MATERIAL TEORIA DA COMPUTAÇÃO-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, RajeevMotwani (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 aprender a reconhecer os casos que recaem nessa terceira categoria. 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, isso é, de avaliar se um programa qualquer sempre para, ou seja, não entra em"loops" ou chamadas recursivas infinitas ao processar qualquer uma de suas instâncias. Por exemplo, ainda não foi descoberto procedimento algum que permita decidir se o processamento do programa abaixo para (ou seja, converge para o valor " *1*") qualquer que seja o número natural submetido a ele:

```
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;
```

Segundo a Conjectura matemática de Collatz (ou Conjectura *3n + 1*), o processo de cálculo representado em tal programa sempre converge para o valor *1*, qualquer que seja o natural *n* submetido a ele. Assim sendo, caso tal conjectura seja demonstrada como verdadeira um dia (convertendo-se em teorema), o programa acima representará um procedimento efetivo para a geração da série numérica por ele produzida (pois sempre parará quando a série convergir para *1*). Por outro lado, caso se prove que a conjectura é falsa, o referido programa não representará um procedimento efetivo (pois entrará em loop para alguma instância natural *n* cuja série correspondente jamais convirja para *1*).

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;
- { εa } = {aε } = {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 deL_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 = { Ø , { ε }, {a}, {b}, {ε, a}, {ε, b}, {a, b}, {ε, a, b}, ... ,

{εa}, {εb}, {εε}, {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, ababab, abababab,...},...
{a, ε, ab, abab, ababab, ...,}, ...,

{ε, εa}, {ε, εb}, {ε, εε}, {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´ = ΣU{), (, 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 representadapela expressão regular *(α)** 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.

Demonstração Por Indução (Ver arquivo *TC-1-1-Anexo-INDUÇAO-MATEMATICA* com resumo explicativo de Prova Por Indução)

Enunciado: Uma linguagem é regular se, e somente se, ela é denotada por uma expressão regular.

A bi-implicação será demonstrada por meio da demonstração da conjunção das duas implicações envolvidas em I e II (ou seja, por meio da prova das proposições expressas em $P_1(x)$ e $P_2(x)$, respectivamente).

I) $P_1(x)$: se x é linguagem regular em um alfabeto $\sum$, então x é denotada por uma expressão regular.

OBS: o domínio da propriedade P_1 é o conjunto das linguagens regulares em um alfabeto $\sum$.

(i) **Base de indução**:

- $P_1(\emptyset)$ é verdadeira, pois a linguagem $\emptyset$ é denotada pela expressão regular $\emptyset$;
- $P_1(\{\varepsilon\})$ é verdadeira, pois a linguagem $\{\varepsilon\}$ é denotada pela expressão regular ε;
- $P_1(\{s\})$, onde $s \in \sum$, é verdadeira, pois a linguagem $\{s\}$ é denotada pela expressão regular s .

(ii) **Passo de indução**:

Sejam L_1 e L_2 linguagens em $\sum$. Assim sendo, o passo de indução exige que se provem as seguintes proposições:

1^a) $(P_1(L_1) \wedge P_1(L_2)) \rightarrow P_1(L_1 \cup L_2)$
2^a) $(P_1(L_1) \wedge P_1(L_2)) \rightarrow P_1(L_1 . L_2)$
3^a) $P_1(L_1) \rightarrow P_1(L_1^*)$

Provando a primeira implicação: $(P_1(L_1) \wedge P_1(L_2)) \rightarrow P_1(L_1 \cup L_2)$

Hipótese: $(P_1(L_1) \wedge P_1(L_2))$ é verdadeiro, ou seja os itens a) e b) a seguir são verdadeiros:

a) A linguagem regular L_1 é representada por uma expressão regular α;
b) A linguagem regulara L_2 é representada por uma expressão regular β.

