema sobre a qual o programa atuará.

Exemplo 2: Caso as instâncias do exemplo anterior sejam representadas por números binários, um exemplo de programa capaz de resolvê-lo testa se o último dígito é 0 (instância par) ou 1 (instância ímpar). Um outro programa para resolver o mesmo problema poderia efetuar a conversão da instância para número decimal e checar se o último dígito da representação produzida pertence ou não ao conjunto {0,2,4,6,8} (instância par, caso pertença, ou ímpar, caso contrário).

Exemplo 3:

- Ordenar uma tabela de números é um problema;
- Determinar se um programa pára qualquer que seja os valores de entrada é um problema (problema da parada);
- Determinar se um polinômio a coeficientes inteiros tem raízes inteiras é um problema (décimo problema de Hilbert).

O primeiro desses problemas é solúvel por um programa de computador. Ao longo desta abordagem será mostrado que os dois outros não o são.

Será tratada aqui uma classe limitada de problemas: aqueles com resposta binária (sim ou não, tal como o da parada). Os resultados obtidos para tal classe podem ser generalizados aos problemas não binários (tal como o problema de otimização de grafos).

1.2.2 A noção de Programa

O objetivo desta seção é caracterizar as soluções obtidas por ***procedimentos efetivos***. Uma solução produzida a partir da execução por um computador de um código de máquina compilado pode ser um exemplo de procedimento efetivo. Tal solução contém em si mesma toda a informação necessária para resolver o problema, sem precisar, para tanto, de tomar decisão alguma. Neste sentido, um programa em C poderia ser citado como exemplo de procedimento efetivo. Estendendo a análise, não há procedimentos efetivos capazes de resolver o problema da parada, ou seja, de avaliar se um problema qualquer não tem "loops" ou chamadas recursivas infinitas para qualquer de suas instâncias. Exemplo:

```
recursive function threen (n: integer) : integer;

begin

If (n = 1) then 1

        else if even(n) then threen (n ÷ 2)

                        else threen (3 ∗ n + 1);

end;
```

É impossível definir um procedimento efetivo capaz de resolver o problema da parada para o programa acima. Assim sendo, a título de completeza, seria necessário acrescentar que um programa C é um procedimento efetivo somente se ele pára sempre (ou seja, forneça uma resposta para qualquer de suas instâncias de entrada). Isso implica que para determinar se um programa C é um procedimento efetivo seria necessário resolver o problema da parada, que, por sua vez, não é solúvel por um procedimento efetivo. É uma circularidade desta forma que permitirá demonstrar a insolubilidade do problema de parada.

Apesar dos exemplos acima, a abordagem a ser adotada para formalizar a noção de procedimento efetivo não deve ser baseada nas linguagens de programação usuais, principalmente porque elas somente conseguem definir um procedimento efetivo por meio de um procedimento de interpretação ou de um compilador, o que complica a citada formalização. Contudo, o fato de que tal procedimento de interpretação existe é suficiente para garantir que os resultados aqui demonstrados são aplicáveis aos programas escritos em linguagem de programação usual. Dessa forma, a noção de procedimento efetivo será formalizada através de programas expressos por meio de formas particulares de linguagem de programação que são simples ao ponto de apresentarem um procedimento de interpretação imediato. Tais programas são chamados ***autômatos*** e têm como vantagem o fato de apresentarem comportamentos que são mais facilmente analisáveis. Um autômato é, portanto, um

programa e, não, uma máquina física sobre a qual o programa é executado. A cada ***classe de autômatos*** (linguagem de programação) a ser definida será associado um ***mecanismo de execução*** (bem simples) que permitirá a análise do que se passa durante a execução de um dado ***autômato*** (programa) expresso através dessa classe. Logo, a teoria de autômatos é o estudo dos dispositivos de computação abstratos. Tal estudo foi iniciado entre as décadas de 1940 e 1950 com a introdução de autômatos específicos denominados ***autômatos finitos***, que tinham como finalidade inicial a modelagem das funções cerebrais. O estudo dos autômatos teve como precursores os trabalhos de A. Turing. Antes mesmo da existência dos computadores, na década de 1930, A. Turing estudou um máquina abstrata que tinha todas as características dos computadores atuais, pelo menos no que se refere à sua capacidade de cálculo. Seu objetivo era descrever, com exatidão, o limite do que sua máquina abstrata poderia calcular. Fato extraordinário é que os resultados de suas conclusões são também aplicáveis às máquinas físicas atuais. Também nos anos 50 o lingüista N. Chomsky iniciou o estudo de ***gramáticas formais*** que têm estreito relacionamento com os autômatos e que hoje servem de base para alguns importantes componentes de software, incluindo os compiladores.

E a noção de Algoritmo ? Até hoje não existe uma definição única e aceita por todos. Para Minsky (1967), por exemplo, um algoritmo (um conjunto de regras que especificam a cada instante o próximo comportamento) é sinônimo de procedimento efetivo.

1.3. Objetivos do Curso

- Apresentar as linguagens formais, as máquinas abstratas reconhecedoras (autômatos) e as gramáticas principais da Hierarquia de Chomsky, mostrando o relacionamento existente entre cada tipo de linguagem, os autômatos que as reconhecem, e as gramáticas que as geram.

- formalizar a noção de procedimento efetivo e de analisar os procedimentos efetivos utilizados como solução de problemas.

- Evidenciar a linguagem reconhecida por um autômato como uma expressão de sua computabilidade e, a partir daí, aprofundar a noção de indecidibilidade e discutir os limites da computação convencional.

2. A Formalização de Problemas

Para que um problema seja resolvido por um procedimento efetivo, suas instâncias devem ser representadas de um modo inteligível para ele. Para tanto, será adotada aqui uma representação geral baseada na seguinte abstração: como no nível do código de máquina os dados são representados por seqüências de bits, cada instância do problema será aqui representada por uma cadeia finita de símbolos a serem escolhidos em função do problema. Exemplos de conjuntos de símbolos: {0,...,9}, para um problema envolvendo os números naturais, ou {a, ...,z} para problema envolvendo instruções de uma linguagem de programação.

2.1.1 Alfabetos e Palavras

Definição 2.1. : Um alfabeto Σ é um conjunto finito de símbolos cujo tamanho é dado pelo número de elementos do conjunto. Ex: Σ = {a, b} tem tamanho 2.

Definição 2.2. : Uma palavra w (também chamada cadeia, string ou seqüência) definida sobre um alfabeto é uma seqüência finita de símbolos do alfabeto. O comprimento de uma palavra w, representado por |w|, indica a quantidade de símbolos que a compõem, sendo que:

- |w| = 0 : refere-se a uma palavra vazia (referenciada por *e*, ε ou λ);
- Em uma palavra de tamanho n, w(i), onde 1 ≤ i ≤n, corresponde ao símbolo que ocupa a i-ésima posição na palavra;
- Sendo $n \geq 0$ e $w \neq \varepsilon$: $\boldsymbol{w^n}$ representa a palavra obtida por *n* justaposições da palavra *w*. Exs: $(ab)^2$ = abab; $(ab)^0 = \varepsilon$; a^2b^3 = aabbb;
- $\forall w \neq \varepsilon$, $w^0 = \varepsilon$; ε^0 é **indefinido;**
- Sendo $n > 0$ e $w \neq \varepsilon$: $\varepsilon^n = \varepsilon$; $w^n = w^{n-1}w$
- $\forall w$, $w\varepsilon = \varepsilon w = w$; $\varepsilon\varepsilon = \varepsilon$

2.1.2 A Representação dos Problemas

Todos os dados manipulados em computação (instâncias dos problemas) podem ser representados por palavras (cadeias de caracteres). Exs: Uma imagem pode representada por uma seqüência das intensidade dos seus pontos e um som pode ser representado por uma seqüência de números. Neste curso, a função que converte uma instância do problema para sua representação através de uma cadeia de caracteres será denominada ***função de codificação***.

Seja um problema binário cujas instancias foram codificadas por palavras definidas num alfabeto Σ. O conjunto de todas as palavras definidas em Σ pode ser subdividido em 3 grupos:

- Instâncias Positivas do Problema: são as palavras que representam as instâncias do problema para as quais a resposta dada pelo procedimento efetivo é SIM;
- Instâncias Negativas do Problema: são as palavras que representam as instâncias do problema para as quais a resposta dada pelo procedimento efetivo é NÃO;
- Palavras que não representam instâncias do problema

Freqüentemente, os dois últimos grupos são aglutinados em um único grupo de instâncias negativas.

Um problema pode ser caracterizado pelo conjunto de palavras que são instâncias positivas. Exemplo:

- alfabeto Σ = {0,...,9,a,...,z}
- Problema dos números pares
- 19 : resposta NÃO (instância negativa)
- 26 : resposta SIM (instância positiva)
- 2a9 : resposta : não é uma instância do problema (é um tipo de NÃO, logo, assumindo a aglutinação acima, seria uma instância negativa).

3. Linguagem

Definição 3.1: Uma linguagem é um conjunto de palavras definidas a partir de um mesmo alfabeto.

Logo, um problema é caracterizado pela linguagem que codifica as instâncias positivas do problema e a resolução desse problema consiste em reconhecer as palavras dessa linguagem que caracterizam as instâncias positivas do problema. Exs:

- Os conjuntos {ε,aaaaa,a,bbbbbb}, {aab,aaaa,ε,a,b,abb} e Ø (linguagem vazia ou desprovida de palavras) são linguagens (finitas) no alfabeto {a,b};
- {ε,0,1,00,01,10,11,000,001,...} é a linguagem de todas as palavras que podem ser formadas no alfabeto {0,1}. Da mesma forma, o conjunto de todas as representações binárias dos números pares é uma linguagem;
- O conjunto das palavras representando os programas em C que param sempre é uma linguagem;
- { ε , a } = { a} .

OBSERVAÇÕES: A linguagem vazia Ø (também representada por *{ }*) é distinta da linguagem {ε} (linguagem que contém a palavra vazia).

3.1 Descrição das Linguagens

Não existe uma representação (notação) que descreva qualquer linguagem. Para descrever uma linguagem finita, basta enumerar as palavras que pertencem a ela. Contudo, através de tal estratégia não é possível definir uma linguagem infinita. A notação a ser introduzida aqui permite a descrição de todas as linguagens finitas e de determinadas linguagens infinitas. Ela se baseia na representação direta de certas linguagens elementares (tal como aquelas contendo uma única palavra formada de um único símbolo do alfabeto), bem como na descrição de linguagens mais complexas por meio da aplicação de operações a linguagens mais simples.

3.1.1 Operação sobre as linguagens

Considerem-se duas linguagens L_1 e L_2.

- A **união** de L_1 e L_2 é a linguagem formada por todas as palavras pertencentes a L_1 e todas as palavras pertencentes a L_2, ou seja:
  $L_1 \cup L_2 = \{w / w \in L_1 \text{ ou } w \in L_2\}$;
- A **concatenação** de L_1 e L_2 é a linguagem formada por todas as palavras produzidas a partir da seguinte aglutinação: uma palavra de L_1 seguida de uma palavra de L_2, ou seja, $L_1 . L_2 = \{w / w = xy \text{ com } x \in L_1 \text{ e } y \in L_2\}$;
- O **fechamento iterativo** de L_1 ou fechamento de Kleene (Kleene closure ou Kleene star) é o conjunto das palavras formadas por uma concatenação finita de palavras de L_1, ou seja:
  $L_1^* = \{w / \exists(k \geq 0 \text{ e } w_1 , ..., w_k \in L_1) \text{ tais que } w = w_1 w_2 \ ... w_k\}$;

- O **complemento** L_1^C de uma linguagem L_1 é o conjunto de todas as palavras sobre o mesmo alfabeto que não pertencem a L_1, ou seja, $L_1^C = \{w / w \notin L_1\}$.

EXEMPLOS:

a) $L_1 = \{a, b\}$; $L_2 = \{c\}$

$L_1 \cup L_2 = L_2 \cup L_1 = \{a, b, c\}$

$L_1 . L_2 = \{ac, bc\}$

$L_2 . L_1 = \{ca, cb\}$

$L_1^* = \{ \varepsilon, a, b, ab, ba, aa, bb, aaa, bbb, abab, ..., bbabaaa, ... , abababab, ...\}$

Obs: note que o fechamento de L_1 corresponde ao conjunto infinito formado pela palavra vazia e por qualquer seqüência finita de palavras compostas sobre o alfabeto {a,b}.

b) $L_1 = \{\varepsilon, a, b\}$; $L_2 = \{c\}$

$L_1 \cup L_2 = L_2 \cup L_1 = \{\varepsilon, a, b, c\}$

$L_1 . L_2 = \{\varepsilon c, ac, bc\}$

$L_2 . L_1 = \{c\varepsilon, ca, cb\}$

$L_1^* = \{ \varepsilon, \varepsilon\varepsilon, a, b, ab, ba, aa, bb, aaa, bbb, abab, ... , a\varepsilon b, bb\varepsilon\varepsilon aba, ... , \varepsilon\varepsilon\varepsilon, bbabaaa, ... , abababab, ...\}$

Observações decorrentes das definições das operações:

- Qualquer que seja a linguagem L , $\{ \} . L = L . \{ \} = \{ \}$;
- Qualquer que seja a palavra w, $\{ \varepsilon \} . \{ w \} = \{ w \} . \{ \varepsilon \} = \{ w \}$
- $\{ \varepsilon \} . \{ \varepsilon \} = \{ \varepsilon\varepsilon \} = \{ \varepsilon \}$

4. As Linguagens Regulares

Definição 4.1. O conjunto **R** das **linguagens regulares** sobre um alfabeto Σ é o MENOR conjunto de linguagens tal que:

(1) $\emptyset \in R$; $\{ \varepsilon \} \in R$;
(2) $\{a\} \in R$ para todo $a \in \Sigma$;
(3) Se $A, B \in R$, então $A \cup B$, $A.B$ e $A^* \in R$.

EXEMPLO:

Conjunto infinito R de linguagens regulares sobre o alfabeto $\Sigma = \{a,b\}$:

R = { $\emptyset$, { ε }, {a}, {b}, {ε, a}, {ε, b}, {a, b}, {ε, a, b}, ... ,

{εa}, {εb}, {$\varepsilon\varepsilon$}, {aε}, {ab}, {aa}, {bε}, {ba}, {bb}, {aaa},{bbb}, {ab, ba}, {abab}, ... ,
{ε, b, bb, bbb, ...}, ..., {b, bb, bbb, bbbb, ...}, { ε, ab, abab, ababab, ...}, ..., {ab, abab, abababab,...}, ...
{a, ε, ab, abab, ababab, ...,}, ...,

{ε, εa}, {ε, εb}, {ε, $\varepsilon\varepsilon$}, {aεbε, bεa}, ...

...

}

OBS: note que qualquer conjunto finito de palavras possíveis sobre o alfabeto Σ é uma linguagem que pertence a R, ou seja, corresponde a uma linguagem regular sobre tal alfabeto, pois é formado a partir de alguma(s) das condições estabelecidas na definição 4.1. Além disso, ainda com base na definição, já é possível perceber que ALGUNS conjuntos infinitos também pertencem a R, tal como a linguagem formada por palavras que correspondem a seqüências de ***ab***. Contudo, isso será visto com mais detalhes ao longo do curso.

O estudo das linguagens regulares, ou do Tipo 3, pode ser abordado através dos seguintes formalismos:

- Denotacional: baseado nas Expressões Regulares;
- Operacional ou Reconhecedor: baseado nos Autômatos Finitos;
- Axiomático ou Gerador: baseado nas Gramáticas Regulares

Tais formalismos serão apresentados nas seções subseqüentes.

5. Expressões regulares

Uma expressão regular é uma notação utilizada para representar uma linguagem regular. Tal notação indica como um conjunto regular é obtido a partir de conjuntos regulares elementares.

Definição 5.1. As **expressões regulares** para um alfabeto Σ são as expressões formadas pelas seguintes regras:

(1) Ø, ε e os elementos de Σ são expressões regulares;
(2) Se α e β são expressões regulares, então (α ∪ β) , (αβ) , (α)* são também expressões regulares

Observa-se que as próprias expressões regulares representam uma linguagem (na ótica das definições apresentadas), pois cada expressão regular é uma cadeia de caracteres sobre o seguinte alfabeto Σ´: Σ´ = Σ ∪ {), (, Ø, ∪, *, ε }. Isso faz com que a definição de linguagem adotada aqui seja útil para definir não somente a codificação de problemas, mas, também, a sintaxe das expressões (a mesma abordagem é usada nas linguagens de programação).

A título de praticidade, serão adotadas a partir daqui as seguintes simplificações sintáticas nas expressões regulares:

- Os parênteses serão omitidos sempre que isso não trouxer ambigüidades sintáticas;
- O operador ∗ será usado predominantemente na forma de um índice superior: Ex: a^* ao invés de (a)∗

Definição 5.2. A linguagem L(ξ) representada por uma expressão regular ξ é definida da seguinte forma:

(1) L(Ø) = Ø , L(ε) = {ε}
(2) L(a) = {a} para todo a ∈ Σ
(3) L((α ∪ β)) = L(α) ∪ L(β)
(4) L((αβ)) = L(α) . L(β)
(5) L($(\alpha)^*$) = L(α)* (a linguagem regular representada pela expressão regular $(\alpha)^*$ é o fechamento de KLEENE da linguagem L(α))

Teorema 5.1. Uma linguagem é regular se, e somente se, ela é representada por uma expressão regular.

OBS: a demonstração deste teorema pode ser feita facilmente por indução (a ser feita em sala de aula).

Exemplos envolvendo linguagens regulares representadas por expressões regulares no alfabeto Σ = {a,b}:

a) Expressão regular: (a ∪ (ab)) representa a linguagem regular {a,ab}. De fato:

L((a ∪ (ab))) = L(a) ∪ L((ab)) =
{a} ∪ L(a) . L(b) = {a} ∪ {a}.{b} = {a} ∪ {ab} = {a, ab}

b) Expressão regular: (a(b)*) representa a linguagem regular correspondente ao conjunto de palavras formadas a partir do alfabeto dado cuja primeira letra é *a* e as demais letras são seqüências finitas crescentes de *b* (começando com nenhum *b*). De fato:

L((a(b)*)) = L(a) . L((b)*) =
{a} . L(b)* = {a} . {b}* = {a} . { ε , b, bb, bbb, ...} = {a ε, ab, abb, abbb, abbbb, ...}

c) Expressão regular: (ab ∪ (b)*) representa a linguagem regular formada pela palavra ab e pelas palavras que são seqüencias finitas crescentes de *b* (começando com nenhum *b*). De fato:

Linguagem regular representada por ela: L(((ab) ∪ (b)*)) = L(ab) ∪ L((b)*) =
L(a) . L(b) ∪ L(b)* = {ab} ∪ {b}* = { ab, ε, b, bb , bbb, ...}

d) Expressão regular: (a ∪ b)* : representa a linguagem regular composta por todas as palavras possíveis a partir do alfabeto {a,b}. De fato:

L((a ∪ b)*) = L((a ∪ b))* = (L(a) ∪ L(b))* = ({a} ∪ {b})* = {a,b}* =

{ ε , a, b, b, aa, bb, aaa, bbb, ..., ab, ba, abba,abab,baab, aab, bba, aabbabab, bbaababa, ...}

OBS: Tal exemplo baseado no alfabeto {a, b} pode ser generalizado da seguinte forma: a linguagem regular correspondente ao conjunto de todas as palavras definidas a partir de um alfabeto Σ = {a_1, ... , a_n} é representada pela expressão regular (a_1 ∪ ... a_n)*, comumente abreviada como **Σ***.

e) Expressão regular: (a ∪ b) (a ∪ b)* : representa a linguagem regular composta por todas as palavras NÃO VAZIAS possíveis a partir do alfabeto {a,b}. De fato:

L((a ∪ b) (a ∪ b)*) = L((a ∪ b)) . L((a ∪ b)*) = L(a) ∪ L(b) . L((a ∪ b)*) =
{a} ∪ {b} . L((a ∪ b)*) = {a} ∪ {b} . {a,b}* = {a,b} . **Σ*** =

{ a, b, b, aa, bb, aaa, bbb, ..., ab, ba, abba,abab,baab, aab, bba, aabbabab, bbaababa, ...},

que corresponde a uma linguagem regular composta por todas as palavras que têm *a* ou por *b* por primeiro símbolo e como símbolos remanescentes uma seqüência que corresponde a qualquer palavra possível a partir do alfabeto {a, b}. Logo, a linguagem regular L((a ∪ b) (a ∪ b)*) corresponde à linguagem regular L((a ∪ b)*) desfalcada da palavra vazia.

OBS: Tal exemplo baseado no alfabeto {a, b} pode ser generalizado da seguinte forma: A linguagem regular correspondente ao conjunto de todas as palavras não vazias (diferentes de ε) definidas a partir de um alfabeto $\Sigma = \{a_1, \ldots, a_n\}$ é representada pela expressão regular $(a_1 \cup \ldots \cup a_n)(a_1 \cup \ldots \cup a_n)^*$, comumente abreviada como $\Sigma\,\Sigma^*$, ou como Σ^+.

f) Expressão regular: $(a \cup b)^* a\ (a \cup b)^*$: corresponde à linguagem regular das palavras compostas sobre o alfabeto {a,b} que contêm pelo menos um *a* em qualquer posição. De fato:

$L((a \cup b)^* a\ (a \cup b)^*) = L((a \cup b)^* a) \,.\, L(a \cup b)^* = L(\Sigma^*).\, L(a).\, L(\Sigma^*) =$
$L(\Sigma^*) \,.\, \{a\} \,.\, L(\Sigma^*)$

g) Expressão regular: $(a \cup b)^* a$: corresponde à linguagem regular de todas as palavras compostas sobre o alfabeto {a,b} que terminam em *a*. De fato:

$L((a \cup b)^* a) = L((a \cup b)^*) \,.\, L(a) = L(\Sigma^*) \,.\, \{a\}$

h) $(a^*b)^* \cup (b^*a)^* = (a \cup b)^*$, ou seja, as expressões regulares $(a^*b)^* \cup (b^*a)^*$ e $(a \cup b)^*$ representam a mesma linguagem (ou conjunto) regular. De fato:

- $L((a^*b)) = \{a\}^* \,.\, \{b\} = \{ b, ab, aab, aaab, aaaab, \ldots \}$, que representa a linguagem regular formada por todas as palavras *W* sobre o alfabeto {a,b} que são uma seqüência de uma quantidade inteira n ($n \geq 0$) de símbolos *a* que termina com um *b* , a ser representada aqui por:

  $W = a^n b$ (ver seção 2.1.1)

  Logo, o fechamento $L((a^*b)^*)$ desta última linguagem representa a linguagem regular formada pela palavra vazia e por todas as palavras W_1 sobre o alfabeto {a, b} que correspondam a seqüências de palavras do tipo W, ou seja:

  $W_1 = W \ldots W = a^{n1} b \ldots a^{nk} b$, onde n1 e nk são inteiros ≥ 0.