Pergunta-se: sendo a) e b) proposições verdadeiras, pode-se concluir que $P_1(L_1 \cup L_2)$ é verdadeira, ou seja, que a expressão regular $L_1 \cup L_2$ pode ser representada por uma expressão regular?
A resposta é sim pois:

- ✓ Como L_1 é representada pela expressão regular α, L(α) = L_1 (ver definição 1.6);
- ✓ Como L_2 é representada pela expressão regular β, L(β) = L_2 (ver definição 1.6);

Logo, $L_1 \cup L_2$ = L(α) ∪ L(β) que, pelo item *(3)* da definição 1.6 é igual a L((α ∪ β)), o que indica, pela definição do mapeamento L (definição 5.2) que *(α ∪ β)* é a expressão regular que denota a linguagem $L_1 \cup L_2$.

Provando a *2ª)* e *3ª)* implicações:

Tais implicações podem ser provadas de modo análogo à prova da *1ª)* implicação.

II) $P_2(x)$: Se x é uma linguagem em um alfabeto ∑ denotada por uma expressão regular, então x é uma linguagem regular.

A prova desta proposição pode ser feita de modo análogo à efetuada no item *I)*.

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

a) Expressão regular: $(a \cup (ab))$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, …} = {aε, ab, abb, abbb, abbbb, …}

c) Expressão regular: (ab∪ $(b)^*$)representa a linguagem regular formada pela palavra ab e pelas palavras que são sequências 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: $(a \cup b)^*$: representa a linguagem regular composta por todas as palavras possíveis a partir do alfabeto {a,b}. De fato:

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

{ ε , a, 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 $\Sigma = \{a_1,...,a_n\}$ é representada pela expressão regular $(a_1 \cup ... \ a_n)^*$, comumente abreviada como $\mathbf{\Sigma^*}$.

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

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

{ 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 $L((a \cup b) (a \cup b)^*)$ corresponde à linguagem regular $L((a \cup b)^*)$ 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 $\Sigma = \{a_1, ... ,a_n\}$ é representada pela expressão regular $(a_1 \cup ... \cup a_n)(a_1 \cup ... \cup a_n)^*$, 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, ... }, 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 ≥ 0)de símbolos *a* que termina com um *b* , a ser representada aqui por:

  $W = a^nb$ (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:

  $W_1 = W...W = a^{n1}b...a^{nk}b$, 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, ... }, 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 ≥ 0) de símbolos *b* que termina com um *a* , a ser representada aqui por:

  $W = b^na$

  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($(a^*b)^*$)∪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 $(a^*b)^*\cup(b^*a)^*= (a\cup b)^*$.

i) Prove que $\forall L,\ L^* = (L^*)^*$

$L^* = \{w \mid \exists k \geq 0 \text{ e } w_1, \ldots, w_k \in L \mid w = w_1 \ldots w_k\}$

$(L^*)^* = \{w' \mid \exists k' \geq 0 \text{ e } w'_1, \ldots, w'_{k'} \in L^* \mid w' = w'_1 \ldots w'_{k'}\}$

$(L^*)^* = \{w' \mid w' \text{ é composta por sequências de palavras de } L^*\}$

$\therefore (L^*)^* = \{w' \mid w' \text{ é composta de sequências de } k' \text{ palavras } (k' \geq 0), \text{ onde cada uma dessas } k' \text{ palavras é formada por sequências de } k \text{ palavras de } L \ (k \geq 0)\}$

Logo, todas as palavras $w' \in (L^*)^*$ são sequências finitas de k' palavras de L, ou seja, $\forall w', w' \in L^*$ (caso particular em que $k = k'$).

Analogamente, $\forall w \in L^*$, $w \in$ também a $(L^*)^*$, pois w corresponde a uma sequência finita de palavras de L.

j) Prove que $\exists L_1, \exists L_2 \mid (L_1 \cup L_2)^* \neq L_1^* \cup L_2^*$

$L_1^* = \{w \mid \exists k \geq 0 \text{ e } w_1, \dots, w_k \in L_1 \mid w = w_1 \dots w_k\}$

$L_2^* = \{w' \mid \exists k' \geq 0 \text{ e } w_1', \dots, w_k' \in L_2 \mid w' = w_1' \dots w_k'\}$

$(L_1 \cup L_2)^* = \{w'' \mid \exists k'' \geq 0 \text{ e } w_1'', \dots w_k'' \in (L_1 \cup L_2) \mid w'' = w_1'' \dots w_k''\}$ ∴ que cada palavra w_i'' $(1 \leq i \leq k'')$ que compõe a palavra w'' pode pertencer a L_1 ou a L_2.