  Assim sendo, tal fechamento corresponde à linguagem regular constituída pela palavra vazia e por todas as palavras sobre o alfabeto {a, b} que terminam com o símbolo *b*.

- $L((b^*a)) = \{b\}^* \,.\, \{a\} = \{ a, ba, bba, bbba, bbbba, \ldots \}$, que representa a linguagem regular formada por todas as palavras *W* sobre o alfabeto {a,b} que são uma seqüência de uma quantidade inteira n ($n \geq 0$) de símbolos *b* que termina com um *a* , a ser representada aqui por:

  $W = b^n a$

Logo, o fechamento $L((b^*a)^*)$ desta última linguagem representa a linguagem regular formada pela palavra vazia e por todas as palavras W_1 sobre o alfabeto {a, b} que correspondam a seqüências de palavras do tipo W, ou seja:

$W_1 = W...W = b^{n1}a...b^{nk}a$, onde n1 e nk são inteiros ≥ 0.

Assim sendo, tal fechamento corresponde à linguagem regular constituída pela palavra vazia e por todas as palavras sobre o alfabeto {a, b} que terminam com o símbolo *a*.

- Mas, por definição, $L((a^*b)^* \cup (b^*a)^*) = L((a^*b)^*) \cup L((b^*a)^*)$. Logo, a partir da análise acima, conclui-se que $L((a^*b)^* \cup (b^*a)^*)$ é a linguagem regular formada por todas as palavras geradas a partir do alfabeto {a, b}, o que demonstra o enunciado proposto que afirma que $(a^*b)^* \cup (b^*a)^* = (a \cup b)^*$.

6. Os Autômatos Finitos

Conforme dito anteriormente, os autômatos finitos são um formalismo que, como as expressões regulares, permitem caracterizar as linguagens regulares. Além disso, ele serve também como ferramenta auxiliar para delimitar a noção de procedimento efetivo. Independentemente da relação deles com a noção de procedimento efetivo, os autômatos finitos podem ser muito úteis, também, na resolução de certos problemas da informática, tais como os problemas de busca de cadeias de caracteres em um texto. A título de entender em que sentido o conceito de autômato finito é uma modelagem da noção de procedimento efetivo, usar-se-á aqui a noção da execução de um programa fixo por um computador, o qual necessita para tal de:

- Um processador que conta com um conjunto de registradores (inclusive o *contador de programa*, ou PC);
- Uma memória contendo o programa (como ele é fixado, pode ser ROM);
- Uma memória para armazenar os dados.

Supondo que, no início, os dados a serem tratados estejam na memória, tal execução corresponde a uma sucessão de ciclos definidos conforme se segue :

(1) A instrução apontada pelo PC é transferida da memória para o processador;
(2) O processador executa tal instrução, o que tem como efeito modificar o conteúdo dos registros e/ou da memória de dados;
(3) PC é incrementado (ou alterado, caso a instrução executada seja de desvio);

Como descrever uma sucessão de tais ciclos de modo mais abstrato? Para determinar o efeito de um ciclo (ou instrução), basta conhecer o conteúdo da memória de dados e dos registradores - tal conteúdo é denominado ***estado*** do computador. Logo,

objetivamente, o ***estado*** de um sistema qualquer em um dado momento é toda informação necessária para predizer sua evolução futura. Dispondo-se desse conhecimento, pode-se determinar, sem ambigüidade, qual será o conteúdo da memória de dados e dos registradores (ou *estado*) após a execução de um ciclo (ou de uma instrução). Assim sendo, a execução de uma instrução pode ser vista como uma transformação de tais conteúdos (ou dos *estados*). O efeito da execução de uma instrução depende, apenas, do estado *s* que antecede sua execução, não apresentando dependência alguma do histórico que levou ao estado *s*. Cada variação no conteúdo da memória de dados ou dos registradores define um estado diferente. Conseqüentemente, cada ciclo transforma o estado da máquina. Tal transformação depende exclusivamente do programa e da máquina, podendo ser vista como uma função do conjunto de estados da máquina nele mesmo: a chamada ***função de transição***. Logo, basta conhecer tal função e o estado imediatamente anterior ao começo da execução de um programa (denominado ***estado inicial***) para saber com precisão o que ocorrerá durante a execução do mesmo. Para uma dada máquina, o número de estados possíveis é, a priori, finito e fixo, ainda que gigantesco.

Exemplo 6.1: Uma máquina de 8 Mbytes de memória e 16 registradores de 32 bits tem $2^{67109376}$ estados possíveis.

Considerem-se os seguintes fatos: a princípio, podem-se examinar todos os elementos de um conjunto finito; e, na prática, não se chegaria jamais ao fim da tarefa de se examinar os $2^{67109376}$ estados do exemplo acima. Com base nesses fatos, seria verdadeiramente legítimo raciocinar assumindo que a gigantesca quantidade de $2^{67109376}$ estados da máquina do exemplo representa um número "finito" de estados? Nesta seção será considerado que sim. Posteriormente, isso será rediscutido. Chega-se assim à seguinte primeira modelagem de um programa executado por uma máquina:

- Um conjunto finito de estados;
- Uma função de transição definida sobre tal conjunto;
- Um estado inicial.

Uma execução do programa pela máquina é representada pela seqüência de estados obtidos por sucessivas aplicações da função de transição ao estado inicial.

A hipótese de números de estados possíveis finitos tem as seguintes conseqüências extremamente fortes:

- Para cada estado inicial é possível definir qual será a seqüência de estados. Tal seqüência pode-se repetir infinitamente, mas como o número de estados é finito, fatalmente um mesmo estado aparecerá duas vezes na seqüência. A partir de tal momento, a parte da seqüência compreendida entre as duas aparições deste estado se repetirá indefinidamente, tornando desnecessário o cálculo da continuidade da seqüência;
- A modelagem adotada implica que a quantidade de estados iniciais possíveis é finita. Logo, o número de dados distintos que podem ser tratados é finito e fixo. Contudo, os problemas na presente abordagem são modelados por linguagens, o que significa que um procedimento efetivo para resolvê-lo deve poder ser

aplicado à uma infinidade de palavras diferentes, que é uma alternativa impossível no modelo proposto.

Considerando tais conseqüências, o modelo proposto deve ser adaptado de modo a permitir que palavras com comprimento arbitrário possam ser tratadas. Futuramente isso será feito a partir da suposição de que a memória da máquina é infinita. Mas, por enquanto, será mantida a hipótese de um número finito de estados e serão estudadas as conseqüências de tal escolha. Assim sendo, considera-se que a palavra a ser tratada será posta à disposição do programa caracter a caracter (ignorando a memória). Para tanto, supõe-se que a máquina conta com um dispositivo de leitura apto a consultar, a cada instante, o caracter seguinte da palavra tratada. Neste caso, cada ciclo tratará um desses caracteres, da seguinte forma:

- Se os caracteres são fornecidos mais lentamente do que os ciclos da máquina, supõe-se que esta fique bloqueada até que o caracter seguinte esteja disponível;
- Na modelagem adotada, um ciclo equivale a uma aplicação da função de transição. Caso sejam necessários diversos ciclos para tratar um caracter, tais ciclos podem ser substituídos por um ciclo único, bastando, para tanto, redefinir a função de transição de modo que ele modele tal sucessão de ciclos através de uma única transição. Além disso, como já mencionado, as execuções infinitas não são alvo de interesse na hipótese de um número finito de estados. Pode-se assim supor que o tratamento de cada caracter se faz em um número finito de ciclos.
- Pelo mesmo motivo, pode-se supor que, após ter lido o último caracter, a máquina pára e fornece a resposta.

Há dois tipos de autômatos finitos: determinísticos e não determinístico, sendo que os segundos podem ser convertidos nos primeiros pela eliminação do não determinismo.

6.1 Autômatos Finitos Determinísticos:

6.1.1 Descrição

Um **autômato finito determinístico**, a partir deste ponto e nesta subseção simplesmente referenciado por **autômato** ou **autômato finito**, é constituído por:

- Uma fita (*ruban* ou *input tape*) de entrada onde se aloca a palavra de entrada a ser tratada. Tal fita é dividida em casas, sendo que cada casa abriga um caracter da palavra de entrada. O autômato possui uma cabeça de leitura (*tête de lecture* ou *read head*) que indica a posição do caracter seguinte da palavra de entrada a ser lido (ver Figura 6.1).

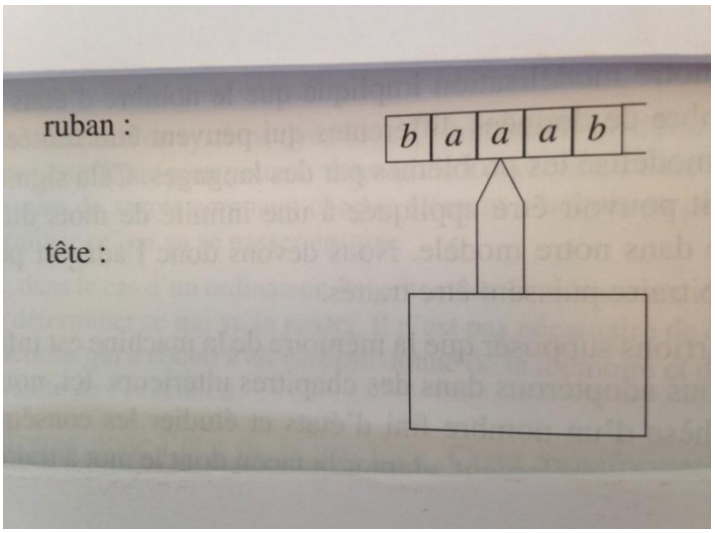


Figura 6.1 Autômato Finito Determinístico

- Um conjunto finito de estados, distinguindo-se dentre eles:
  (1) Um estado inicial: estado do autômato no início da execução;
  (2) Estados de aceitação: definem as palavras aceitas pelo autômato.

- Uma função de transição: indica o estado subseqüente a cada estado e símbolo lido;

A execução de um autômato finito sobre uma dada palavra de entrada pode ser descrita como se segue :

- Inicialmente, a máquina se encontra no estado inicial e a palavra de entrada é alocada na fita (a cabeça de leitura aponta para seu primeiro caracter);
- A cada etapa de sua execução, o autômato lê um símbolo da fita, determina o estado subseqüente (através da função de transição) e avança a cabeça de leitura para a casa seguinte da fita;
- O autômato pára tão logo toda a cadeia tenha sido lida. A palavra tratada é aceita (o autômato responde que ela pertence à linguagem) se a máquina se encontra então em um estado de aceitação.

6.1.2 Formalização

Autômato finito determinístico: definido por uma quíntupla $M = (Q , \Sigma, \delta, s , F)$, onde:

- Q é um conjunto finito de estados possíveis;
- Σ é um alfabeto;
- $\delta : Q \times \Sigma \rightarrow Q$ é a função de transição (nos autômatos finitos determinísticos, cada aplicação da função de transição trata um único símbolo da palavra executada).
- $s \in Q$ é o estado inicial;
- $F \subseteq Q$ é o conjunto de estados de aceitação.

Intuitivamente um autômato é uma máquina que resolve um problema, ou seja, que reconhece (aceita) uma certa linguagem. Logo, atribuir um significado a um autômato consiste em definir a linguagem que ele aceita. A ***configuração*** de um autômato finito representa a informação necessária para determinar sua evolução futura, compondo-se de um par formado pelo estado corrente do autômato e da parte remanescente da palavra de entrada que falta para ser tratada. Assim sendo, uma configuração é um elemento $(q, w) \in Q \times \Sigma^*$. Observe-se que a ***configuração*** de um autômato finito é, de fato, seu ***estado*** (tal como definido no início da seção 6). Logo, a rigor, os elementos de Q não representam o que se denomina "estado" de um autômato.

Definição 6.1 A configuração (q´, w´) é *derivável em uma única etapa* da configuração (q, w) através de uma máquina M (escreve-se **$(q,w) \vdash_M (q', w')$**) se:

- $w = \sigma w'$ (σ : primeiro caracter de w; w´: w desfalcada do primeiro caracter);
- $q' = \delta(q, \sigma)$

Igualmente, uma configuração é *derivável* de uma outra (subtendido em diversas etapas) se ela é obtida a partir desta última por uma seqüência finita (eventualmente de comprimento nulo) de derivações em uma única etapa, ou seja:

Definição 6.2 A configuração (q´, w´) é derivável da configuração (q,w) através de uma máquina M (escreve-se **$(q,w) \vdash^*_M (q', w')$**) se:

- Existem $k \geq 0$ e configurações (q´, w´), $0 \leq i \leq k$, tais que:
  (1) $(q,w) = (q_0, w_0)$;
  (2) $(q',w') = (q_k, w_k)$;
  (3) Para todo $0 \leq i \leq k$ **,** $(q_i,w_i) \vdash_M (q_{i+1}, w_{i+1})$ **.**

Definição 6.3 A execução de uma palavra w por um autômato é representada pela seguinte seqüência de configurações (sendo *s* o estado inicial e ε a palavra vazia):

$(s,w) \vdash_M (q_1, w_1) \vdash_M (q_2, w_2) \vdash_M \ldots \vdash_M (q_n, \varepsilon)$

- Nos autômatos finitos determinísticos cada palavra define uma única execução;
- Uma palavra w é aceita por um autômato se o último estado da execução do autômato sobre w (no caso, q_n) é um estado de aceitação (ver Definição 6.4).

Definição 6.4 Uma palavra w é aceita por um autômato M se, e somente se:

$(s,w) \vdash^*_M (q, \varepsilon)$ e $q \in F$.

Exemplo: Seja um autômato $M = (Q , \Sigma, \delta, s , F)$ cujo alfabeto é $\Sigma = \{a, b\}$ e seja w = ababa. A execução de w pelo autômato é:
$(s,ababa) \vdash_M (q_1, baba) \vdash_M (q_2, aba) \vdash_M (q_3, ba) \vdash_M (q_4, a) \vdash_M (q_5, \varepsilon)$

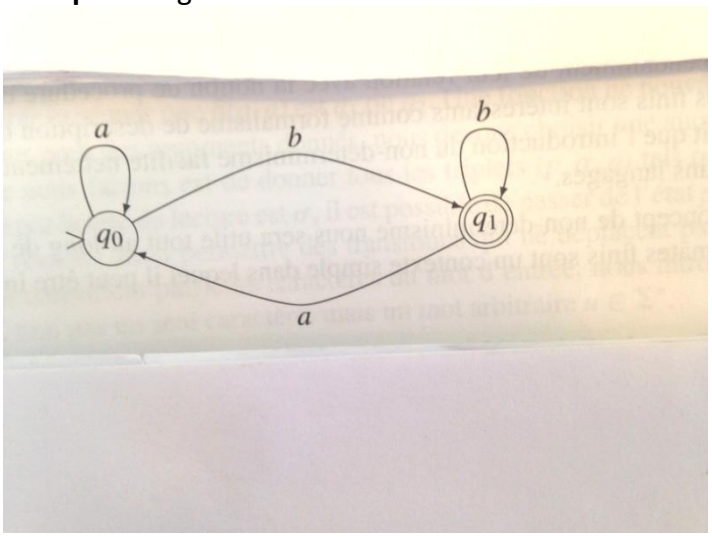
Definição 6.5 Uma linguagem aceita por um autômato M (com notação *L(M)*) é o conjunto de palavras definido da seguinte forma:

$L(M) = \{w \in \Sigma^* \mid (s, w) \vdash_M^* (q, \varepsilon) \text{ com } q \in F\}$.

6.1.3 Representação e Exemplos

A quíntupla correspondente a um autômato pode também ser representada por um grafo em que: cada estado do autômato é representado por um vértice do grafo e cada relação de transição é representada por um arco etiquetado. Se $\delta(p,\sigma) = q$, o grafo contém um arco etiquetado por σ conectando os vértices p e q. Estado inicial: ponta de seta; estados de aceitação: representados por circunferência dupla. Saliente-se que nessa representação em grafo dos autômatos finitos determinísticos, em decorrência da definição da função de transição, em cada estado do grafo é necessária a explicitação de uma (e somente uma) transição para cada símbolo do alfabeto (pois uma função é um tipo de relação que tem de ser TOTAL, ou seja, não pode haver elemento do domínio $Q \times \Sigma$ que não esteja associado a um elemento do contra-domínio Q).

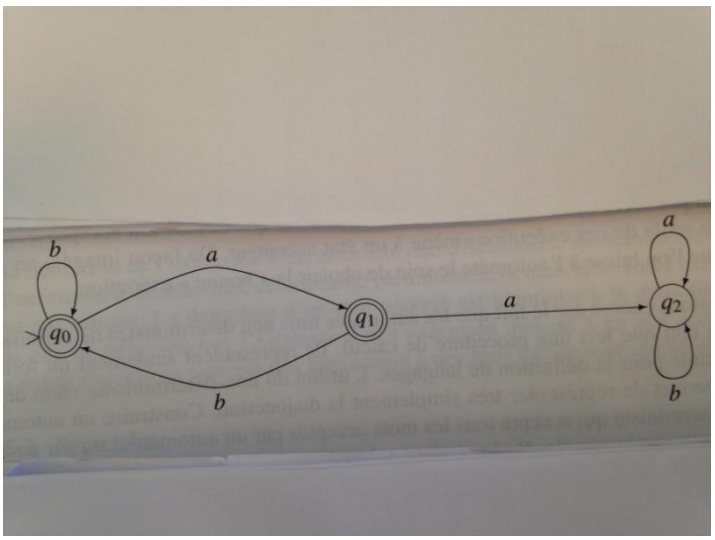
Exemplo 1: o grafo



representa o seguinte autômato que aceita as palavras terminadas em b : $Q = \{q_0 , q_1 \}$; $\Sigma = \{a, b\}$; $s = q_0$, $F = \{ q_1 \}$ e
δ:

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

Exemplo 2: O autômato abaixo aceita a linguagem {w|w não contém dois símbolos *a* consecutivos} :



6.2 Autômatos Finitos não Determinísticos

Representam generalizações dos autômatos finitos determinísticos que permitem diversas alternativas de escolha de execução. Apesar de tal possibilidade se contrapor à noção de procedimento efetivo, ela se justifica pelos seguintes motivos:

- Ela permite que diversas alternativas sejam investigadas durante a modelagem de um procedimento efetivo.
- Ela facilita enormemente a descrição de certas linguagens.

6.2.1 Descrição

Os autômatos finitos não determinísticos se caracterizam por permitir:

- Aplicação de uma transição a qualquer sub-palavra (inclusive ε) da palavra executada (e não apenas a um único símbolo, como nos determinísticos);
- Mais de uma transição correspondente a um mesmo símbolo em um mesmo estado (conforme exemplo 2 desta subseção).

Logo, tais autômatos podem ter diversas execuções de uma mesma palavra. Para que tal palavra seja aceita, basta que uma dessas execuções leve a um estado de aceitação. Apesar de esses autômatos não definirem um procedimento de cálculo, eles são extremamente úteis para representar disjunções de linguagens, ou seja, para representar todas as palavras aceitas por mais de um autômato.

6.2.2 Formalização

Formalmente, os autômatos finitos não determinísticos diferem dos determinísticos apenas por apresentarem uma relação de transição no lugar de uma função de transição, uma vez que um mesmo estado pode apresentar diversos sucessores através de um mesmo símbolo da palavra executada. Logo, eles podem ser representados por uma quíntupla M = (Q , Σ, Δ, s , F), onde:

- Q é um conjunto finito de estados possíveis;
- Σ é um alfabeto;
- $\Delta \subset (Q \times \Sigma^* \times Q)$ (sub-conjunto finito) é a relação de transição (nos autômatos finitos não determinísticos, cada aplicação da função de transição trata uma sub-palavra da palavra executada);
- $s \in Q$ é o estado inicial;
- $F \subseteq Q$ é o conjunto de estados de aceitação.

Assim sendo, as transições são definidas pelas triplas *(p, u, q)*, onde *u* é uma sub-palavra da palavra executada, com a seguinte interpretação: se a sub-palavra apontada pela cabeça de leitura é *u*, é possível passar do estado *p* ao estado *q*. É importante ressaltar que as transições nos autômatos finitos não precisam ser TOTAIS, pois a relação de transição é um subconjunto de $(Q \times \Sigma^* \times Q)$. Conseqüentemente, tal relação não precisa definir todas as transições possíveis relativas aos pares $Q \times \Sigma^*$. Na prática, isso significa que, em tais autômatos, diferentemente dos determinísticos (onde as transições são definidas por uma função), não é necessário que, para cada estado, se defina toda e qualquer transição possível envolvendo todas as palavras do alfabeto (conforme exemplos 1 e 2 da sub-seção 6.2.3).

Definição 6.6 A configuração (q´, w´) é derivável em uma etapa da configuração (q, w) por uma máquina M, ou seja, $\mathbf{(q,w) \vdash_M (q', w')}$ se:

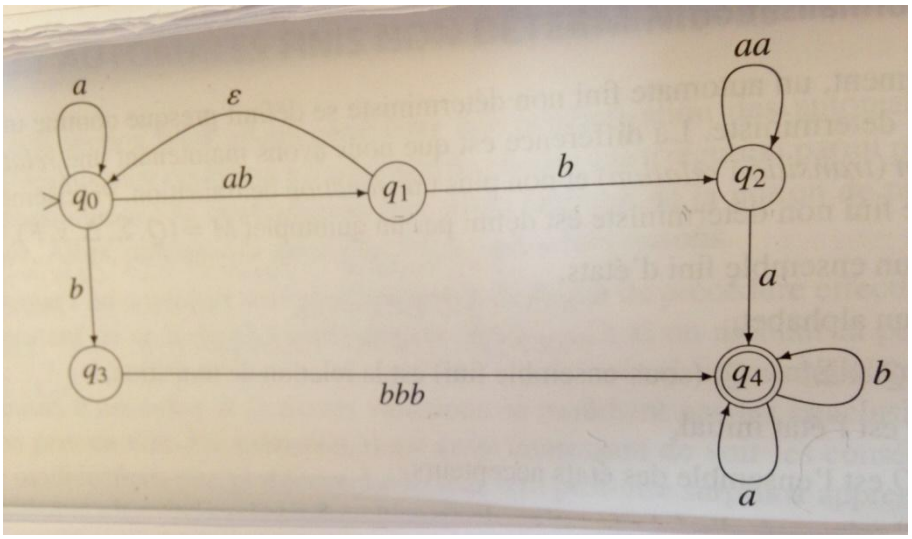
- w = uw´ (a palavra w começa por um prefixo $u \in \Sigma^*$);
- $(q, u, q') \in \Delta$ (a terna *(q, u, q´)* pertence à relação de transição)

Definição 6.7 Uma palavra *w* é aceita por um autômato finito não determinístico M se, e somente se, EXISTE PELO MENOS UMA derivação $\mathbf{(s,w) \vdash^*_M (q, \varepsilon)}$ em que $\mathbf{q \in F}$.

Saliente-se que a definição de execução de um autômato finito não determinístico sobre uma palavra *w* é semelhante à de um determinístico (definição 6.3), diferindo apenas pelo fato de, nele, a possibilidade de execução não ser única. Conseqüentemente, a definição de uma palavra w aceita pelos autômatos finitos não determinísticos difere da mesma definição relativa aos autômatos finitos determinísticos (definição 6.4) apenas pelo fato de exigir a existência de PELO MENOS UMA execução da palavra w que leve a um estado de aceitação (as demais execuções de w podem levar a estados que NÃO são de aceitação).

6.2.3 Exemplos

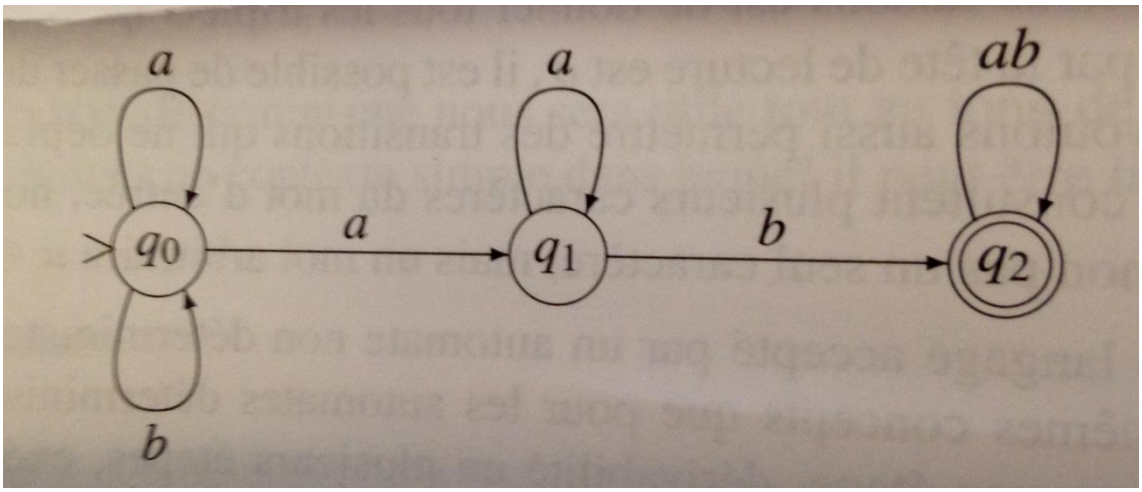
Exemplo 1: A linguagem aceita pelo autômato *M*:



é denotada pela seguinte expressão regular: *((a ∪ ab)*bbbbΣ*) ∪ ((a ∪ ab)*abb(aa)*aΣ*)* (que representa TODOS os caminhos do grafo que produzem uma palavra aceita pela linguagem).

Tal como ilustra este exemplo 1, em decorrência da definição dos autômatos finitos não determinísticos, uma transição pode ser etiquetada com a palavra vazia ε, pois $\varepsilon \in \Sigma^*$. Uma transição *(p, ε, q)* permite migrar do estado *p* para o estado *q* sem que a cabeça do autômato avance. Tal transição indica que o autômato, estando no estado *p*, pode passar para o estado *q* INDEPENDENTEMENTE da sub-palavra da fita que ainda falta ser processada.

Exemplo 2: a linguagem aceita pelo autômato M



é denotada pela seguinte expressão regular: *Σ*ab(ab)**, ou seja, o conjunto de palavras que terminam por pelo menos uma repetição de *ab*.

Este exemplo 2 ilustra a propriedade dos autômatos finitos não determinísticos de permitir que, em um mesmo estado, mais de uma transição seja etiquetada com um mesmo símbolo do alfabeto (é esse o fator que insere o não determinismo). Como ilustração desse não determinismo, pode-se citar o fato de que, no mesmo exemplo 2,

a execução da palavra ab (que é aceita por este autômato) pode parar no estado q_0 (que não é de aceitação) ou no estado q_2 (que é de aceitação). Tais exemplos ilustram, também, o fato de a relação de transição nos autômatos finitos não determinísticos não precisarem ser, necessariamente, totais. De fato, por exemplo, não há transições definidas para a palavra a nos estado q_3 e q_1 do exemplo 1.

6.3 Eliminação do Não-Determinismo

Conforme mostrado nesta seção, o não determinismo não acrescenta capacidade alguma aos autômatos finitos. Logo, qualquer autômato finito não determinístico pode ser substituído por um autômato finito determinístico que lhe seja equivalente.

Definição 6.8. Dois autômatos M_1 e M_2 são equivalentes se eles aceitam a mesma linguagem, ou seja, a linguagem aceita por ambos é donotada pela mesma expressão regular.

Teorema 6.1. Para todo autômato finito não determinístico, é possível construir um autômato finito determinístico que lhe seja equivalente (a veracidade deste teorema pode ser verificada de maneira bem intuitiva a partir da descrição do princípio de conversão de autômatos não determinísticos para determinísticos apresentada a seguir).

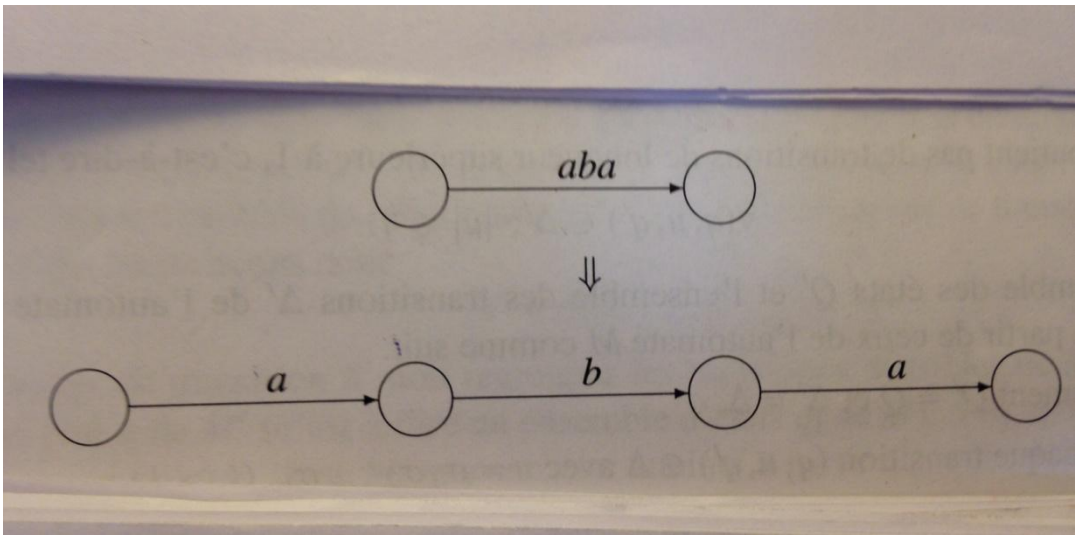
6.3.1 Princípio da Construção do Autômato Finito Determinístico Equivalente a um Dado Autômato Finito Não Determinístico

São as seguintes as duas diferenças entre os autômatos finitos determinísticos e não determinísticos:

- A presença de transições sobre palavras com tamanho 0 ou superior a 1;
- O fato de poder haver mais de uma transição, para um mesmo estado, etiquetada com uma mesma sub-palavra (transições NÃO EXCLUSIVAS).

Etapas a serem cumpridas para construir o autômato finito determinístico equivalente a um não determinístico:

- Eliminar as transições etiquetadas com palavras com tamanho superior a 1. Ex:



- A fim de eliminar as transições não exclusivas, o autômato finito determinístico deve levar em consideração todas as transições possíveis do não determinístico que lhe é equivalente, ou seja, deve-se construir um autômato determinístico que, a cada etapa, memorize todos os estados nos quais o autômato não determinístico que lhe é equivalente possa se encontrar. Logo, cada estado do determinístico corresponderá a um conjunto de estados do não determinístico. A figura 6.2 mostra um exemplo de como as transições não determinísticas etiquetadas com *a* em um autômato não determinístico são passíveis de serem representadas por meio de uma única transição etiquetada com *a* e de conjuntos de estados em um autômato determinístico.

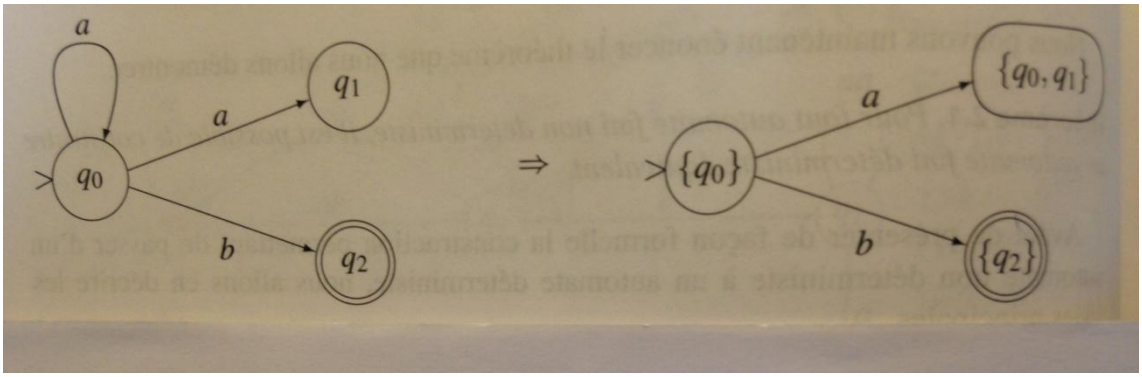


Figura 6.2. Conversão de transições não determinísticas em uma determinística

6.3.2 Formalização da Construção do Autômato Finito Determinístico Equivalente a um Dado Autômato Finito Não Determinístico

Eliminação das Transições Etiquetadas com sub-palavras de comprimento > 1:

Seja um autômato finito *não determinístico* M = {Q, Σ, Δ, s, F). Inicialmente, constrói-se um autômato *não determinístico* M´ = {Q´, Σ, Δ´, s, F), equivalente a M, que não contenha transições de comprimento superior a 1, ou seja:

$\forall(q, u, q') \in \Delta'$, $|u| \leq 1$. Para tanto, Q' e Δ' são obtidos conforme abaixo:

- Inicialmente, $Q' = Q$ e $\Delta' = \Delta$.
- Para cada transição $(q, u, q') \in \Delta$ com $u = \sigma_1\sigma_2 \ldots \sigma_k$, (k > 1):
  - Exclui-se tal transição de Δ';
  - Acrescentam-se novos estados $p_1, \ldots, p_{k-1}$ a Q';
  - Acrescentam-se as transições $(q, \sigma_1, p_1), (p_1, \sigma_2, p_2), \ldots, (p_{k-1}, \sigma_k, q')$ a Δ'.

Eliminiação do Não Determinismo (Transições não exclusivas):

Seja o autômato *não determinístico* $M = \{Q, \Sigma, \Delta, s, F\}$ em que toda transição $(q, u, q') \in \Delta$ tem $|u| \leq 1$, ou seja, é da forma (q, σ, q'), onde $\sigma \in \Sigma$, ou (q, ε, q'). O objetivo é construir um autômato *determinístico* $M' = \{Q', \Sigma, \delta', s', F'\}$, equivalente a M. Para isso, é necessário eliminar as transições sobre a palavra vazia (ε).

Definição 6.9. Para um estado q de um autômato M, E(q) é o conjunto dos estados que podem ser acessados a partir de q por uma seqüência de transições sobre ε:

$E(q) = \{p \in Q \mid (q, w) \vdash_M^* (p, w)\}$

Os elementos de M′ são os seguintes:

- O conjunto dos estados Q′ de M′ é o conjunto dos sub-conjuntos de estados de M, ou seja: $Q' = 2^Q$. Logo, cada estado do autômato determinístico corresponde a um conjunto de estados do autômato não determinístico.

- O estado inicial s′ de M′ deve representar os estados em que o autômato não determinístico M pode encontrar-se antes de começar a ler sua palavra de entrada. Logo: $s' = E(s)$.

- A função de transição δ' deve agrupar as transições possíveis de M. Logo, para um estado **q** de M′ (ou seja, um conjunto de estados q_i de M), $\delta'(\mathbf{q}, l)$ é o conjunto de estados que, em M, podem ser atingidos a partir de um dos estados $q_i \in \mathbf{q}$ por uma seqüência de transições em que a primeira é etiquetada pelo símbolo l de Σ e as seguintes por ε:

  $\delta'(\mathbf{q}, l) = \cup \{E(p) \mid \exists q \in \mathbf{q} : (q, l, p) \in \Delta\}$.

  Obviamente, deve-se definir o valor de $\delta'(\mathbf{q}, l)$ para todo estado **q** de M′ (começando pelo estado inicial, ou seja, pelo estado **q** correspondente a **s′**). Além disso, as definições das transições relativas a um dado estado **q** devem ser feitas sobre todos os símbolos l de Σ (pois a função de transição tem que ser TOTAL). Por exemplo, a obtenção das transições do autômato determinístico da última figura acima foi feita através dos seguintes cálculos:

  $\delta'(\mathbf{q}, l) = \delta'(\{q_0\}, a) = \{q_0, q_1\}$
  $\delta'(\mathbf{q}, l) = \delta'(\{q_0\}, b) = \{q_2\}$

- Um estado do autômato determinístico M´será de aceitação se ele contiver um estado aceitador do autômato não determinístico M. De fato, em situações em que o autômato determinístico se encontra em um estado **q**, o autômato não determinístico estaria em qualquer dos estados q $\in$ **q** :

  $F' = \{\mathbf{q} \in Q' \mid \mathbf{q} \cap F \neq \varnothing\}$

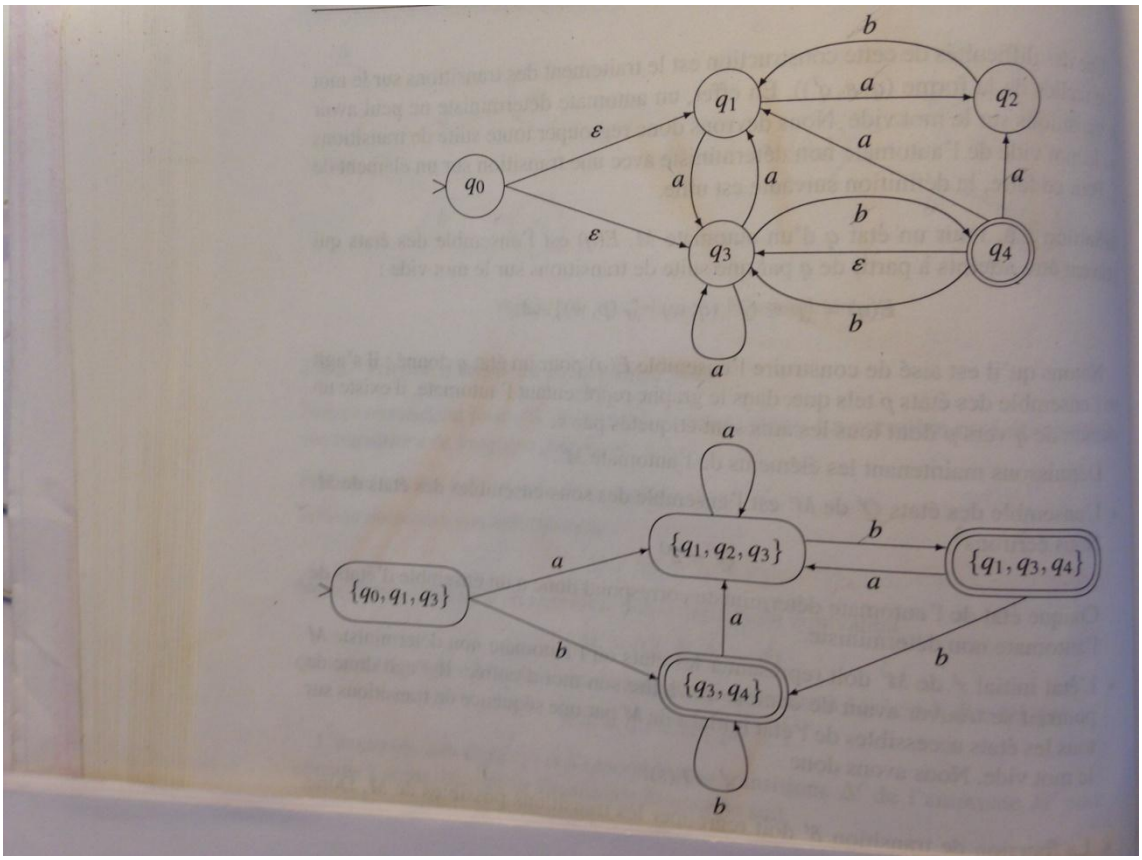
Definição 6.10. Um estado *p* é acessível a partir de um estado *q* em um autômato M se existe uma palavra *w* tal que:

$(q, w) \vdash_M^* (p, \varepsilon)$

Ou, ainda, um estado *p* é acessível a partir de um estado *q* em um autômato M se no grafo que representa M existir um caminho de *q* para *p*. Um estado de um autômato que não é acessível partir do estado inicial, não figurará jamais em uma execução deste autômato, podendo ser omitido sem modificar a linguagem aceita por ele. A fim de evitar a geração de estados inacessíveis, pode-se modificar a construção do autômato determinístico, de modo que Q´seja gerado conforme se segue:

1. Inicialmente, Q´ contém o estado inicial s´.
2. As seguintes operações se repetem até que elas não modifiquem mais o conjunto Q´:
   a) Escolhe-se um estado **q** $\in$ Q´ ao qual a operação b) ainda não foi aplicada;
   b) Para cada símbolo a $\in \Sigma$, calcula-se o estado **p** tal que **p** = δ'(**q**, a). O estado **p** é acrescentado a Q´.

Exemplo 1. A figura abaixo representa um autômato não determinístico e o autômato determinístico que lhe corresponde:



Observa-se que a figura não representa todos os 2^5 estados de M´, representando apenas os estados *acessíveis* a partir do estado inicial de M´.

A seguir mostra-se, passo a passo, o processo de obtenção de M´ a partir de M relativo à figura acima (gerando apenas os estados acessíveis de Q´):

$E(q_0) = \{ q_0 , q_1 , q_3 \}$;
$E(q_1) = \{ q_1 \}$;
$E(q_2) = \{ q_2 \}$;
$E(q_3) = \{ q_3 \}$;
$E(q_4) = \{ q_3 , q_4 \}$.

$s´ = E(s) = E(q_0) = \{ q_0 , q_1 , q_3 \}$.