$L_1^* \cup L_2^* = \{w''' \mid w''' \in L_1^* \text{ ou } w''' \in L_2^*\}$. Logo, por exemplo, caso $L_1 \cap L_2 = \emptyset$, qualquer palavra w''' será composta por palavras pertencentes tão somente a L_1 ou tão somente a L_2, isto é, w''', diferentemente de w'', jamais será composta por palavras provenientes de L_1 e de L_2. Logo, neste caso, palavras do tipo $w'' \in (L_1 \cup L_2)^*$ não pertencerão a $L_1^* \cup L_2^*$.

Consequentemente, por exemplo, sempre que $L_1 \cap L_2 = \emptyset$, $(L_1 \cup L_2)^* \neq L_1^* \cup L_2^*$.

Exc: $L_1 = \{a\}$; $L_2 = \{b\}$.

- **FAZER EXERCÍCIOS DA LISTA 1**

6. As Linguagens Não Regulares

A título de lembrança, de acordo com o conteúdo estudado, sempre que nos referimos a “linguagens”, “problemas” , “expressões regulares” e a “procedimentos efetivos”, tais referências estão NECESSARIAMENTE associadas à ideia de um dado alfabeto finito.

Conforme mencionado, as expressões regulares permitem descrever certas linguagens (as linguagens regulares), mas não todas. Nesta seção será mostrado que existem linguagens não regulares por meio de uma observação fundamental: *não há expressões regulares suficientes para representar todas as linguagens!*

O raciocínio é simples: apesar de haver um número infinito de expressões regulares e de linguagens, diferentes conjuntos infinitos podem apresentar tamanhos diferentes. O tamanho de um conjunto C (cardinalidade de C, ou |C|) é definido pela sua quantidade de elementos, baseando-se, assim, nos números naturais e em suas propriedades. Apesar de essa noção ser intuitivamente bastante direta no caso de conjuntos finitos, torna-se menos intuitiva no caso dos conjuntos infinitos. Torna-se importante, assim, contar com um critério que estabeleça quando dois conjuntos têm o mesmo tamanho. Tal critério consiste em mostrar que os elementos dos dois conjuntos, um a um, podem ser colocados em correspondência, ou seja, que há uma bijeção entre esses conjuntos.

Exemplo: os conjuntos (0, 1, 2, 3} e {+, %, #, @} têm o mesmo tamanho, conforme mostra a bijeção {(0, +), (1, %), (2, #), (3,@)}.

Além da cardinalidade, outra medida ligada ao tamanho de um conjunto é relevante: a ordinalidade, a qual é definida, UNICAMENTE, sobre conjuntos ordenados (sobre os quais foi definida uma relação de ordem). Dois conjuntos têm a mesma ordinalidade se existe entre eles uma bijeção que preserve as relações de ordem desses conjuntos, ou seja, se as imagens dos elementos *a* e *b* do primeiro conjunto forem, respectivamente, *a’* e *b’* do segundo conjunto e, além disso, se $a \leq b$ (ou seja, *a* precede *b*) no primeiro conjunto, consequentemente, $a' \leq b'$ (ou seja, *a’* precede b’) no segundo conjunto. Para os conjuntos finitos, a cardinalidade e a ordinalidade coincidem (quando temos uma bijeção entre conjuntos, temos uma bijeção que respeita a ordem). O mesmo não ocorre para os conjuntos infinitos.