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

A seguir serão calculadas as transições $\delta´(\mathbf{q} , l)$, a partir de cada estado **q** de M´, sobre cada símbolo *l* de Σ (pois a função de transição tem de ser TOTAL):

a) $\delta´(\mathbf{q} , l) = \delta´(s´ , a) = \delta´(\{ q_0 , q_1 , q_3 \}, a) = \cup \{E(p) \mid \exists q \in \{ q_0 , q_1 , q_3 \} : (q, a, p) \in \Delta \}$

No caso, os elementos $q \in \{ q_0 , q_1 , q_3 \}$ tal que $p : (q, a, p) \in \Delta$ (ou seja, os elementos do conjunto $\{ q_0 , q_1 , q_3 \}$ a partir dos quais parte uma transição etiquetada com o símbolo a no autômato original M), são os seguintes: q_1 e q_3 . Além disso, os destinos (ou seja, os valores de p) dessas transições em M são: q_2 e q_3 , relativos a q_1 , e q_3 e q_1 , relativos a q_3 , conforme tabela a seguir:

q	p	p
q_1	q_2	q_3
q_3	q_3	q_1

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

b) $\delta'(\mathbf{q} , l) = \delta'(s' , b) = \delta'(\mathbf{\{} q_0 , q_1 , q_3 \}, b) = \cup \{E(p) \mid \exists q \in \{ q_0 , q_1 , q_3 \} : (q, b, p) \in \Delta \}$

Analogamente, calculando agora a transição do estado $\{ q_0 , q_1 , q_3 \}$ etiquetada com o símbolo b:

q	p
q_3	q_4

Logo, $\delta'(\mathbf{\{} q_0 , q_1 , q_3 \}, b) = \cup \{ E(q_4) \} = \{ q_3 , q_4 \}$

c) $\delta'(\mathbf{q} , l) = \delta'(\mathbf{\{} q_1 , q_2 , q_3 \}, a) = \cup \{E(p) \mid \exists q \in \{ q_1 , q_2 , q_3 \} : (q, a, p) \in \Delta \}$

Analogamente, calculando agora a transição do estado $\{ q_1 , q_2 , q_3 \}$ etiquetada com o símbolo a:

q	p	p
q_1	q_2	q_3
q_3	q_3	q_1

Logo, $\delta'(\mathbf{\{} q_1 , q_2 , q_3 \}, a) = \cup \{ E(q_2), E(q_3) , E(q_1) \} = \{ q_1, q_2, q_3 \}$

d) $\delta'(\mathbf{q} , l) = \delta'(\mathbf{\{} q_1 , q_2 , q_3 \}, b) = \cup \{E(p) \mid \exists q \in \{ q_1 , q_2 , q_3 \} : (q, b, p) \in \Delta \}$

Analogamente, calculando agora a transição do estado $\{ q_1 , q_2 , q_3 \}$ etiquetada com o símbolo b:

q	p
q_2	q_1
q_3	q_4

Logo, $\delta'(\mathbf{\{} q_1 , q_2 , q_3 \}, b) = \cup \{ E(q_1), E(q_4) \} = \{ q_1, q_3 , q_4 \}$

e) $\delta'(\mathbf{q} , l) = \delta'(\{ q_1 , q_3 , q_4 \}, a) = \cup \{E(p) \mid \exists q \in \{ q_1 , q_3 , q_4 \} : (q, a, p) \in \Delta \}$

Analogamente, calculando agora a transição do estado $\{ q_1 , q_3 , q_4 \}$ etiquetada com o símbolo *a*:

q	p	p
q_1	q_2	q_3
q_3	q_3	q_1
q_4	q_2	q_1

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

f) $\delta'(\mathbf{q} , l) = \delta'(\{ q_1 , q_3 , q_4 \} , b) = \cup \{E(p) \mid \exists q \in \{ q_1 , q_3 , q_4 \} : (q, b, p) \in \Delta \}$

Analogamente, calculando agora a transição do estado $\{ q_1 , q_3 , q_4 \}$ etiquetada com o símbolo *b*:

q	p
q_3	q_4
q_4	q_3

Logo, $\delta'(\{ q_1 , q_3 , q_4 \} , b) = \cup \{ E(q_4), E(q_3) \} = \{ q_3 , q_4 \}$

g) $\delta'(\mathbf{q} , l) = \delta'(\{ q_3 , q_4 \} , a) = \cup \{E(p) \mid \exists q \in \{ q_3 , q_4 \} : (q, a, p) \in \Delta \}$

Analogamente, calculando agora a transição do estado $\{ q_3 , q_4 \}$ etiquetada com o símbolo *a*:

q	p	p
q_3	q_3	q_1
q_4	q_2	q_1

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

h) $\delta'(\mathbf{q} , l) = \delta'(\{ q_3 , q_4 \} , b) = \cup \{E(p) \mid \exists q \in \{ q_3 , q_4 \} : (q, b, p) \in \Delta \}$

Analogamente, calculando agora a transição do estado $\{ q_3 , q_4 \}$ etiquetada com o símbolo *b*:

q	p
q_3	q_4
q_4	q_3

Logo, $\delta'(\{ q_3 , q_4 \} , b) = \cup \{ E(q_4), E(q_3) \} = \{ q_3 , q_4 \}$

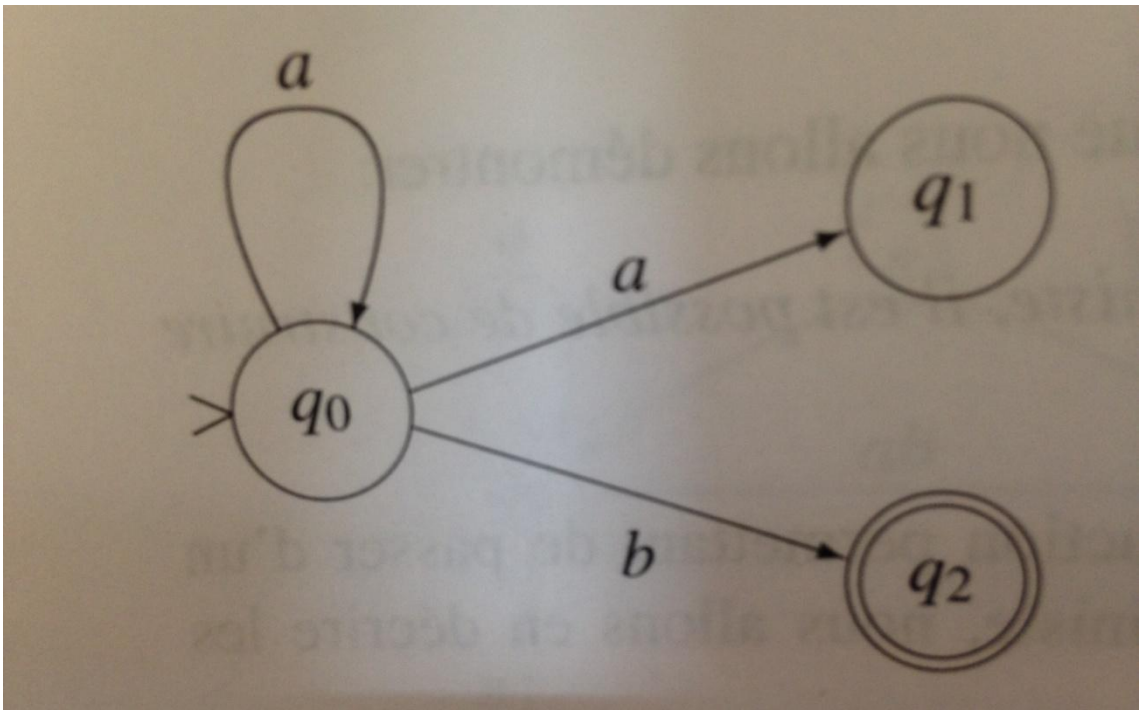
Os estados de aceitação do autômato finito são calculados abaixo

$F' = \{\mathbf{q} \in Q' \mid \mathbf{q} \cap F \neq \varnothing \} = \{ \{q_1, q_3 , q_4\}, \{q_3 , q_4\} \}$

A tabela abaixo resume todas as transições calculadas acima:

q	δ'(**q** , a)	δ'(**q** , b)
$\{q_0, q_1, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_3, q_4\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_3, q_4\}$
$\{q_3, q_4\}$	$\{q_1, q_2, q_3\}$	$\{q_3, q_4\}$
$\{q_1, q_3, q_4\}$	$\{q_1, q_2, q_3\}$	$\{q_3, q_4\}$

Exemplo 2. Gerando o autômato finito determinístico completo correspondente ao seguinte autômato não determinístico apresentado na Figura 6.2:



$E(q_0) = \{ q_0 \}$;
$E(q_1) = \{ q_1 \}$;
$E(q_2) = \{ q_2 \}$;

$s' = E(s) = E(q_0) = \{ q_0 \}$.

$\Delta = \{ (q_0 , a , q_0), (q_0 , a , q_1), (q_0 , b , q_2) \}$

A seguir serão calculadas as transições $\delta'(\mathbf{q} , l)$, a partir de cada estado **q** de M´, sobre cada símbolo *l* de Σ (pois a função de transição tem de ser TOTAL). A título de síntese, será apresentada apenas a tabela final das transições obtidas.

a) $\delta'(\mathbf{q} , l) = \delta'(s' , a) = \delta'(\{ q_0 \}, a) = \cup \{E(p) \mid \exists q \in \{ q_0 \} : (q, a, p) \in \Delta \} =$
$\cup \{ E(q_0), E(q_1) \} = \{ q_0, q_1 \}$

b) $\delta'(\mathbf{q} , l) = \delta'(s' , b) = \delta'(\{ q_0 \}, b) = \cup \{E(p) \mid \exists q \in \{ q_0 \} : (q, b, p) \in \Delta \} =$
$\cup \{ E(q_2) \} = \{ q_2 \}$

c) $\delta'(\mathbf{q} , l) = \delta'(\{ q_0, q_1 \} , a) = \cup \{E(p) \mid \exists q \in \{ q_0, q_1 \} : (q, a, p) \in \Delta \} =$
$\cup \{ E(q_0), E(q_1) \} = \{ q_0, q_1 \}$

d) $\delta'(\mathbf{q} , l) = \delta'(\{ q_0, q_1 \} , b) = = \cup \{E(p) \mid \exists q \in \{ q_0, q_1 \} : (q, b, p) \in \Delta \} =$
$\cup \{ E(q_2) \} = \{ q_2 \}$

e) $\delta'(\mathbf{q} , l) = \delta'(\{ q_2 \} , a) = = \cup \{E(p) \mid \exists q \in \{ q_2 \} : (q, a, p) \in \Delta \} =$
$\cup \{ \{ \} \} = \{ \}$

f) $\delta'(\mathbf{q}, l) = \delta'(\{ q_2 \}, b) = = \cup \{E(p) \mid \exists q \in \{ q_2 \} : (q, b, p) \in \Delta \} =$
$\cup \{ \{ \} \} = \{ \}$

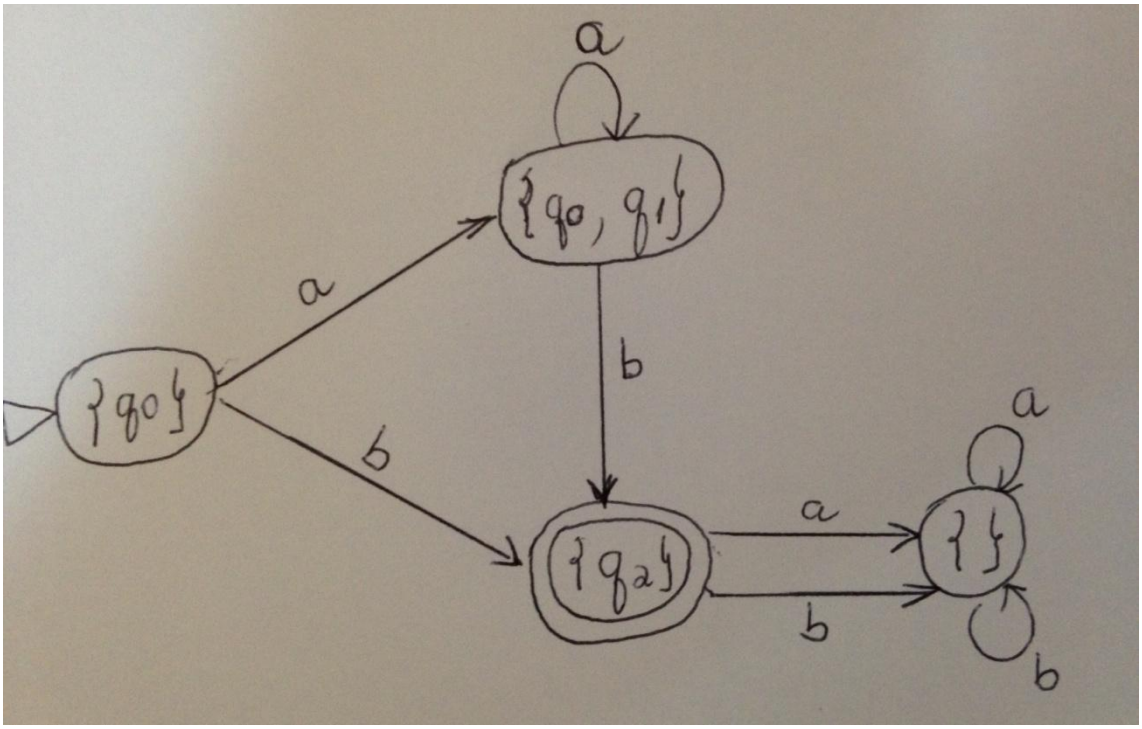
Logo, a tabela geral das transições é representada por:

q	$\delta'(\mathbf{q}, a)$	$\delta'(\mathbf{q}, b)$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_2\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_2\}$
$\{q_2\}$	$\{\}$	$\{\}$

Os estados de aceitação do autômato finito são calculados abaixo (no caso, ele é único):

$F' = \{\mathbf{q} \in Q' \mid \mathbf{q} \cap F \neq \varnothing \} = \{ \{q_2\} \}$

Assim sendo, o grafo que representa o autômato determinístico correspondente M´ é:



Saliente-se que, tanto M quanto M´, aceitam apenas as palavras sobre o alfabeto {a, b} que sejam uma seqüência de qualquer tamanho (inclusive zero) de símbolos *a* seguida de um, e somente um, símbolo *b* que encerra a palavra, conforme ilustrado nos exemplos de execução de palavras abaixo:

$(\{ q_0 \}, abb) \vdash_{M'} (\{ q_0 \}, bb) (\{ q_0, q_1 \}, bb) \vdash_{M'} (\{ q_2 \}, b) \vdash_{M'} (\{\}, \varepsilon)$,

onde o estado final obtido $\{ \} \notin F'$.

$(\{ q_0 \}, ba) \vdash_{M'} (\{ q_2 \}, a) \vdash_{M'} (\{ \}, \varepsilon)$,

onde o estado final obtido $\{ \} \notin F'$.

$(\{ q_0 \}, b) \vdash_{M'} (\{ q_2 \}, \varepsilon)$,

onde o estado final obtido $\{ q_2 \} \in F'$.

$(\{ q_0 \}, ab) \vdash_{M'} (\{ q_0 , q_1\}, b) \vdash_{M'} (\{ q_2 \}, b) \vdash_{M'} (\{ q_2 \}, \varepsilon)$,

onde o estado final obtido $\{ q_2 \} \in F'$.

6.4 Autômatos Finitos e Expressões Regulares

Teorema 6.2. Uma linguagem é regular se, e somente se, ela é aceita por um autômato finito.

A demonstração deste teorema será baseada no teorema 5.1 (uma linguagem é regular se, e somente se, é descrita por uma expressão regular). Além disso, o teorema 6.1 permite que a demonstração se limite aos autômatos finitos não determinísticos. Assim sendo, a demonstração do teorema 6.2 será feita por meio da demonstração dos dois lemas que se seguem.

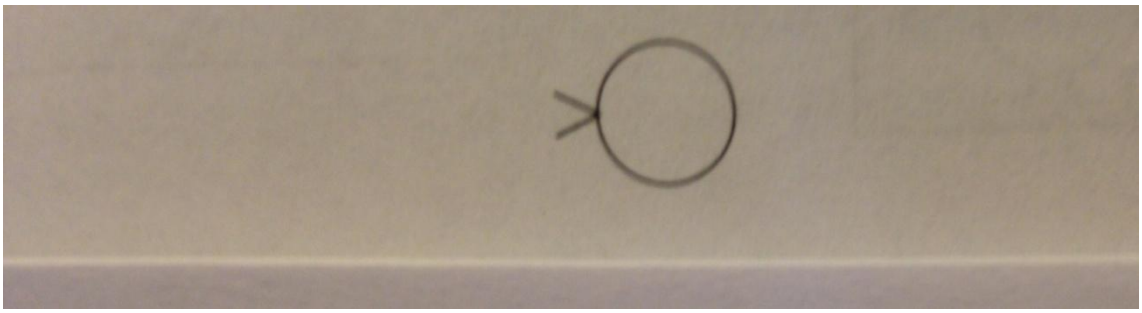
Lema 6.1. Se uma linguagem é denotada por uma expressão regular, ela é aceita por um autômato finito não determinístico.

Lema 6.2. Se uma linguagem é aceita por um autômato finito não determinístico, ela é regular.

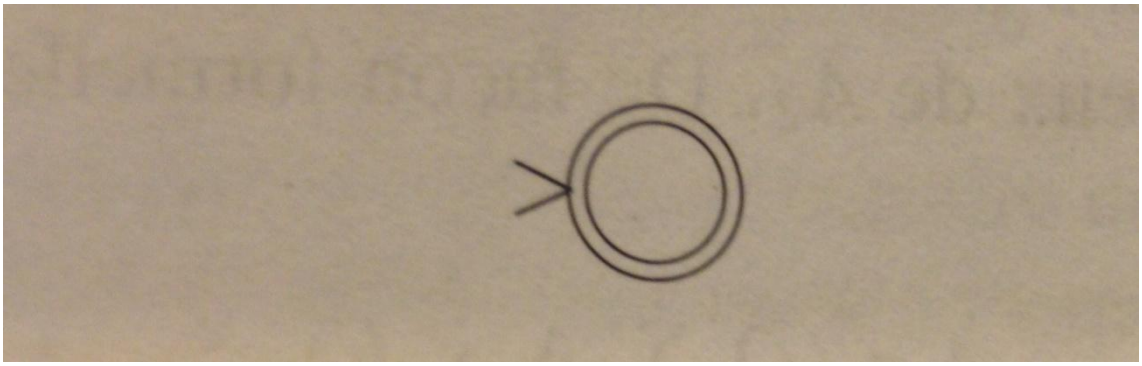
6.4.1 Das Expressões Regulares aos Autômatos

Suponha-se uma linguagem descrita por uma expressão regular. Mostra-se a seguir como construir um autômato finito não determinístico que aceite tal linguagem. Será mostrado que para cada expressão regular de base ($\varnothing$, ε, $a \in \Sigma$) existe um autômato que aceita tal linguagem e que para cada expressão regular composta ($(\alpha_1\alpha_2)$, $(\alpha_1 \cup \alpha_2)$, $(\alpha_1)^*$) é possível obter um autômato que aceita tal linguagem a partir dos autômatos que aceitam as linguagens descritas por α_1 e por α_2 .

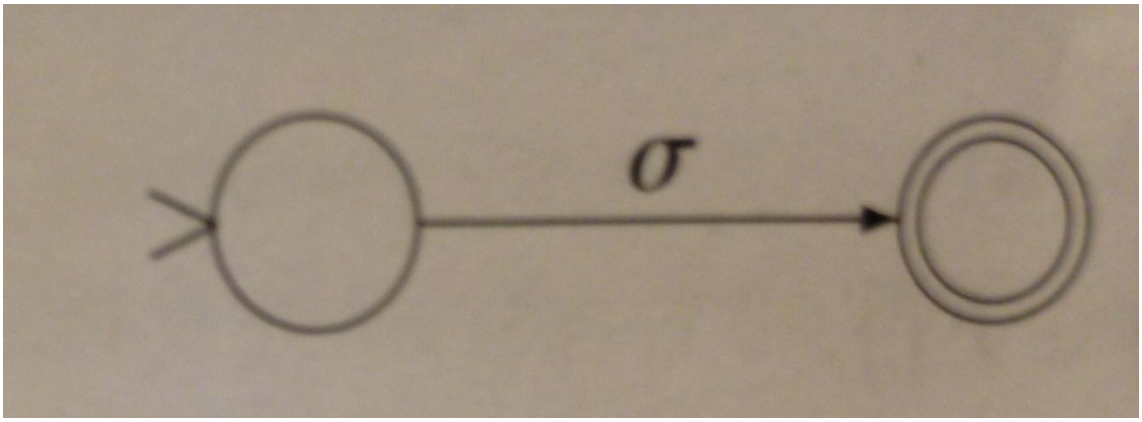
Para as expressões regulares de base: Para $\varnothing$, o autômato é:



Para ε o autômato é:

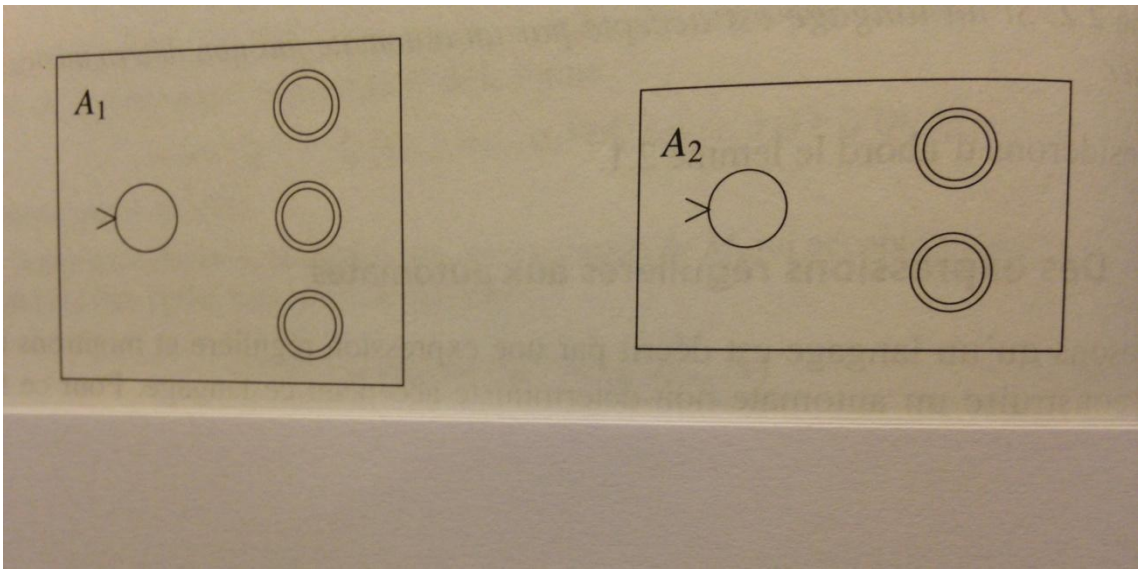


Finalmente, para $\sigma \in \Sigma$ o autômato é:

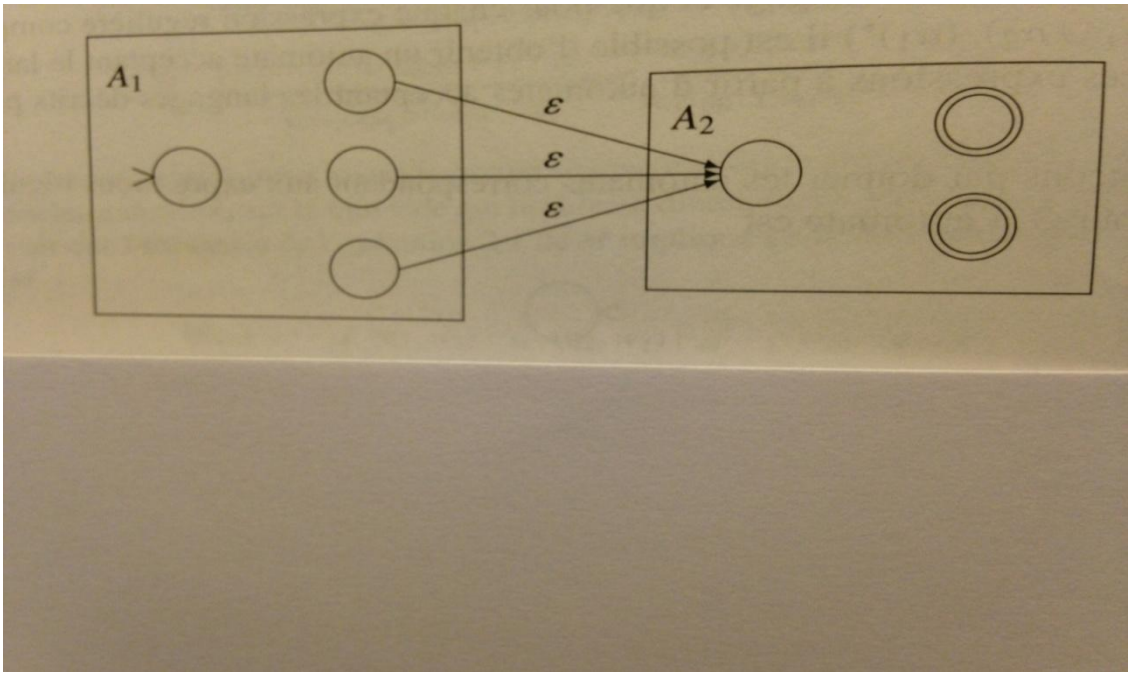


Suponha-se agora que já estejam construídos os seguintes autômatos correspondentes às expressões regulares α_1 e α_2, respectivamente:

$A_1 = (Q_1, \Sigma, \Delta_1, s_1, F_1)$ e $A_2 = (Q_2, \Sigma, \Delta_2, s_2, F_2)$, sendo Q_1 e Q_2 escolhidos estados disjuntos. Considerem-se os seguintes grafos representando tais autômatos (estando evidenciados apenas os estados iniciais e os estados de aceitação):



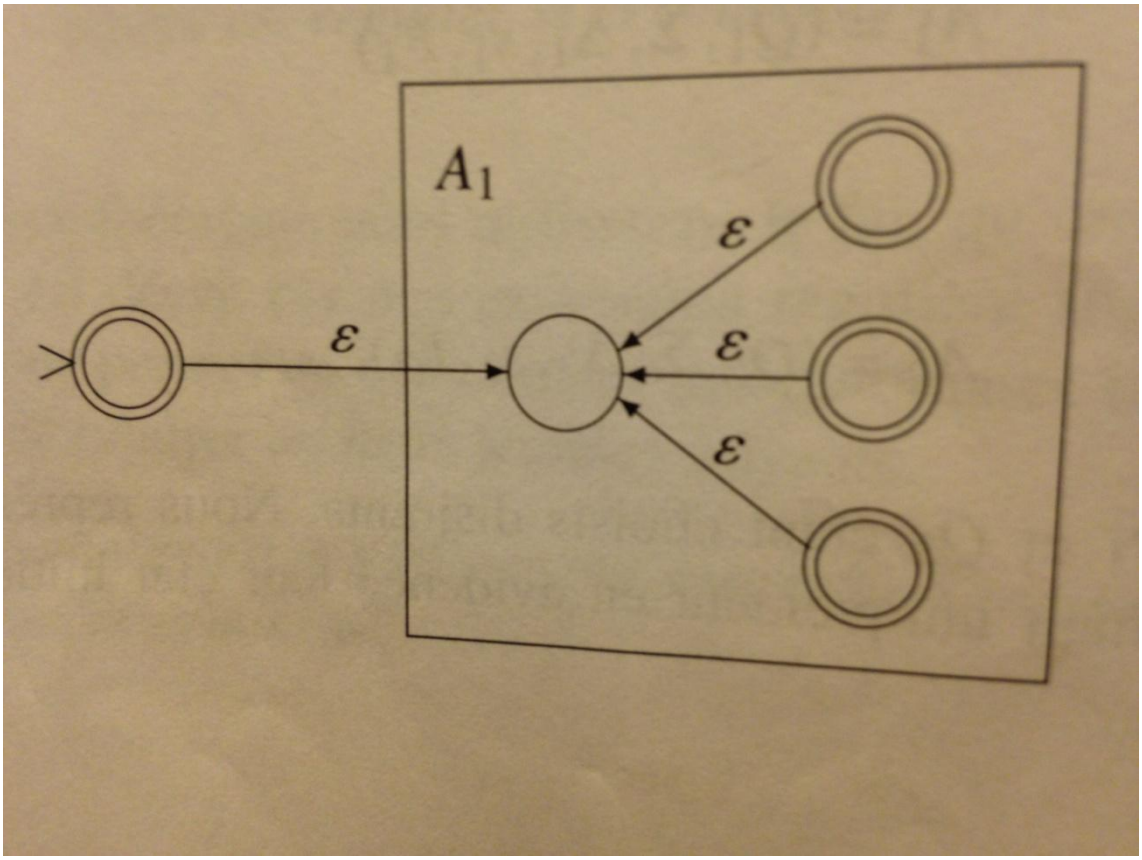
O autômato para a expressão regular ($\alpha_1\alpha_2$) é, então:



Tal autômato foi obtido conectando-se os estados de aceitação de A_1 ao estado inicial de A_2 por meio de transições ε. O estado inicial do novo autômato é aquele de A_1 e seus estados de aceitação são aqueles de A_2. Formalmente, construiu-se um autômato $A = (Q, \Sigma, \Delta, s, F)$ em que:

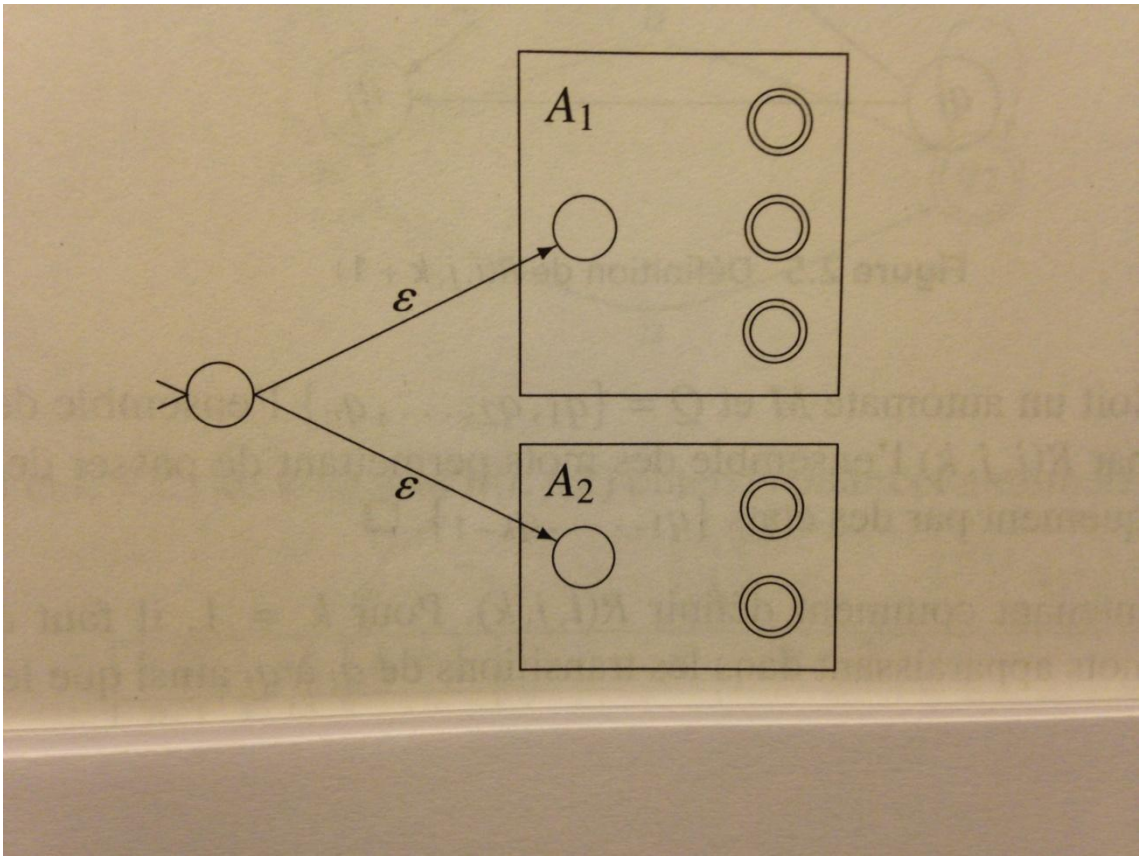
- $Q = Q_1 \cup Q_2$,
- $\Delta = \Delta_1 \cup \Delta_2 \cup \{ (q, \varepsilon, s_2) \mid q \in F_1 \}$,
- $s = s_1$,
- $F = F_2$.

Para o fechamento iterativo ($\alpha = \alpha_1^*$), constrói-se o seguinte autômato:



Saliente-se que foi inserido um novo estado para servir como estado inicial (isso é necessário pois $\varepsilon \in L(\alpha_1^*)$ e pode ocorrer de ε não pertencer a $L(\alpha_1)$.

Finalmente, para a operação de união ($\alpha = \alpha_1 \cup \alpha_2$), obtém-se o autômato:



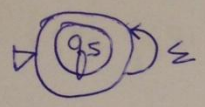
observe-se que a utilização de autômatos não determinísticos torna tal demonstração muito mais simples do que seria com a utilização dos determinísticos.

Exemplo de Uso dos Tópicos de Linguagens Regulares Estudados à Solução de Problema Prático:

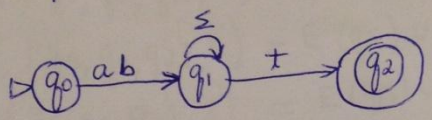
As quatro figuras abaixo ilustram o uso dos métodos estudados nas sub-seções 6.3.2 e 6.4.1 na solução do problema prático de localizar as ocorrências de determinados identificadores em uma cadeia de caracteres:

Busca em uma cadeia de caracteres: problema de encontrar os pontos de uma longa cadeia de caracteres w onde termina pelo menos uma instância de uma expressão regular α.
Exc: encontrar em um arquivo contendo texto w de um programa os pontos onde termina pelo menos uma ocorrência de identificadores cujas primeiras letras são "ab" e cuja última letra é "t".

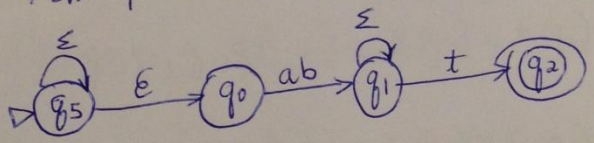
- NDA (aut. fin. não determinístico) para Σ^*, onde Σ é o conjunto de todos os caracteres ASCII:



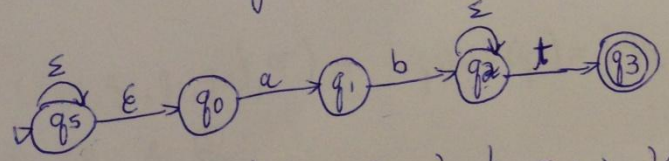
- NDA para α: $ab\Sigma^*t$



- NDA para β: $\Sigma^*\alpha$ (usando as regras da seção 6.4.1):



- convertendo o NDA para β em um DFA (aut. finito determinístico), usando as regras da seção 6.3.2):



$E(q_5) = \{q_5, q_0\}$; $E(q_0) = \{q_0\}$; $E(q_1) = \{q_1\}$; $E(q_2) = \{q_2\}$

$E(q_3) = \{q_3\}$

$s' = E(q_5) = \{q_5, q_0\}$

$\delta'(\{q_5, q_0\}, a) = E(q_5) \cup E(q_1) = \{q_5, q_0, q_1\}$

$\delta'(\{q_5, q_0\}, \sigma) = E(q_5) = \{q_5, q_0\}$, $\forall \sigma \in (\Sigma - \{a\})$

$\delta'(\{q_5, q_0, q_1\}, a) = E(q_5) \cup E(q_1) = \{q_5, q_0, q_1\}$

$\delta'(\{q_5, q_0, q_1\}, b) = E(q_5) \cup E(q_2) = \{q_5, q_0, q_2\}$

$\delta'(\{q_5, q_0, q_1\}, \sigma) = E(q_5) = \{q_5, q_0\}$, $\forall \sigma \in (\Sigma - \{a, b\})$

$\delta'(\{q_5, q_0, q_2\}, a) = E(q_5) \cup E(q_1) \cup E(q_2) = \{q_5, q_0, q_1, q_2\}$

$\delta'(\{q_5, q_0, q_2\}, t) = E(q_5) \cup E(q_2) \cup E(q_3) = \{q_5, q_0, q_2, q_3\}$

$\delta'(\{q_5, q_0, q_2\}, \sigma) = E(q_5) \cup E(q_2) = \{q_5, q_0, q_2\}$, $\forall \sigma \in (\Sigma - \{a, t\})$

$\delta'(\{q_5, q_0, q_1, q_2\}, a) = E(q_5) \cup E(q_1) \cup E(q_2) = \{q_5, q_0, q_1, q_2\}$

$\delta'(\{q_5, q_0, q_1, q_2\}, b) = E(q_5) \cup E(q_2) \cup E(q_2) = \{q_5, q_0, q_2\} \rightarrow ①$

$\delta'(\{q_5, q_0, q_1, q_2\}, t) = E(q_5) \cup E(q_2) \cup E(q_3) = \{q_5, q_0, q_2, q_3\}$

$\delta'(\{q_5, q_0, q_1, q_2\}, \sigma) = E(q_5) \cup E(q_2) = \{q_5, q_0, q_2\}$, $\forall \sigma \in (\Sigma - \{a, b, t\}) \rightarrow ②$

de ① e ②:

$\delta'(\{q_5, q_0, q_1, q_2\}, \sigma) = \{q_5, q_0, q_2\}$, $\forall \sigma \in (\Sigma - \{a, t\})$

$\delta'(\{q_5, q_0, q_2, q_3\}, a) = E(q_5) \cup E(q_1) \cup E(q_2) = \{q_5, q_0, q_1, q_2\}$

$\delta'(\{q_5, q_0, q_2, q_3\}, b) = E(q_5) \cup E(q_2) = \{q_5, q_0, q_2\}$ ①

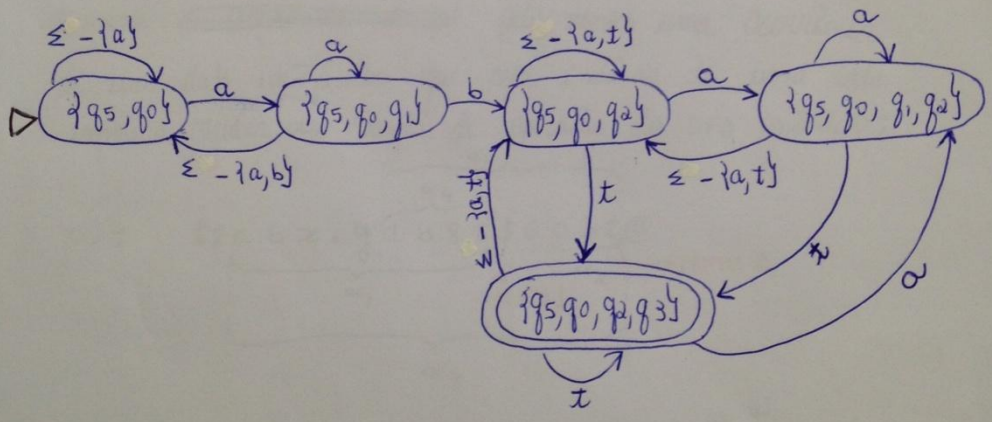
$\delta'(\{q_5, q_0, q_2, q_3\}, t) = E(q_5) \cup E(q_2) \cup E(q_3) = \{q_5, q_0, q_2, q_3\}$

$\delta'(\{q_5, q_0, q_2, q_3\}, \sigma) = E(q_5) \cup E(q_2) = \{q_5, q_0, q_2\}, \forall \sigma \in (\Sigma - \{a, b, t\})$ ②

de ① e ②

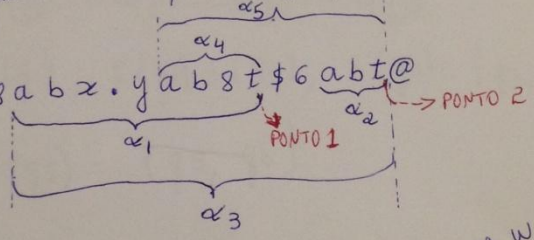
$\delta'(\{q_5, q_0, q_2, q_3\}, \sigma) = \{q_5, q_0, q_2\}, \forall \sigma \in (\Sigma - \{a, t\})$

DFA correspondente à expressão regular β:



Resposta Final do Exercício:

A fim de encontrar pontos em uma longa cadeia de caracteres w (pertencente à linguagem denotada por uma expressão regular β) onde termina pelo menos uma ocorrência de uma instância de uma expressão regular α, basta simular a execução do DFA correspondente a β para a palavra w. A cada vez que tal autômato se encontra em um estado de aceitação, isso sinaliza um desses pontos. Isso serve, por exemplo, para delimitar pontos de um programa w em que termina ~~a ocorrência de~~ pelo menos uma ocorrência de um dado identificador que seja instância de uma dada expressão regular α. Exemplo de aplicação do DFA construído:



Note que a simulação do DFA de β para w passa duas vezes por um estado de aceitação: PONTO 1 e PONTO 2. Logo, em cada um desses pontos ocorreu pelo menos uma nova instância de α. Tal método pode ser usado como algoritmo para detectar TODAS as ocorrências de instâncias de α em w (no caso do exemplo, 5 ocorrências). Nesse algoritmo (de complexidade $O(|w|)$), para cada sequência "ab" encontrada, cada "t" que a sucede delimita uma nova ocorrência de uma instância de α.

6.4.2 Dos Autômatos às Linguagens Regulares

Seja um autômato M. Para se construir uma expressão regular que descreva L(M), poder-se-ia pensar na seqüência dos seguintes passos:

(1) Selecionam-se os caminhos entre o estado inicial de M e os estados de aceitação de M.
(2) Obtém-se a expressão regular correspondente a cada um desses caminhos, concatenando-se as etiquetas de suas transições. Os laços (iterações) são tratados pela inserção do operador * .
(3) A expressão regular desejada é a união das expressões regulares assim obtidas.

Tal solução é, essencialmente, a que será adotada. Contudo, ela comporta um elemento impreciso - a existência dos laços – que exigirá uma abordagem mais rigorosa para produzir a expressão regular procurada. A solução para a obtenção dessa expressão é indutiva. Suponha-se que o conjunto de estados de M é Q = $\{q_1, q_2, ..., q_n\}$, sendo q_1 o estado inicial. Para obter o conjunto de palavras que permitem passar de um estado q_i a um estado q_j, serão consideradas inicialmente as palavras que permitem passar de q_i a q_j sem nenhum estado intermediário; depois, passando somente por estados pertencentes ao conjunto $\{q_1\}$; depois, passando somente por estados pertencentes ao conjunto $\{ q_1 , q_2 \}$; depois, passando somente por estados pertencentes ao conjunto $\{q_1 ,q_2 ,q_3\}$, depois, ... ; finalmente, passando somente por estados pertencentes ao conjunto $\{ q_1 , q_2 , ..., q_n \}$.

Definição 6.11. Seja um autômato M e Q = $\{q_1, q_2, ..., q_n\}$ o conjunto de seus estados, sendo q_1 **o estado inicial**. Designa-se R(i, j, k) ao conjunto de palavras que permitem passar do estado q_i ao estado q_j diretamente ou passando unicamente por estados pertencentes ao conjunto $\{ q_1, ... , q_{k-1} \}$.

Mostra-se a seguir, como gerar os valores pertinentes de R(i, j, k).

Para k = 1, consideram-se apenas as palavras que aparecem nas transições de q_i a q_j, bem como a palavra vazia (esta última para o caso em que i = j). Tem-se assim:

$R(i, j, 1) = \{w \mid (q_i, w, q_j) \in \Delta\}$, se $i \neq j$;
$R(i, j, 1) = \{ \varepsilon \} \cup \{w \mid (q_i, w, q_j) \in \Delta\}$, se $i = j$.

Para k > 1, a definição será indutiva. As palavras que permitem passar de q_i a q_j utilizando apenas os estados pertencentes ao conjunto $\{q_1, q_2, ..., q_k\}$ pertencem ao conjunto formado pela união dos seguintes conjuntos:

- Conjunto das palavras que permitem efetuar a transição de q_i a q_j passando apenas por estados pertencentes a $\{q_1, q_2, ..., q_{k-1}\}$;
- Conjunto das palavras que permitem, passando apenas por estados pertencentes a $\{q_1, q_2, ..., q_{k-1}\}$, efetuar as seguintes transições: de q_i a q_k , em seguida, de q_k a q_k um certo número de vezes e, finalmente, de q_k a q_j , (ver Figura 6.3 abaixo, que é a definição de R(i, j, k + 1)).