Propriedades da relação “ter mesma cardinalidade”:

- Ela é reflexiva: $|A| = |A|$;
- Ela é simétrica: $|A| = |B| \rightarrow |B| = |A|$;
- Ela é transitiva: $((|A| = |B|) \wedge (|B| = |C|)) \rightarrow (|A| = |C|)$

É natural agrupar em classes os conjuntos de mesma cardinalidade. Sabemos que a cardinalidade de um conjunto finito contendo *n* elementos é *n*. A seguir veremos como entender a cardinalidade de um conjunto infinito. O primeiro e menor dos conjuntos infinitos é o dos números naturais $N = \{0, 1,2, ...\}$. Utilizando o conceito de bijeção, podemos definir os conjuntos que têm mesma cardinalidade que *N*.

Definição 5.3. Um conjunto infinito é *enumerável* se existe uma bijeção entre ele e o conjunto dos números naturais.

A cardinalidade dos conjuntos enumeráveis é habitualmente denotada $\aleph_0$ ($\aleph$ é lida como "aleph") é a primeira letra do alfabeto hebraico. Diversos conjuntos são enumeráveis (ou seja, têm cardinalidade $\aleph_0$), tais como os mostrados abaixo.

Exemplos:

a) O conjunto dos números pares é enumerável, conforme prova a seguinte função bijetora*: {(0,0),(2,1),(4,2),(6,3),...}*. Uma generalização deste exemplo é que todo sub-conjunto infinito dos naturais é enumerável.

b) O conjunto de palavras sobre o alfabeto *{a, b}* é enumerável, fato demonstrado pela seguinte bijeção com os naturais em que as palavras são ordenadas pelo tamanho e, empatando tal quesito, pela ordem lexicográfica: *{(ε, 0), (a, 1), (b,2), (aa, 3), (ab, 4), (ba, 5), (bb, 6), (aaa, 7), ... }*.

c) O conjunto dos números racionais é enumerável. Tal conjunto é formado pelos valores *a/b* , tal que: $b \neq 0$, *a* seja inteiro e *b* seja natural e, a fim de evitar repetições de mesmos valores, *a* e *b* não tenham fatores comuns. Isso pode ser provado por uma bijeção ordenada, primeiramente, pela soma |a| + b , em segundo lugar, para somas idênticas, pelo valor crescente do módulo do numerador e, em terceiro lugar, para tais módulos idênticos, pelo valor do numerador.

{(0/1 *, 0), ((-1/1 , 1), (1/1, 2), (-1/2, 3), (1/2, 4), (-2/1, 5), (2/1 , 6), (-1/3, 7), (1/3, 8), (-3/1, 9), (3/1, 10), ... }*.

OBS: note que os valores *a/b* em que *a* é inteiro (positivo ou negativo) e *b* é um inteiro negativo, apesar de não se encaixarem diretamente nas restrições acima, também pertencem aos racionais, pois cada um desses valores é igual a um dos valores do conjunto acima. Exemplo: *1/-2* é racional, pois é igual a *-1/2*, o qual, por sua vez, é um racional que, por respeitar as restrições, aparece explicitamente no conjunto acima. O mesmo vale para *-1/-2*, que é igual a *1/2* .

d) As expressões regulares são enumeráveis. De fato, as expressões regulares são cadeias de caracteres sobre um alfabeto finito. O conjunto de todas as cadeias de caracteres sobre um alfabeto é enumerável (ver exemplo *b* acima). Como as expressões regulares são

um sub-conjunto infinito dessas cadeias de caracteres, são também enumeráveis (ver exemplo *a* acima).

Nem todos os conjuntos infinitos têm cardinalidade $\aleph_0$. Alguns deles têm cardinalidade superior a $\aleph_0$, tal como o conjunto de todos os sub-conjuntos de um conjunto enumerável.

Teorema 5.2. O conjunto de todos os sub-conjuntos de um conjunto infinito enumerável não é enumerável.

Demonstração. A demonstração deste teorema se baseia em uma técnica simples mas poderosa: *a técnica da diagonalização.* Seja um conjunto infinito enumerável A

$$A = \{a_0, a_1, a_2, ...\}$$

e seja S o conjunto de todos seus sub-conjuntos. Suponha-se que S seja enumerável por uma bijeção em que cada um de seus elementos esteja em correspondência biunívoca com os elementos do conjunto *{s_0, s_1, s_2, ...}*.

Construamos o seguinte "tableau" infinito para representar a bijeção :

	a_0	a_1	a_2	a_3	a_4	...
s_0	×	×		×		
s_1	×	□		×		
s_2		×	×		×	
s_3	×		×	□		
s_4		×		×	□	
⋮						

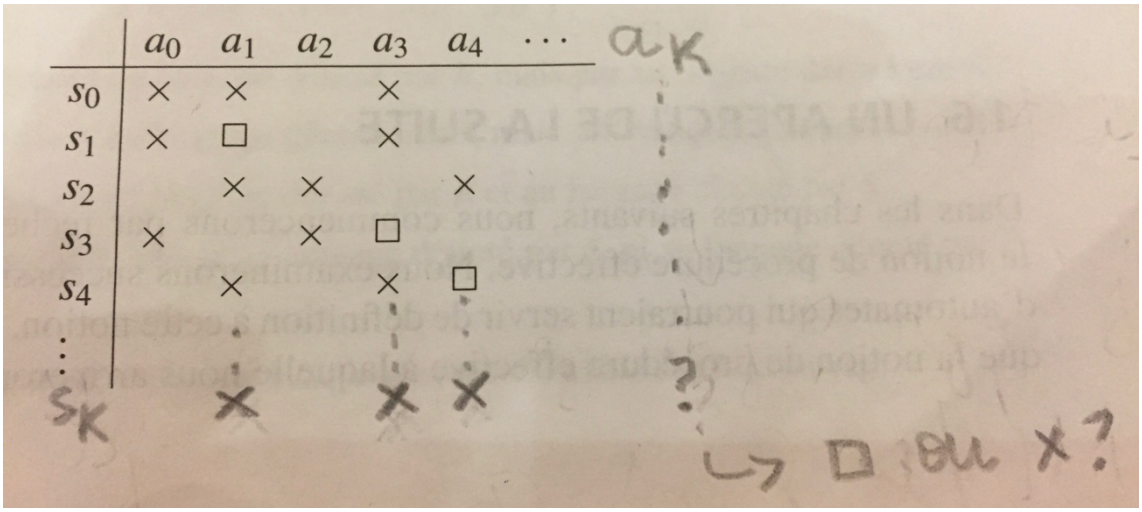
Nesse "tableau" a bijeção é representada da seguinte forma: cada linha (elemento) s_j corresponde ao subconjunto de A (elemento de *S*) composto pelos elementos a_i daquela mesma linha que estão marcados por uma cruz. Exemplo de pares da bijeção: { ({a_0 , a_1 , a_3 , ...} , s_0), ({a_0 , a_3 , ...} , s_1), ..., ({a_1 , a_3 , ...} , s_4), ... }. A título de simplificação de notação, a partir deste ponto esta bijeção será re-escrita por meio do seguinte conjunto:

$$S = \{ s_0 , s_1 , ..., s_4 , ... \}.$$

Consideremos agora o conjunto :

$D = \{a_i \mid a_i \notin s_i\}$.

Note que tal conjunto é composto pelos elementos de A correspondentes aos pequenos retângulos das diagonais, ou seja, *D* é o seguinte subconjunto de *A:*
D = { a_1 , a_3 , a_4 , ...} , onde a_1 , a_3 e a_4 pertencem a *D* por não pertencerem, respectivamente, a s_1 , s_3 e s_4 . Contudo, apesar de *D* existir e ser um subconjunto de *A* (**OBSERVAÇÃO RITA**: sendo, portanto, enumerável), ele não pode ser um dos s_i . De fato, provando por absurdo, suponhamos que $D = s_k$. Neste caso, temos que:



onde, pela definição de *D*, $a_k \notin s_k$, ou seja, deveria ser alocado um pequeno retângulo na intersecção da coluna de a_k com a linha de s_k . Consequentemente, ainda pela definição de *D* , o mesmo a_k precisaria ser inserido em s_k , ou seja, o retângulo da intersecção acima precisaria ser substituído por uma cruz. Mas, paradoxalmente, a partir disso, a_k não poderia mais pertencer a *D* (pois violaria a definição do conjunto). Se o retirássemos, ele voltaria a pertencer... Logo, chegamos a uma contradição lógica que mostra que a hipótese inicial, afirmando que existe um s_k correspondente a *D* (uma vez que *D* é subconjunto de *A*) é falsa. Consequentemente, é falso que o conjunto de subconjuntos de *A* é enumerável.