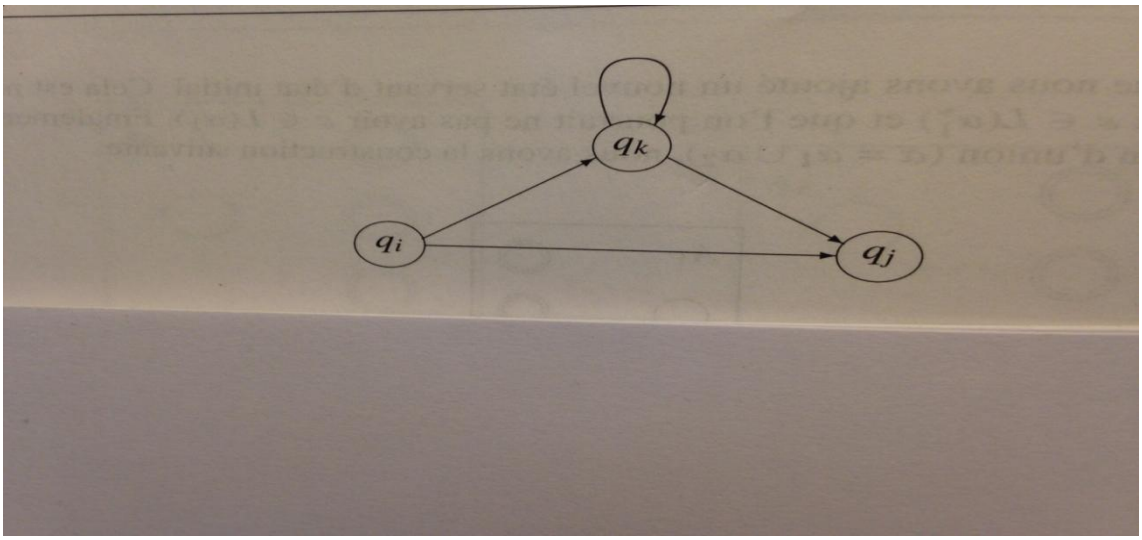


Figura 6.3. Definição de R(i, j, k + 1)

Tem-se, assim:

$R(i, j, k+1) = R(i, j, k) \cup$
$R(i, k, k)\, R(k, k, k)^{*}\, R(k, j, k)$

Finalmente, a linguagem aceita pelo autômato se exprime como sendo a união de todos os conjuntos de palavras que permitem passar do estado inicial a um de seus estados de aceitação passando por estados que o compõem. Supondo que o estado inicial é q1 e que o autômato comporta n estados, têm-se assim:

$L(M) = \cup R(1, j , n+1)$, onde o estado destino $q_j \in F$ (ou seja, é um estado de aceitação).

Resumindo, para calcular L(M), onde M é um autômato com n estados, para cada par ordenado possível de estados q_i e q_j de M, deve-se efetuar a seguinte seqüência de passos:

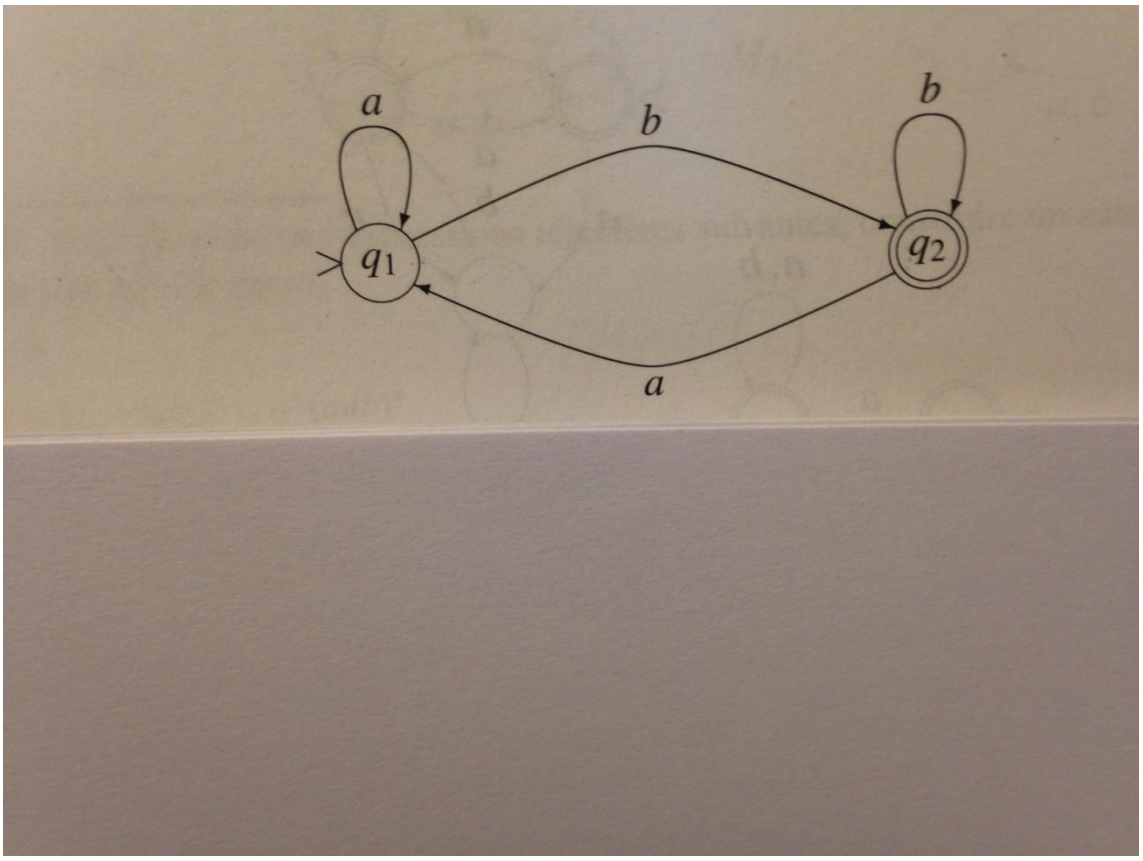
- Calcula-se a linguagem R(i, j, 1) , ou seja, o conjunto de palavras que, executadas por M, levam o autômato do estado q_i ao estado q_j diretamente, ou seja, sem passar por estado intermediário algum.
- A partir daqui, partindo-se do valor de R(i, j, 1), calculam-se os sucessivos R(i, j, K +1), até que se conclua o cálculo de R(1, j , n + 1). Tal sucessão poder ser resumida como: calcula-se a linguagem R(i, j, 2), ou seja, o conjunto de palavras que, executadas por M, levam o autômato do estado q_i ao estado q_j , diretamente, ou passando somente por estados pertencentes ao conjunto { q_1 } (isto é, pelo próprio q_1. Logo, no cálculo de R(i, j, 2), q_1 corresponde ao novo estado q_k inserido na análise feita na Figura 6.3)); Calcula-se a linguagem R(i, j, 3), ou seja, o conjunto de palavras que, executadas por M, levam o autômato do estado q_i ao estado q_j , diretamente, ou passando somente por estados pertencentes a { q_1 } (o que corresponde ao valor R(i, j, 2) calculado anteriormente), ou passando somente por estados pertencentes a { q_1 , q_2 } . Logo, no cálculo de R(i, j, 3), q_2 corresponde ao novo estado q_k inserido na

análise feita na Figura 6.3)); ... ; finalmente, de modo análogo, calcula-se R(i, j, n + 1).

- Concluídos tais cálculos, eles serão utilizados como parâmetros para que se determine L(M) por meio da equação apresentada acima.

Saliente-se que, para definir as linguagens R(i, j, k), utilizaram-se apenas conjuntos finitos de palavras e as operações de união, concatenação e de fechamento iterativo. Tais linguagens são regulares, fato que estabelece que L(M) também é uma linguagem regular.

Exemplo: Seja o autômato abaixo:



O conjunto de estados do autômato M acima é definido por Q = { q_1 , q_2 }, sendo q_1 seu estado inicial. Assim sendo, a fim de calcular a linguagem L(M) aceita por ele, inicialmente devem-se calcular TODAS as linguagens R(i, j, k) associadas a tal autômato, onde:

- $1 \leq k \leq 2$, pois M contém dois estados.
- $1 \leq i \leq 2$, pois os índices dos estados de M variam entre 1 e 2;
- $1 \leq j \leq 2$, pois os índices dos estados de M variam entre 1 e 2;

Assim sendo, para k = 1 e k = 2, as linguagens R(i, j, k) obtidas para tal autômato são obtidas a partir das seguintes seqüências de cálculo (começando pelo cálculo da linguagem R(1,1,1) e, a partir daí, gerando as demais linguagens R(i, j, k + 1)):

Seqüência 1: (onde i = j = 1):

$R(1, 1, 1) = \varepsilon \cup a$; de fato, a partir de q_1 , somente os processamentos das palavras ε e a levam ao estado q_1 sem que se passe por estado intermediário algum.

R(1, 1, 2) : calculada com base na equação de R(i, j, k+1), ou seja,

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

onde, neste caso, i = j = k = 1:

$$R(1, 1, 2) = R(1, 1, 1) \cup R(1, 1, 1)\, R(1, 1, 1)^* \, R(1, 1, 1) =$$
$$(\varepsilon \cup a) \cup (\varepsilon \cup a)\, (\varepsilon \cup a)^* (\varepsilon \cup a)$$

De fato, a partir de q_1 , somente os processamentos das palavras que sejam seqüências finitas e de tamanho variável (inclusive zero) de símbolos a permitem ir para o estado q_1 passando unicamente, por estados que pertençam ao conjunto $\{ q_1 \}$.

Seqüência 2: (onde i = 1; j = 2):

$R(1, 2, 1) = b$; de fato, a partir de q_1 , somente o processamento da palavra b leva ao estado q_2 sem que se passe por estado intermediário algum.

R(1, 2, 2) : calculada com base na equação de R(i, j, k+1), ou seja,

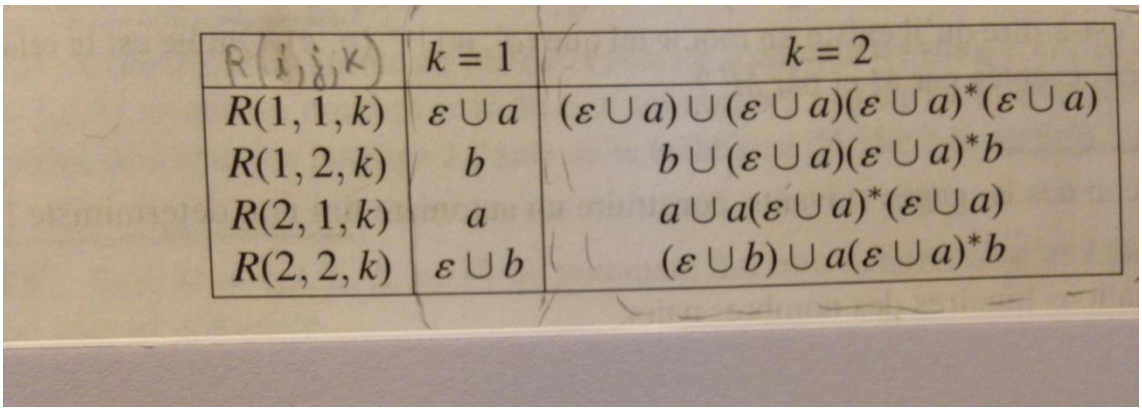
$R(i, j, k+1) = R(i, j, k) \cup R(i, k, k)\, R(k, k, k)^* \, R(k, j, k)$,

onde, neste caso, i = 1; j = 2 ; k = 1:

$$R(1, 2, 2) = R(1, 2, 1) \cup R(1, 1, 1)\, R(1, 1, 1)^* \, R(1, 2, 1) =$$
$$b \cup (\varepsilon \cup a)\, (\varepsilon \cup a)^* b$$

De fato, a partir de q_1 , somente os processamentos das palavras que sejam uma seqüência finita e de tamanho variável (inclusive zero) de símbolos a seguida de um único símbolo b permitem ir para o estado q_2 , diretamente, ou passando, unicamente, por estados pertencentes a $\{ q_1 \}$.

As duas seqüências remanescentes podem ser calculadas de modo análogo (fica como exercício descrevê-las detalhadamente). A tabela abaixo resume o cálculo de todas essas quatro seqüências (cada linha da tabela representa uma dessas seqüências):



$R(i,j,k)$	$k=1$	$k=2$
$R(1,1,k)$	$\varepsilon \cup a$	$(\varepsilon \cup a) \cup (\varepsilon \cup a)(\varepsilon \cup a)^*(\varepsilon \cup a)$
$R(1,2,k)$	b	$b \cup (\varepsilon \cup a)(\varepsilon \cup a)^* b$
$R(2,1,k)$	a	$a \cup a(\varepsilon \cup a)^*(\varepsilon \cup a)$
$R(2,2,k)$	$\varepsilon \cup b$	$(\varepsilon \cup b) \cup a(\varepsilon \cup a)^* b$

Note que: os elementos da coluna do meio representam as linguagens R(i, j, 1) cujas palavras levam do estado q_i ao estado q_j diretamente (sem passar por estados intermediários); os elementos da terceira coluna representam a linguagem R(i, j, 2) cujas palavras levam do estado q_i ao estado q_j diretamente ou passando unicamente por estados pertencentes a $\{ q_1 \}$.

Logo, a linguagem aceita pelo autômato é R(1, 2, 3), cujo cálculo também é obtido a partir da equação de R(i, j, k+1), usando como parâmetros os valores da tabela acima é:

$$[b \cup (\varepsilon \cup a)\,(\varepsilon \cup a)^*\, b] \cup [b \cup (\varepsilon \cup a)\,(\varepsilon \cup a)^*\, b]\,[\,(\varepsilon \cup b) \cup a(\varepsilon \cup a)^*\, b]^* \,[\,(\varepsilon \cup b) \cup a(\varepsilon \cup a)^*\, b]$$

É importante relembrar que R(1, 2, 3) é a linguagem cujas palavras levam do estado q_1 ao estado q_2 diretamente, ou passando somente por estados pertencentes a $\{ q_1 , q_2 \}$.

É relevante observar também que, de fato, a expressão regular correspondente a R(1, 2, 3) descreve a linguagem formada por qualquer palavra em {a, b} que termina em *b*, a qual é equivalente à linguagem aceita pelo autômato apresentado.

7. As Gramáticas Regulares

7.1 Introdução

Esta seção tira o foco da busca de uma formalização para procedimento efetivo, concentrando-se na noção de linguagem como um conjunto de palavras que satisfazem um conjunto de regras (este último, denominado *gramática*) e não como uma ferramenta para a formalização da noção de problema. Usando como primeiro exemplo a linguagem natural, as seguintes regras integram a gramática do Português:

- Uma frase tem a forma de um sujeito seguido de um verbo;
- Um sujeito é um pronome;
- Um pronome é “ele” ou “ela”;
- Um verbo é “dorme” ou “escuta”.

Tal conjunto de regras permite construir as seguintes frases (que correspondem ao termo “palavras” usado nas definições formais de linguagens) pertencentes à linguagem do Português:

(1) ele escuta
(2) ele dorme
(3) ela dorme
(4) ela escuta

Logo, as gramáticas dão uma descrição *gerativa* de uma linguagem, ou seja, indicam como construir os elementos (palavras) que pertencem a tal linguagem. Por outro lado, os autômatos dão uma descrição *analítica* da linguagem, uma vez que provêem um procedimento para reconhecer seus elementos (palavras). Ambas as descrições são complementares e úteis. Por exemplo, no caso de uma linguagem de programação, utiliza-se uma descrição generativa ao se escrever um programa e uma descrição analítica ao se fazer uma análise sintática do programa escrito (durante a compilação). Nesse contexto, a forma de Backus Naur (BNF) freqüentemente usada para descrever as linguagens de programação é uma gramática. É muito importante a correspondência entre descrições gerativas e analíticas. Assim sendo, nesta seção, por exemplo, é apresentada uma descrição gerativa baseada em gramática para as linguagens regulares (que é uma alternativa à descrição gerativa baseada em expressões regulares) e, na seqüência, será mostrado como tal descrição gerativa pode ser convertida em uma descrição analítica (por meio de autômatos finitos) e vice-versa.

Finalmente, mesmo que a teoria das linguagens apresentada no presente curso não seja adequada para prover uma descrição completa da linguagem natural, ela é perfeitamente adaptada e largamente utilizada para a descrição e a análise das linguagens de programação.

7.2 As gramáticas

7.2.1 Definição

As regras de uma gramática são denominadas Regras de Re-escrita, uma vez que elas permitem que uma dada seqüência de símbolos pode ser substituída por uma outra. As palavras geradas por uma gramática são aquelas que podem ser obtidas aplicando-se suas regras a partir de um símbolo de partida.

Definição 7.1. Uma gramática é uma quádrupla G = (V, Σ, R, S), onde:

- V é um alfabeto (conjunto finito de símbolos);
- $\Sigma \subseteq$ V é um conjunto de símbolos *terminais*. V - Σ é o conjunto de símbolos *não terminais*. Os símbolos terminais fazem parte das palavras que compõem a linguagem definida pela gramática G. Por outro lado, os símbolos não terminais não aparecem nas palavras geradas, sendo utilizados, apenas, ao longo do processo de geração das mesmas.
- R $\subseteq$ (V^+ X V^*) é um conjunto finito de regras também chamadas de *produções*. Logo, R é um mapeamento do tipo R: $V^+ \rightarrow V^*$, onde V^+ é composto por todas as palavras em V, exceto a palavra vazia, e V^* é o conjunto de todas as palavras em V. Cada par ordenado (α, β) de R corresponde a uma regra do tipo $\alpha \rightarrow \beta$ cujo significado intuitivo é: o primeiro elemento α do par (onde $\alpha \in V^+$) pode ser substituído pelo segundo elemento β (onde $\beta \in V^*$). Quando se deseja indicar que uma dada regra (α, β) integra uma gramática G, escreve-se $\alpha \rightarrow_G \beta$;
- S $\in$ V - Σ designa o símbolo de partida a partir do qual começará a geração das palavras da linguagem por meio das regras da gramática;

Notações a serem usadas na descrição de uma gramática:

- Os elementos de V - Σ são representados por letras maiúsculas A, B, ...;
- Os elementos de Σ são representados por letras minúsculas a, b, ...;
- Como nas expressões regulares, a palavra vazia é denotada por ε.

Exemplo 7.1. Definindo uma gramática G:

- V = {S, A, B, a, b };
- Σ = {a, b};
- R = {S $\rightarrow$ A, S $\rightarrow$ B, B $\rightarrow$ bB, A $\rightarrow$ aA, A $\rightarrow \varepsilon$, B $\rightarrow \varepsilon$ };
- S é o símbolo de partida.

Definição 7.2. Seja uma gramática G = (V, Σ, R, S) e u $\in V^+$, v $\in V^*$. A gramática G permite derivar a palavra *v* a partir da palavra *u* em uma única etapa (passo), com notação $u =>_G v$, se e somente se:

- u = xu´y (*u* pode ser decomposta em três partes *x, u´* e *y*; as partes x e y podem eventualmente ser a palavra vazia);
- v = xv´y (*v* pode ser decomposta em três partes *x*, *v´*e *y*);
- u´$\rightarrow_G$ v´ (ou seja, a regra (u´, v´) pertence a R).

OBS: a partir daqui, sempre que se quiser indicar que *v* pode ser derivada a partir de *u* em um único passo por meio da regra $u' \rightarrow_G v'$, será usada a seguinte notação:

$$u \overset{u' \rightarrow v'}{=\!=\!=\!=\!=\!=\!=\!=>_G} v$$

EXEMPLOS DESTA DEFINIÇÃO 7.2. SERÃO VISTOS EM SALA DE AULA!

Definição 7.3. Seja uma gramática G = (V, Σ, R, S) e $u \in V^+$, $v \in V^*$. A gramática G permite derivar a palavra *v* a partir da palavra *u* em diversas etapa (passos), com notação $u \Rightarrow_G^* v$, se e somente se, $\exists k \geq 0$ tal que $v_0 \ldots v_{k-1} \in V^+$ e $v_k \in V^*$, onde:

- $u = v_0$;
- $v = v_k$;
- $v_i \Rightarrow_G v_{i+1}$, para $0 \leq i < k$.

Logo, uma palavra é derivada de outra em diversas etapas se ela pode ser obtida a partir desta última por uma sucessão de derivações em uma única etapa.

EXEMPLOS DESTA DEFINIÇÃO 7.3. SERÃO VISTOS EM SALA DE AULA!

Definição 7.4. As palavras geradas por uma gramática G são as palavras $v \in \Sigma^*$ (e, portanto, compostas unicamente por símbolos terminais) que podem ser derivadas a partir do símbolo de partida, ou seja:

$S \Rightarrow_G^* v$.

Exemplo: a palavra *aaaa* é gerada a partir da gramática G do exemplo 7.1, pois:

$$S \overset{S \rightarrow A}{=\!=\!=\!=>_G} A \overset{A \rightarrow aA}{=\!=\!=\!=\!=>_G} aA \overset{A \rightarrow aA}{=\!=\!=\!=\!=>_G} aaA \overset{A \rightarrow aA}{=\!=\!=\!=\!=>_G} aaaA \overset{A \rightarrow aA}{=\!=\!=\!=\!=>_G} aaaaA \overset{A \rightarrow \varepsilon}{=\!=\!=\!=>_G} aaaa$$

Definição 7.5. A linguagem gerada por uma gramática G (denominada L(G)) é o conjunto de palavras que podem ser geradas por G:

$L(G) = \{v \in \Sigma^* \mid S \Rightarrow_G^* v\}$.

Exemplo: A linguagem gerada pela gramática do exemplo 7.1. é aquela formada pela palavra vazia e pelas palavras formadas exclusivamente por *a* ou exclusivamente por *b*.

3.2.2 Tipos de Gramáticas

A classificação das gramáticas se baseia nas restrições que definem a forma de suas regras. Serão considerados neste curso quatro tipos de gramáticas:

Tipo 0 : não há restrições sobre as regras;

Tipo 1: chamadas gramáticas sensíveis ao contexto. As regras $\alpha \rightarrow \beta$ que as compõem devem satisfazer a restrição : $| \alpha | \leq | \beta |$, ou seja, o membro da direita deve conter pelo menos a mesma quantidade de símbolos do membro à esquerda. A fim de permitir que tais gramáticas gerem a palavra vazia, permite-se uma única exceção à restrição precedente: a regra $S \rightarrow \varepsilon$ pode fazer parte da gramática, sendo que, para tanto, S não pode aparecer no membro direito de alguma regra da gramática.

Tipo 2: chamadas gramáticas livres de contexto. Nestas gramáticas todas as regras têm que ter a forma A $\rightarrow \beta$, onde $A \in V - \Sigma$ e onde não há restrições sobre β. Logo, uma gramática é livre de contexto se o membro esquerdo de cada uma de suas regras é constituído de um único símbolo não terminal.

Tipo 3: chamadas gramáticas regulares. Uma gramática é regular se todas as regras têm uma das seguintes formas:

$A \rightarrow wB$
$A \rightarrow w$

Onde $A, B \in V - \Sigma$ e $w \in \Sigma^*$. Toda regra de uma gramática regular tem então o membro esquerdo constituído de um único símbolo não terminal e o membro direito constituído de uma palavra composta por símbolos terminais seguidos, eventualmente, de um único símbolo não terminal.