Por analogia com o fato de o conjunto das partes de um conjunto finito A conter $2^{|A|}$ elementos, é de praxe representar o conjunto de subconjuntos dos naturais por $2^{\mathbb{N}}$ e sua cardinalidade por $2^{\aleph_0}$.

O teorema 5.2. é uma das causas dos limites fundamentais da informática que estudaremos. Ele implica, por exemplo, que todas as linguagens não podem ser regulares. De fato:

- O conjunto das linguagens é o conjunto dos subconjuntos de um conjunto enumerável (o conjunto de palavras) e não é, portanto, enumerável (teorema 5.2.);

- O conjunto das linguagens regulares é enumerável, pois cada linguagem regular é denotada por uma expressão regular e o conjunto das expressões regulares é enumerável;
- Há, assim, mais (e muito mais!) linguagens além das linguagens regulares.

Tal resultado nos permitirá mostrar que existem problemas insolúveis por meio de procedimentos efetivos. De fato, conforme apresentado anteriormente, cada conjunto de palavras pode ser visto como um problema (no caso, as instâncias positivas de tal problema), ou seja, cada problema corresponde à uma linguagem. Assim sendo, o conjunto de problemas corresponde ao conjunto de linguagens. Como o conjunto de linguagens corresponde ao conjunto de subconjuntos do conjunto enumerável infinito das palavras, pelo teorema 5.2, ele não é enumerável. Logo, o conjunto de problemas também não é enumerável. Por outro lado, o conjunto de procedimentos efetivos é enumerável, pois é um subconjunto da linguagem (um procedimento efetivo é uma cadeia finita de caracteres, ou seja, uma palavra). Por exemplo, o conjunto de programas em C é enumerável, pois representa um subconjunto do conjunto enumerável da linguagem produzida sobre o alfabeto que contém os caracteres do C. Logo, não pode haver um procedimento efetivo para cada problema, pois não existe uma bijeção entre problemas e procedimentos efetivos, o que indica que há problemas que não são passíveis de solução .

7. 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 7.1: 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.

7.1 Autômatos Finitos Determinísticos:

7.1.1 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 7.1).

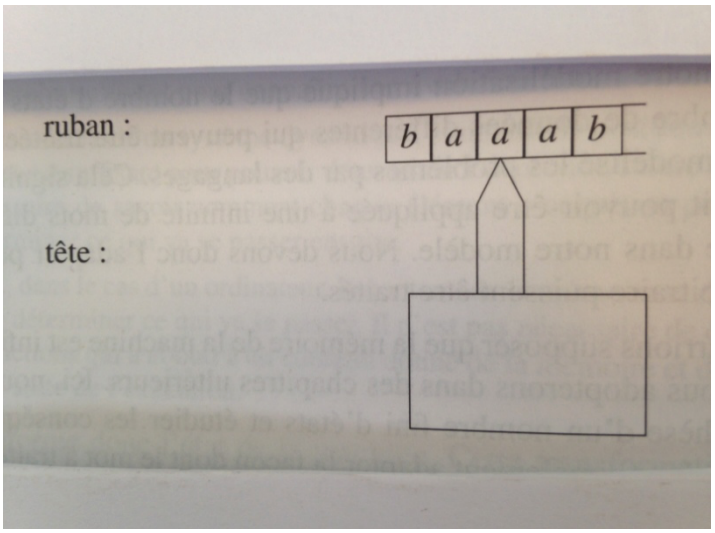


Figura 7.1 Autômato Finito Determinístico

- 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.