As gramáticas regulares, tais como acima definidas, são também denominadas *gramáticas lineares à direita*. Há outra alternativa para representar as gramáticas regulares (por meio das chamadas *gramáticas lineares à esquerda*), em que as suas regras têm uma das seguintes formas ($A, B \in V - \Sigma$ e $w \in \Sigma^*$) :

$A \rightarrow Bw$
$A \rightarrow w$

Prova-se que toda linguagem passível de ser definida por uma gramática linear à direita pode também ser definida por uma gramática linear à esquerda, e vice-versa.

Os tipos de gramática apresentados podem ser inter-relacionados. Adianta-se o seguinte fato (considerando-se $\subset$ como sendo a *inclusão própria*) :

Tipo 3 $\subset$ Tipo 2 $\subset$ Tipo 1 $\subset$ Tipo 0.

Logo, para i > j, há linguagens que podem ser geradas por uma gramática do tipo *j* , mas não por uma gramática do tipo *i* .

Nota-se que todas essas relações de inclusões podem ser facilmente verificadas, com exceção da seguinte: Tipo 2 $\subset$ Tipo 1, uma vez que há o seguinte problema: uma gramática do tipo 2 pode conter regras do tipo $A \rightarrow \varepsilon$ (onde *A* é um símbolo não terminal $\neq S$) para as quais a restrição $| \alpha | \leq | \beta |$ das gramáticas do tipo 1 não

é satisfeita (para todos os demais tipos de regras de uma gramática livre de contexto tal restrição é atendida). Contudo, tal problema pode ser resolvido partindo-se do fato de que tais tipos de regras podem ser eliminados das gramáticas do tipo 2 sem alterar as linguagens geradas por elas. Antes de apresentar um procedimento para eliminá-las, considere-se a seguinte definição relativa a uma gramática livre de contexto G:

Definição 7.6. Sendo *Y* um símbolo de $V - \Sigma$ (ou seja, não terminal, incluindo S):

- *Y* é um *símbolo anulável de G* se a regra $Y \rightarrow \varepsilon$, contendo a palavra vazia como corpo, pertence a *G*;
- *Y* é um *símbolo anulável de G* se existe uma regra $Y \rightarrow C_1C_2...C_k$, onde cada C_i do corpo da regra é um *símbolo anulável de G*.

Procedimento para Eliminar as Regras do tipo $A \rightarrow \varepsilon$ de uma Gramática Livre de contexto $G = (V, \Sigma, R, S)$ - logo, *A* representa um símbolo não terminal:

(1) Detectam-se todos os símbolos *anuláveis de G*;
(2) Constrói-se uma nova gramática $G_1 = (V, \Sigma, R_1, S)$ cujo conjunto de regras R_1 é definido da seguinte maneira:

Para cada regra $A \rightarrow X_1 X_2 ... X_k$ de *R*, onde $k \geq 1$, assumindo que *m* represente a quantidade dos símbolos anuláveis de *G* dentre os *k* símbolos X_i do corpo da regra, a nova gramática G_1 terá 2^m versões dessa regra, sendo que os símbolos X_i´s anuláveis de *G*, em todas as 2^m combinações possíveis, são caracterizados pelo fato de estarem "presentes" ou "ausentes". Há uma única exceção: se $m = k$, isto é, se todos os símbolos X_i´s são anuláveis de G, então não se inclui a regra na qual todos os X_i´s estão ausentes. Além disso, caso a regra $A \rightarrow \varepsilon$ pertença a *G*, ela não será inserida em R_1.

Logo:

- caso $\varepsilon \notin L(G)$, $L(G1) = L(G)$.
- caso $\varepsilon \in L(G)$, $L(G1) = L(G) - \{\varepsilon\}$. Neste caso, a gramática livre de contexto G_2 tal que $L(G2) = L(G)$ e que contém como regra do tipo $A \rightarrow \varepsilon$ unicamente a regra $S \rightarrow \varepsilon$ (onde *S* é o símbolo de partida) pode ser obtida da seguinte maneira : $G_2 = (V, \Sigma, R_2, S)$, onde $R_2 = R_1 \cup \{ S \rightarrow \varepsilon \}$.

Exemplo:

Calcule as gramáticas G1 e G2 relativas à gramática $G = (V, \Sigma, R, S)$, cujo conjunto R de regras é composto pelas 5 regras apresentadas abaixo: (as barras verticais que eventualmente aparecem em algumas regras separam as alternativas de corpo da regra correspondentes ao símbolo não terminal da esquerda):

R = { S → AB,
A → aAA| ε ,
B → bBB | ε }

(1) Símbolos Anuláveis de G: { A, B, S }
(2) Analisando as regras $A \rightarrow X_1 X_2 ... X_k$ do conjunto *R* de G onde $k \geq 1$, ou seja:

a) S → AB : ambos os símbolos do corpo da regra são anuláveis. Logo, há quatro combinações independentes possíveis de "presente" ou "ausente" para tais símbolos, devendo ser descartada aquela em que ambos estão ausentes. Conseqüentemente, tal regra provoca a inserção das seguintes 3 regras em G_1 :

S → AB | A | B

b) A → aAA : apenas os símbolos que ocupam a segunda e a terceira posição do corpo da regra contêm símbolos anuláveis. Logo, há quatro combinações independentes possíveis de "presente" ou "ausente", devendo ser todas consideradas. Conseqüentemente, tal regra provoca a inserção das seguintes 4 regras em G_1:

A → aAA | aA | aA | a

Como as duas regras intermediárias são idênticas, uma delas pode ser eliminada, permanecendo apenas três novas regras.

c) B → bBB : de modo análogo, tal regra provoca a inserção das seguintes 3 regras em G_1 :

B → bBB | bB | b

Conseqüentemente, o conjunto R_1 das 9 regras da gramática G_1 é o seguinte:

R_1 = { S → AB | A | B,
A → aAA | aA | a,
B → bBB | bB | b },

onde: $L(G1) = L(G) - \{\varepsilon\}$. Neste caso, o conjunto de regras R_2 da gramática livre de contexto $G_2 = (V, \Sigma, R_2, S)$ tal que $L(G2) = L(G)$ e que contém como regra do tipo $A \rightarrow \varepsilon$ unicamente a regra $S \rightarrow \varepsilon$ é o seguinte: onde $R_2 = R_1 \cup \{ S \rightarrow \varepsilon \}$.

Outra questão natural é a origem das denominações *linguagem livre de contexto* e *linguagem sensível ao contexto*. Uma regra da forma $A \rightarrow \beta$ é dita *livre de contexto* por permitir a substituição de *A* por β independentemente do contexto no qual A aparece (ou seja, independentemente do que antecede ou sucede *A*). Por outro lado, a regra $aAb \rightarrow a\beta b$ é *sensível ao contexto* por permitir a substituição de *A* por β somente em situações em que *A* for antecedido por *a* e sucedido por *b*, ou seja, a aplicação de tal regra é condicionada ao contexto no qual *A* aparece.

7.3 As Gramáticas Regulares

As gramáticas do tipo 3 são chamadas de Regulares pois são geradoras das linguagens regulares.

Teorema 7.1. Uma linguagem é regular se, e somente se, é gerada por uma gramática regular.

Demonstração do Teorema:

- ***Se uma linguagem é regular, ela é gerada por uma gramática regular.***

Seja uma linguagem regular L_1 definida pela seguinte autômato finito não determinístico: M = (Q , Σ , Δ , s , F), ou seja, L(M)= L_1. A meta, no caso, é construir uma gramática regular G = (V_G , Σ_G , S_G , R_G), tal que L(G) = L_1. Os elementos de G, definidos em função dos elementos de M, são:

- $\Sigma_G = \Sigma$ (os símbolos terminais de G correspondem aos do alfabeto de M);
- $V_G = Q \cup \Sigma$ (há um símbolo não terminal em G para cada estado de M);
- S_G = s (o símbolo de partida de G corresponde ao estado inicial de M);

- R_G : A $\rightarrow$ wB, para todo (A, w, B) $\in \Delta$;
  R_G : A $\rightarrow \varepsilon$, para todo A $\in$ F

  Logo, em G há uma regra para cada transição de Δ e uma regra que leva à palavra vazia para cada estado de aceitação de M.

É necessário mostrar também que uma palavra é aceita por M se, e somente se, ela é gerada por G. Tal demonstração fica bastante evidente quando se faz um paralelo entre as execuções de M e as derivações de G. Formalmente, tal demonstração pode ser feita por indução sobre o comprimento das derivações e das execuções.

- ***Se uma linguagem é gerada por uma gramática regular, ela é regular.***

Tal demonstração será feita por meio da prova de que uma linguagem L gerada por uma gramática regular é aceita por um autômato finito não determinístico. Seja G = (V_G , Σ_G , S_G , R_G) a gramática que gera L. Um autômato finito não determinístico M = (Q , Σ , Δ , s , F) que aceita L pode ser definido da seguinte forma:

- Q = V_G - Σ_G $\cup$ {f} (os estados de M são os não terminais de G mais um novo estado f);
- $\Sigma = \Sigma_G$;
- s = S_G ;
- F = { f };
- Δ = *(A, w, B)*, para todo $A \rightarrow wB \in R_G$;
  Δ = *(A, w, f)*, para todo $A \rightarrow w \in R_G$.

Observa-se que essa correspondência entre relação de transição de um autômato e uma regra de gramática evidencia o caráter "procedimento efetivo" de um autômato (objetivando a checar se a palavra da fita pertence ou não a uma dada linguagem) e o caráter "gerativo" de uma gramática (objetivando a gerar uma palavra de uma dada linguagem). De fato, enquanto que a tripla *(A,w,B)* indica uma transição em que, a partir do estado *A* do autômato, consumindo a palavra *w* da fita, passa-se ao estado *B* do autômato, a regra $A \rightarrow wB$ da gramática G indica que, a partir do símbolo não terminal *A* de *G*, pode ser gerada uma palavra composta pela palavra *w* seguida de um não terminal *B* (que, por sua vez, através de outra regra do tipo $B \rightarrow w'$, permitirá a geração de outra palavra *w´* a ser justaposta a w, gerando a palavra ww´ , e assim por diante...).

Logo, a linguagem gerada por G é idêntica à linguagem aceita por M. Uma prova mais formal pode ser feita por indução, analogamente à citada no caso precedente.

7.4 As Linguagens Regulares

Foram apresentadas até aqui quatro diferentes caracterizações das linguagens regulares:

(1) As expressões regulares;
(2) Os autômatos finitos determinísticos;
(3) Os autômatos finitos não determinísticos;
(4) As gramáticas regulares.

É bastante útil dispor de diversas caracterizações das linguagens regulares a fim de contar com alternativas mais cômodas que facilitem os seguintes processos: provar que uma dada linguagem é regular ou demonstrar certas propriedades das linguagens regulares. Seguem algumas evidências que corroboram tal argumento.

7.4.1 Propriedades das Linguagens Regulares

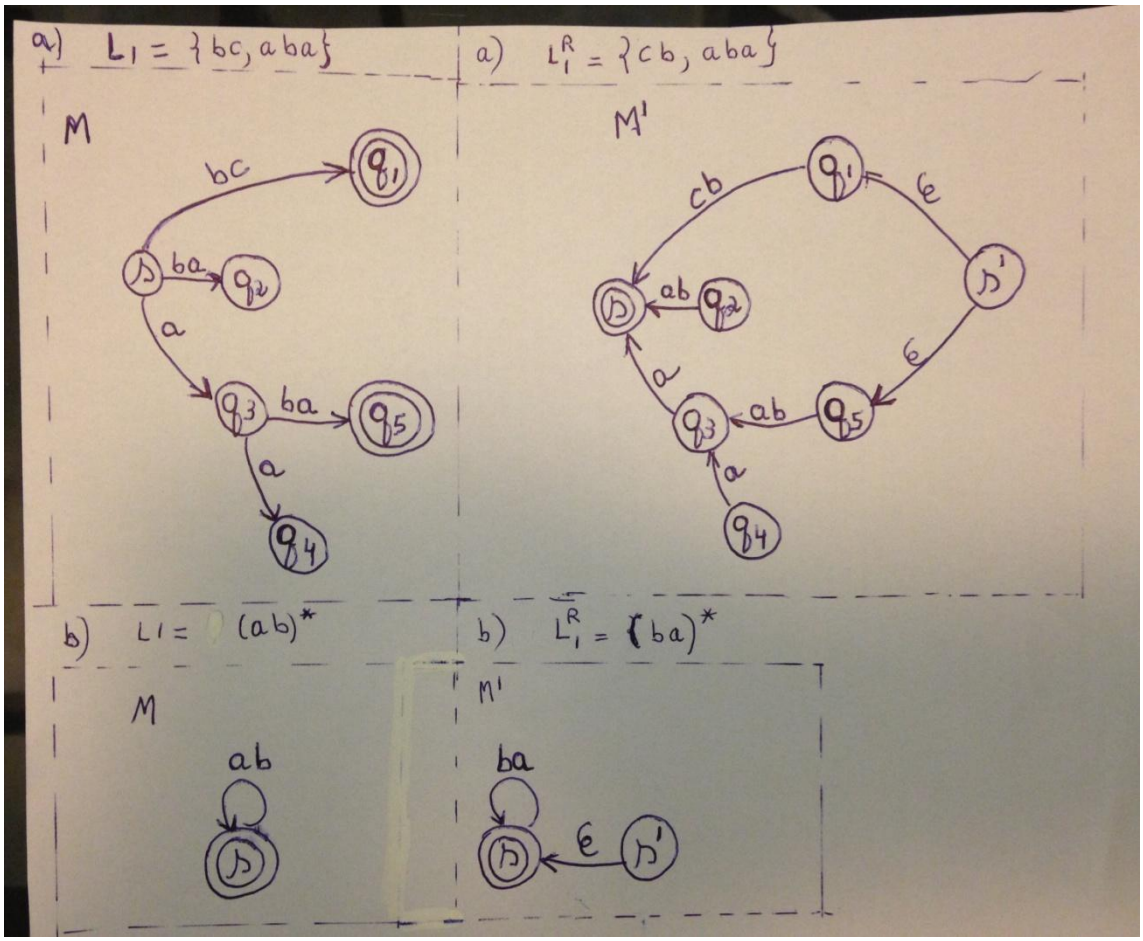
Sejam duas linguagens regulares L_1 e L_2.

- A linguagem $L_1 \cup L_2$ é regular. De fato, sendo α_1 e α_2 expressões regulares denotando, respectivamente, L_1 e L_2, a expressão regular $\alpha_1 \cup \alpha_2$ denota a linguagem $L_1 \cup L_2$, que é, assim, regular (teorema 5.1).

- Analogamente, demonstra-se que as linguagens $L_1 . L_2$ e L_1^* são regulares.

- Seja w^R o inverso de uma palavra *w*, ou seja, a palavra *w* escrita da direita para a esquerda. Assim sendo, linguagem $L_1^R = \{w \mid w^R \in L1\}$ é regular. De fato, a partir de um autômato $M = (Q , \Sigma , \Delta , s , F)$ que aceita L_1 pode-se construir um autômato $M' = (Q , \Sigma , \Delta' , s' , F')$ que aceita L_1^R . Isso se faz invertendo-se as transições de M, ou seja:

  - $F' = \{s\}$;
  - O estado inicial é um novo estado s';
  - $\Delta' = \{ (q, w^R , p) \mid (p, w, q) \in \Delta \} \cup \{s', \varepsilon , q) \mid q \in F\}$ (as transições são as de M invertidas mais uma transição do novo estado inicial em direção a cada estado de aceitação de M).



- Seja Σ o alfabeto de definição de *L1* e $\pi : \Sigma \rightarrow \Sigma'$ uma função de Σ em um outro alfabeto Σ' chamada *função de projeção*. Tal função pode ser estendida às palavras desde que ela seja aplicada a cada um de seus símbolos, isto é: para $w = w_1 \ldots w_k \in \Sigma^*$, $\pi(w) = \pi(w_1) \ldots \pi(w_k)$. A projeção de uma linguagem é, então, o conjunto de projeções de suas palavras e, para L_1 regular, a linguagem $\pi(L_1)$ é regular. De fato, a partir de um autômato que aceite L_1 , obtém-se um autômato que aceita $\pi(L_1)$ aplicando-se a função de projeção aos símbolos que aparecem sobre as transições. Observe que tal construção em geral não preserva o caráter determinista de um autômato, pois símbolos diferentes podem ter a mesma

imagem pela operação de projeção (a função de projeção não precisa ser, necessariamente, injetora).

- A linguagem:

  $\overline{L_1} = \Sigma^* - L_1$, denominada complemento de L1, é regular. De fato, pode-se construir um autômato finito determinístico $M' = (Q, \Sigma, \delta, s, F')$ que aceite $\overline{L_1}$ a partir de um autômato determinístico $M = (Q, \Sigma, \delta, s, F)$ que aceite L_1 . Isso é feito executando-se uma permutação entre os estados de aceitação e de não aceitação de M , ou seja, $F' = Q - F$. Observe-se que a mesma construção não pode ser feita com um autômato não determinístico, uma vez que tal tipo de autômato aceita uma palavra se existe uma execução dela que leve a um estado de aceitação. Isso implica que pode haver execuções de palavras aceitas por tal autômato que levam a estados de não aceitação. Em um autômato determinístico não há tal contratempo, visto que, nele, a execução correspondente a uma palavra é única.

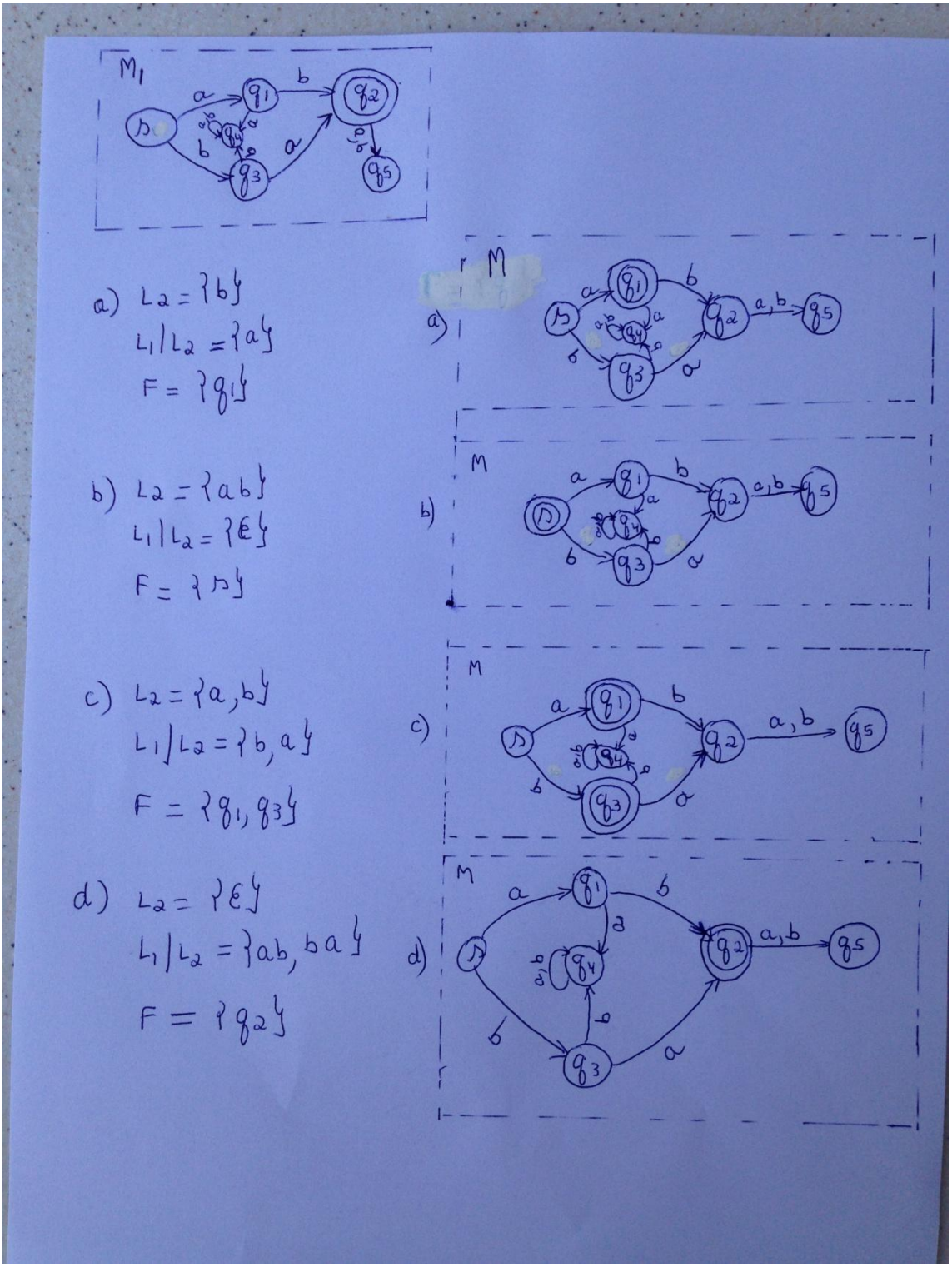
- A linguagem $L_1 \cap L_2$ (contendo todas as palavras que pertençam a L_1 e a L_2) é regular, pois é possível construir um autômato determinístico $M = (Q, \Sigma, \delta, s, F)$ que aceita $L_1 \cap L_2$ a partir dos seguintes autômatos determinísticos:
  $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ e $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$, os quais aceitam, respectivamente, L_1 e L_2 . Tal autômato M simula a execução simultânea dos dois autômatos M_1 e M_2 , aceitando apenas as palavras aceitas por ambos. Logo, os estados Q de M serão pares de estados compostos por um estado q_1 de M_1 e outro estado q_2 de M_2 . Resumindo:

  - $Q = Q_1 X Q_2$;
  - $\delta((q_1, q_2), \sigma) = (p_1, p_2)$ se, e somente se, $\delta_1(q_1, \sigma) = p_1$ e $\delta_2(q_2, \sigma) = p_2$;
  - $s = (s_1, s_2)$;
  - $F = F_1 X F_2$.

- A linguagem $L_1 / L_2 = \{ x \mid \exists y \in L_2$ tal que $xy \in L_1 \}$. Tal linguagem é denominada *quociente de L_1 por L_2* . Ela representa o conjunto de prefixos (podendo ser inclusive ε) das palavras de L_1 cuja concatenação com alguma palavra de L_2 produz uma palavra de L_1. L_1 / L_2 é uma linguagem regular. De fato, se o autômato que aceita L_1 é $M_1 = (Q, \Sigma, \delta, s, F_1)$, então L_1 / L_2 é aceita pelo autômato $M = (Q, \Sigma, \delta, s, F)$, onde:

  $F = \{q \in Q \mid L(Q, \Sigma, \delta, q, F_1) \cap L_2 \neq \varnothing\}$. O autômato M é idêntico ao M_1 exceto pelos estados de aceitação. Particularmente, os estados de aceitação de M correspondem aos estados de M_1 a partir dos quais é possível de se atingir um estado de aceitação do próprio M_1 executando-se uma palavra de L_2.

Exemplo: Dada a linguagem regular L_1 = {ab, ba}, representada pelo autômato $M_1 = (Q, \Sigma, \delta, s, F_1)$ ilustrado a seguir, calcule a linguagem quociente L_1 / L_2 para cada linguagem L_2 apresentada nos itens abaixo, mostrando, também, os respectivos autômatos $M = (Q, \Sigma, \delta, s, F)$ correspondentes às linguagens L_1 / L_2 calculadas:



7.4.2 Algoritmos

A maioria dos problemas envolvendo as linguagens regulares é solucionada por procedimentos efetivos. Exemplos importantes:

- Sejam *L* uma linguagem regular e *w* uma palavra $\in \Sigma^*$. O problema de detectar se $w \in L$ é solucionado pelo seguinte procedimento efetivo: construindo um autômato determinístico a partir da descrição utilizada para definir *L* (autômato, expressão regular ou gramática regular) e observando a execução deste autômato para a palavra *w*.

- Problema: detectar se uma linguagem regular L é vazia. Procedimento efetivo para resolvê-lo: constrói-se um autômato (determinístico ou não determinístico) que aceite L . L é então "não vazio" se existe um caminho entre o estado inicial e um estado de aceitação qualquer (obviamente, esta solução é uma alternativa geral apropriada para o caso em que, a priori, não se sabe se L é ou não é vazia. Quando se sabe que L é vazia desde o início, basta construir um autômato contendo um único estado de não aceitação).

- Problema: determinar se uma linguagem regular L é universal, ou seja, $L = \Sigma^*$.

  Procedimento efetivo correspondente: teste se $\overline{L} = \varnothing$. Para tanto, construa um autômato que aceite o complemento de L e cheque se ele não aceita palavra alguma.

- Problema: sendo L_1 e L_2 linguagens regulares, detectar se $L_1 \subseteq L_2$. Procedimento

  efetivo: cheque se $L_1 \cap \overline{L_2} = \varnothing$.

- Problema: sendo L_1 e L_2 linguagens regulares, detectar se $L_1 = L_2$. Procedimento efetivo: teste se $L_1 \subseteq L_2$ e $L_2 \subseteq L_1$.

  Fatos que tornam as linguagens regulares notáveis:

(1) Elas têm múltiplas caracterizações simples mas bastante distintas, como: as expressões regulares, os autômatos finitos determinísticos ou não, as gramáticas regulares (há outros algébricos ou ligados a certas Lógicas);
(2) As operações habituais pertinentes preservam o caráter regular das linguagens envolvidas: a união a intersecção e o complemento de linguagens regulares são regulares. O mesmo vale para as operações menos habituais, como o fechamento iterativo e a concatenação;
(3) Numerosos problemas relacionados às linguagens regulares são passíveis de serem resolvidos por um algoritmo. Logo, sempre que um problema pode ser enquadrado no domínio das linguagens regulares, em geral ele pode ser resolvido por um algoritmo;

(4) As linguagens regulares são úteis para resolver problemas práticos em computação, tal como o exemplo prático mostrado no final da seção 6.4.1.

7.5 Além das Linguagens Regulares

O objetivo desta subseção é apresentar fundamentos que comprovam a existência de linguagens que não são regulares e que ajudam a definir um limiar entre as linguagens que são regulares e as que não o são.

7.5.1 Observações de Base

(1) Todas as linguagens finitas (contendo um número finito de palavras) são regulares;
(2) Uma linguagem não regular deve, então, comportar um número infinito de palavras (saliente-se que o inverso não é verdadeiro);
(3) Se uma linguagem comporta um número infinito de palavras, não existe um limite para o tamanho das palavras que a compõem;
(4) Toda linguagem regular é aceita por um autômato finito que comporta um número fixo *m* de estados.
(5) Sejam uma linguagem regular infinita e um autômato com *m* estados que a aceita. Para toda palavra com tamanho superior a *m*, a execução dessa palavra pelo autômato deve passar por um mesmo estado s_k pelo menos duas vezes, com uma parte não vazia da palavra separando essas duas passagens (ver Figura 7.1)
(6) Conseqüentemente, se *x* , *u* e *y* são as palavras representadas na Figura 7.1, todas as palavras da forma xu^*y são também aceitas pelo autômato e fazem parte da linguagem;

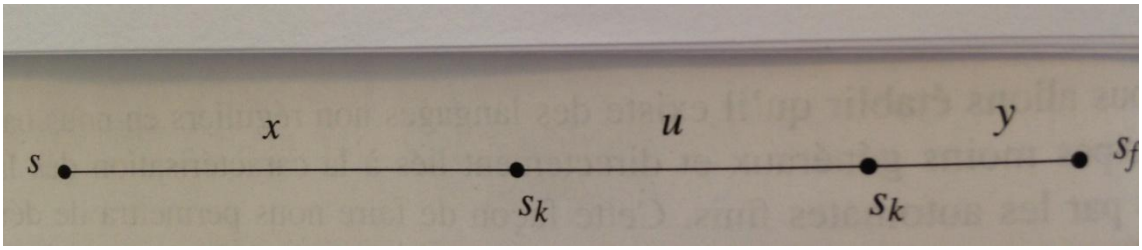


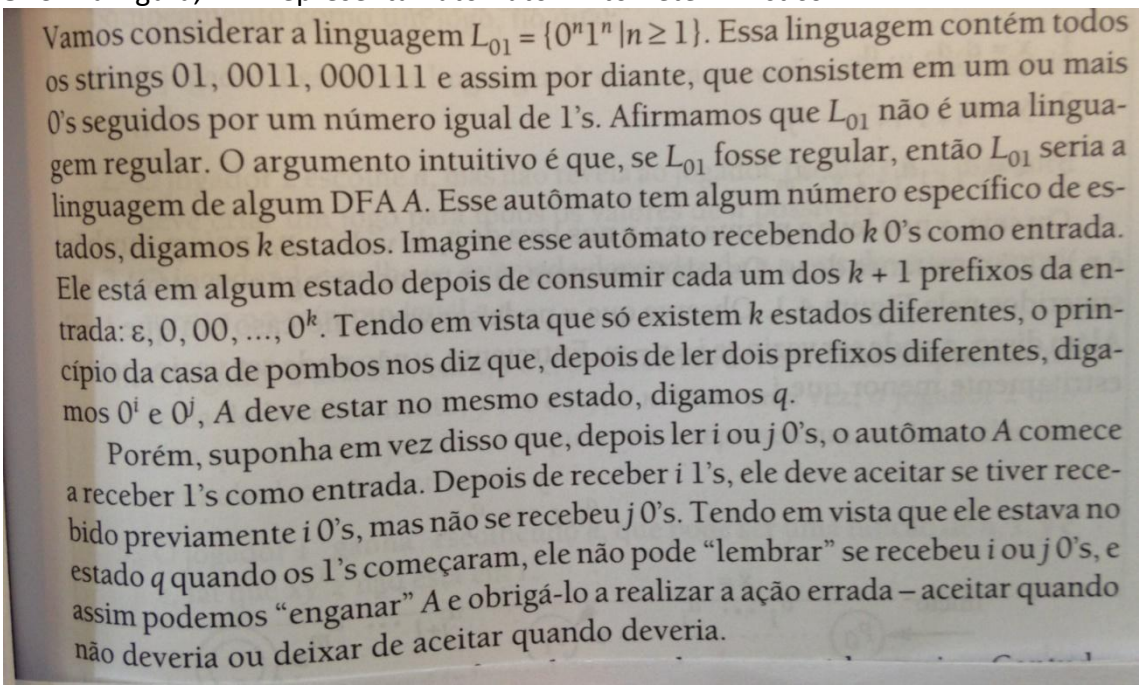
Figura 7.1 Autômato e Linguagem regular infinita

Resumidamente, as observações acima implicam que, para uma linguagem regular infinita, consegue-se permanecer no escopo da linguagem repetindo uma parte das palavras suficientemente longas pertencentes a tal linguagem. A fundamentação da afirmativa de que há linguagens que não são regulares será feita na próxima subseção pela demonstração de que há linguagens que não satisfazem tal propriedade. Nela, será apresentado também o Lema do bombeamento para linguagens regulares, o qual propõe um método para detectar se uma dada linguagem é regular ou não.

7.5.2 Lema do Bombeamento para as Linguagens Regulares

A Figura abaixo resume um exemplo de uma linguagem que não satisfaz a propriedade das linguagens regulares citadas no final da subseção anterior.

OBS: Na Figura, *DFA* representa *Autômato Finito Determinístico*.



Vamos considerar a linguagem $L_{01} = \{0^n 1^n \mid n \geq 1\}$. Essa linguagem contém todos os strings 01, 0011, 000111 e assim por diante, que consistem em um ou mais 0's seguidos por um número igual de 1's. Afirmamos que L_{01} não é uma linguagem regular. O argumento intuitivo é que, se L_{01} fosse regular, então L_{01} seria a linguagem de algum DFA A. Esse autômato tem algum número específico de estados, digamos k estados. Imagine esse autômato recebendo k 0's como entrada. Ele está em algum estado depois de consumir cada um dos $k + 1$ prefixos da entrada: $\varepsilon, 0, 00, \ldots, 0^k$. Tendo em vista que só existem k estados diferentes, o princípio da casa de pombos nos diz que, depois de ler dois prefixos diferentes, digamos 0^i e 0^j, A deve estar no mesmo estado, digamos q.

Porém, suponha em vez disso que, depois ler i ou j 0's, o autômato A comece a receber 1's como entrada. Depois de receber i 1's, ele deve aceitar se tiver recebido previamente i 0's, mas não se recebeu j 0's. Tendo em vista que ele estava no estado q quando os 1's começaram, ele não pode "lembrar" se recebeu i ou j 0's, e assim podemos "enganar" A e obrigá-lo a realizar a ação errada – aceitar quando não deveria ou deixar de aceitar quando deveria.

Teorema 7.2. (Versão simplificada do Lema do Bombeamento das Linguagens Regulares): Seja L uma linguagem regular infinita. Então, existem x , u , $y \in \Sigma^*$, com $u \neq \varepsilon$, tais que $xu^n y \in L$, $\forall n \geq 0$.

A demonstração deste teorema se baseia nas observações da subseção anterior. No caso, consideram-se um autômato com m estados que aceita L e uma palavra qualquer com comprimento superior a m. Decompõe-se tal palavra em três partes x , u e y , como na Figura 7.1 . As palavras x , u e y podem ser obtidas através de uma palavra qualquer suficientemente longa. Pode-se assim reforçar o Teorema 7.2 e obter o teorema que se segue.

Teorema 7.3. (Lema do Bombeamento das Linguagens Regulares): Seja L uma linguagem regular infinita. Para todo $w \in L$ tal que $|w| \geq |Q|$ - onde Q é o conjunto de estados de um autômato determinístico (ou não determinístico que não comporte transições com comprimento superior a 1) que aceita L - existem x , u , $y \in \Sigma^*$, com $u \neq \varepsilon$ e $|xu| \leq |Q|$, tais que $xuy = w$ e, $\forall n \geq 0$, $xu^n y \in L$. Saliente-se que as palavras $xu^n y$ que o teorema garante que também pertencem à linguagem L (caso os requisitos especificados sejam atendidos) são produzidas a partir do "bombeamento" de u em w. Além disso, cada nova palavra de L produzida por tal bombeamento pode ser: menor do que w (caso $n = 0$), igual a w (caso $n = 1$) ou maior do que w (caso $n > 1$).

7.5.3 Exemplos de Aplicação dos Teoremas do Bombeamento

(1) A linguagem a^n é regular, pois pelo Teorema 7.2, existem x , u , $y \in \Sigma^*$, com $u \neq \varepsilon$, tais que $xu^n y \in L \quad \forall n \geq 0$. De fato , tais restrições são satisfeitas para o caso em que $x = y = \varepsilon$ e $u = a$.

(2) A linguagem $L = a^n$ é regular. Prova baseada no Teorema 7.3: Seja m o número de estados de um autômato finito determinístico que aceita L, ou seja, m = |Q|. Seja w uma palavra de L com comprimento igual ou superior a m, ou seja, $|w| \geq m$. Neste caso, o teorema 7.3 exige que se encontrem palavras $x, u, y \in \Sigma^*$, tal que: $\boldsymbol{w = xuy}$, $\boldsymbol{u \neq \varepsilon}$, $\boldsymbol{|xu| \leq m}$ e, além disso, $\boldsymbol{\forall n \geq 0}$, $\boldsymbol{xu^n y \in L}$. Seja $x = a^p$; $u = a^q$; $y = a^r$. As restrições acima podem ser resumidas da seguinte forma:

   (a) $|xu| \leq m$, com $u \neq \varepsilon$: $p + q \leq m$ (indicando que a quantidade de símbolos a da palavra xu é menor ou igual a m);
   (b) $w = xuy$: como $|w| = p + q + r$, então, $p + q + r \geq m$.
   (c) $\forall \boldsymbol{n \geq 0}$, $\boldsymbol{xu^n y \in L}$? Em outras palavras, o teorema pergunta se, pertencendo w a L (ou seja, sabendo que a palavra específica $\boldsymbol{xu^n y}$, onde n = 1, pertence a $\boldsymbol{L}$) as palavras $\boldsymbol{xu^n y}$ - obtidas pelo "*bombeamento da palavra u*"- também $\in \boldsymbol{L}$?

   Análise das restrições*:* existem palavras $x, u, y \in \Sigma^*$ que respeitam todas essas restrições. De fato, considerando $p = 0$ (ou seja, $x = \varepsilon$), $r = 0$ (ou seja, $y = \varepsilon$), e $q = m$ (ou seja, $u = a^m$), $\forall \boldsymbol{n \geq 0}$, $\varepsilon a^{nm} \varepsilon \in \boldsymbol{L}$, provando que L é regular.

(3) A linguagem $ba^n ba$ é regular, pois pelo Teorema 7.2, existem x , u , $y \in \Sigma^*$, com $u \neq \varepsilon$, tais que $\forall n \geq 0$, $xu^n y \in L$. De fato , tais restrições são satisfeitas para o caso em que $x = b$; $u = a$ e $y=ba$.

(4) A linguagem L dada por $(a \cup b)^*$ é regular. Provando pelo Teorema 7.3: assuma-se que a linguagem dada seja regular. Seja m o número de estados do autômato que a reconhece. Neste caso, o teorema 7.3 exige que, para toda palavra w de L com comprimento igual ou superior a m, ou seja, $|w| \geq m$, encontrem-se palavras $x, u, y \in \Sigma^*$, tal que: $\boldsymbol{w = xuy}$, $\boldsymbol{u \neq \varepsilon}$, $\boldsymbol{|xu| \leq m}$ e $\forall n \geq 0$, $xu^n y \in L$.

   De fato , tais restrições são satisfeitas para todo tipo de palavra w pertencente a tal linguagem, a saber:

   - $w = a^*$: neste caso, $x = y = \varepsilon$; $u = a$;
   - $w = b^*$: neste caso, $x = y = \varepsilon$; $u = b$;
   - $w \in (a \cup b)^+ - (a^* \cup b^*)$, que é a linguagem formada por todas as palavras que tenham pelo menos um a e pelo menos um b: neste caso, há duas possibilidades:

- w começa com a : neste caso, $x = \varepsilon$; $u = a$; y $\in \Sigma^+$ / y contém pelo menos um b.
  Exs: w = ab : $x = \varepsilon$; $u = a$; y = b ;
  w = abbaaa : $x = \varepsilon$; $u = a$; y = bbaaa
- w começa com b : neste caso, $x = \varepsilon$; $u = b$; y $\in \Sigma^+$ / y contém pelo menos um a.
  Exs: w = ba : $x = \varepsilon$; $u = b$; y = a ;
  w = bbbaaa : $x = \varepsilon$; $u = b$; y = bbaaa

(5) A linguagem L denotada por $ab^{2n+1}a$ é regular. Provando pelo Teorema 7.3: assuma-se que a linguagem dada seja regular. Seja m o número de estados do autômato que a reconhece. Neste caso, o teorema 7.3 exige que, para toda palavra w de L com comprimento igual ou superior a m, ou seja, $|w| \geq m$, encontrem-se palavras $x, u, y \in \Sigma^*$, tal que: $\boldsymbol{w = xuy}$, $\boldsymbol{u \neq \varepsilon}$, $\boldsymbol{|xu| \leq m}$ e $\forall n \geq 0$, $xu^n y \in L$.

De fato , tais restrições são satisfeitas na seguintes situação: x = a, u = bb ; $y = ba$.

(6) A linguagem $(b \cup a^n)$ é regular. Isso será demonstrado da seguinte forma: nota-se que tal linguagem é composta pela união das linguagens $\{a\}$ e a^* (sendo esta última denotada por a^n). Como a linguagem $\{a\}$ é regular (pois é finita) e como anteriormente foi provado que a união de duas linguagens regulares também é regular (seção 6.4.1., onde se mostrou o método para produzir o autômato finito que aceita a linguagem correspondente à união de duas linguagens regulares), para provar que a linguagem $(b \cup a^n)$ é regular, basta provar, pelo teorema de bombeamento para as linguagens regulares, que a linguagem a^n também é regular. Isso foi feito no exemplos (1) e (2) acima.

(7) A linguagem $a^n b^n$ não é regular. Isso será demonstrado por meio da prova de que não é possível encontrar palavras x, u, y tais que $xu^k y \in a^n b^n \ \forall k$, situação em que o teorema 7.2 é contrariado. Para provar isso, admita-se que a linguagem $a^n b^n$ é regular, ou seja, que conforme exigido pelo teorema 7.2, existem x, u e y que satisfazem os requisitos do teorema. Sejam analisadas as alternativas para valor de u :

u $\in a^*$: impossível, pois, caso se repita u (aplicando o teorema 7.2), o número de letras a não será mais igual ao número de letras b;

u $\in b^*$: impossível, pois, caso se repita u (aplicando o teorema 7.2), o número de letras b não será mais igual ao número de letras a;

u $\in (a \cup b)^* - (a^* \cup b^*)$, ou seja, u é qualquer palavra de Σ^* que não seja formada exclusivamente por símbolos a nem exclusivamente por símbolos b : impossível, visto que, neste caso, uma ocorrência de b precederá uma ocorrência de a em u^k (k > 1) e $xu^k y$ não pertencerá, assim, à linguagem $a^n b^n$. Logo, o teorema 7.2 não se aplica a tal linguagem e ela não pode ser regular.

(8) A linguagem $L = a^{n^2}$ (composta por todas as palavras sobre o alfabeto *{a}* cujo comprimento é um quadrado perfeito) não é regular. Prova por absurdo baseada no Teorema 7.3: seja *m* o número de estados de um autômato finito que aceita *L* , ou seja, $m = |Q|$. Seja $w = a^{\alpha}$ uma palavra de *L* , tal que $|w| \geq m$, ou seja, α é um quadrado perfeito maior ou igual a *m*. Neste caso, inicialmente é necessário encontrar valores $x, u, y \in \Sigma^*$, tal que: $a^{\alpha} = xuy$, $u \neq \varepsilon$, e $|xu| \leq m$. Seja $x = a^p$; $u = a^q$; $y = a^r$. Assim sendo, as restrições acima podem ser resumidas da seguinte forma:

(a) $w \in L = a^{\alpha} = wuy = a^p a^q a^r$, com $a^q \neq \varepsilon$: $p + q \leq m$ (indicando que a quantidade de símbolos *a* da palavra *xu* é menor ou igual a *m*).

(b) Como $p + q + r = |w|$, então $\alpha = p + q + r$ é um quadrado perfeito com valor maior ou igual a *m* .

Logo, o conjunto formado pelas restrições *(a)* e *(b)* é satisfatível, uma vez que ambas são atendidas, por exemplo, quando *q* vale *1*, *p* vale *0* e *r* vale $(m^2 - 1)$. De fato, em tal situação, $\alpha = p + q + r = m^2$, que é um quadrado perfeito com valor maior ou igual a *m*.

Checando agora a última exigência do Teorema: 7.3, ou seja: sabendo que, dada uma palavra $w \in L$ em que $|w| \geq Q$, existem $x , u , y \in \Sigma^*$, com $u \neq \varepsilon$ e $|xu| \leq |Q|$, tais que $xuy = w$, pode-se garantir que: $\forall n \geq 0$, $xu^n y \in L$?

Análise: considere-se a palavra *w´ produzida* pelo bombeamento da palavra $w = a^{\alpha}$ na situação particular em que *n* assume o valor *2*, ou seja, a palavra $w' = xu^2y$. Tem-se que: | w´ | = p + 2q + r, que não é um quadrado perfeito, pois:

- Como $p + q + r = m^2$, então, $m^2 < p + 2q + r$;
- Como $p + 2q + r = (p + q + r) + q = m^2 + q$ e $q \leq m$, então $p + 2q + r \leq m^2 + m$;
- Como $m^2 + m < (m + 1)^2$, considerando o fato de que $(m + 1)^2$ é o quadrado perfeito imediatamente superior ao quadrado perfeito m^2 correspondente ao módulo de w, pode-se concluir que o módulo $p + 2q + r$ da palavra w´, que tem valor menor ou igual a $m^2 + m$, não é um quadrado perfeito, ou seja, a palavra w´ = $\boldsymbol{xu^2y}$ não pertence a *L*.

Conseqüentemente, provou-se que *L* não é uma linguagem regular.