7.1.2 Formalização

Autômato finito determinístico: definido por uma quíntupla M = (Q ,Σ, δ, s , F), onde:

- Q é um conjunto finito de estados possíveis;
- Σé um alfabeto;
- δ : Q X $\Sigma \rightarrow$ 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 7.1 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 7.2 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 7.3 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 7.4).

Definição 7.4 Uma palavra w é aceita por um autômato M se, e somente se:

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

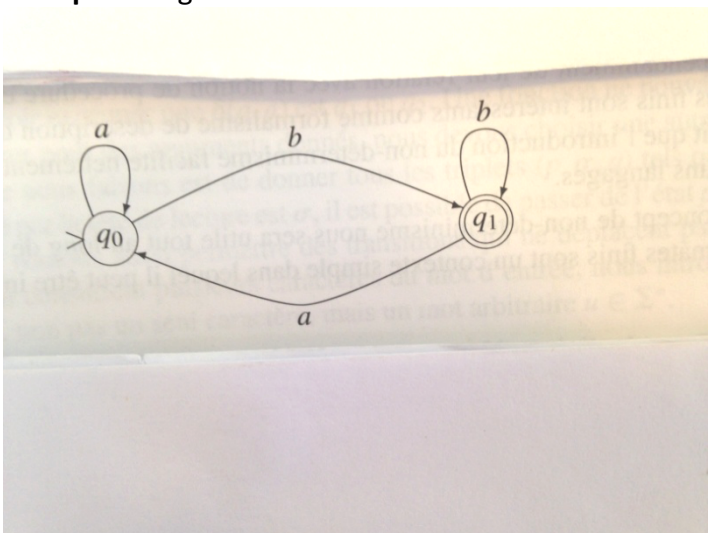
Exemplo: Seja um autômato $M = (Q, \Sigma, \delta, s, F)$ cujo alfabeto é $\Sigma = \{a, b\}$ e seja w = ababa. A execução de w pelo autômato é:
$(s,ababa) \vdash_M (q_1, baba) \vdash_M (q_2, aba) \vdash_M (q_3, ba) \vdash_M (q_4, a) \vdash_M (q_5, \varepsilon)$

Definição 7.5 Uma linguagem aceita por um autômato M (com notação *L(M)*) é o conjunto de palavras definido da seguinte forma:
$L(M) = \{w \in \Sigma^* \mid (s, w) \vdash_M^* (q, \varepsilon) \text{ com } q \in F\}$.

7.1.3 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 σconectando 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

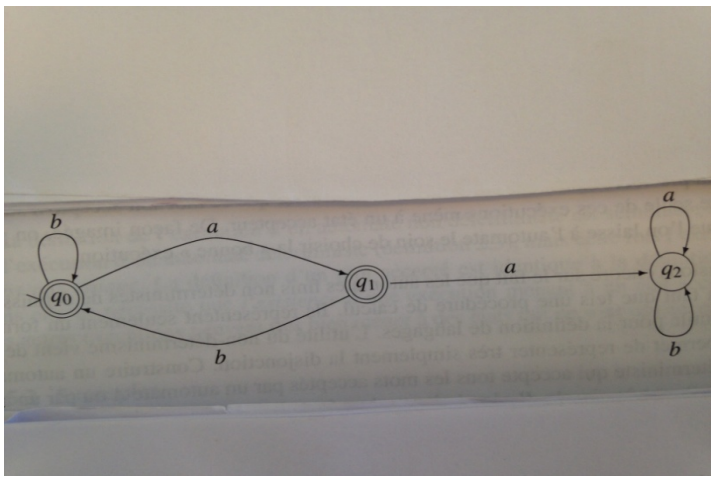


representa o seguinte autômato que aceita as palavras terminadas em *b* : $Q = \{q_0, q_1\}$; $\Sigma = \{a, b\}$; $s = q_0$, $F = \{q_1\}$ e

δ:

q	σ	δ(q, σ)
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} :



- **FAZER EXERCÍCIOS DA LISTA 2**