

# Linguagens Regulares, Operações Regulares

# Definição de Linguagem Regular

Relembre a definição de linguagem regular:

DEF: A **linguagem aceita por** um AF  $M$  é o conjunto de todos os strings que são aceitos por  $M$  e é denotada por  $L(M)$ .

Queremos entender que tipo de linguagens são regulares. O reconhecimento de elementos de linguagens desse tipo é extremamente rápido! Seria bom saber, por exemplo, quais das seguintes linguagens são regulares:

# Exemplos de Linguagens

– Números primos unários:

$$\begin{aligned} & \{ 11, 111, 11111, 1111111, 11111111111, \dots \} \\ & = \{ 1^2, 1^3, 1^5, 1^7, 1^{11}, 1^{13}, \dots \} \\ & = \{ 1^p \mid p \text{ é um número primo} \} \end{aligned}$$

– Quadrados unários:

$$\begin{aligned} & \{ \varepsilon, 1, 1^4, 1^9, 1^{16}, 1^{25}, 1^{36}, \dots \} \\ & = \{ 1^n \mid n \text{ é um quadrado perfeito} \} \end{aligned}$$

– Strings de bits que são palíndromos:

$$\begin{aligned} & \{ \varepsilon, 0, 1, 00, 11, 000, 010, 101, 111, \dots \} \\ & = \{ x \in \{0,1\}^* \mid x = x^R \} \end{aligned}$$

Veremos mais adiante se essas linguagens são ou não regulares.

# Linguagens Finitas

Todas as linguagens dos exemplos anteriores têm cardinalidade *infinita*

NOTA: Os strings que constituem a linguagem são finitos (como todos neste curso); entretanto, o conjunto de strings é infinito, em cada um dos exemplos anteriores.

Antes de examinar linguagens infinitas, vamos primeiro nos ater a linguagens finitas.

# Linguagens de Cardinalidade 1

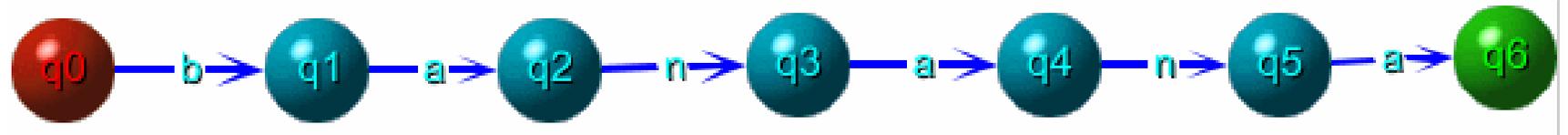
Q: Uma linguagem que contém um único string é regular? Por exemplo,

{ banana }

é regular?

# Linguagens de Cardinalidade 1

R: Sim.



Q: O que há de errado nesse exemplo?

# Linguagens de Cardinalidade 1

R: De fato, nada. Esse é um exemplo de um AF *não determinístico*. Ele constitui uma forma mais concisa de definir a linguagem {banana}

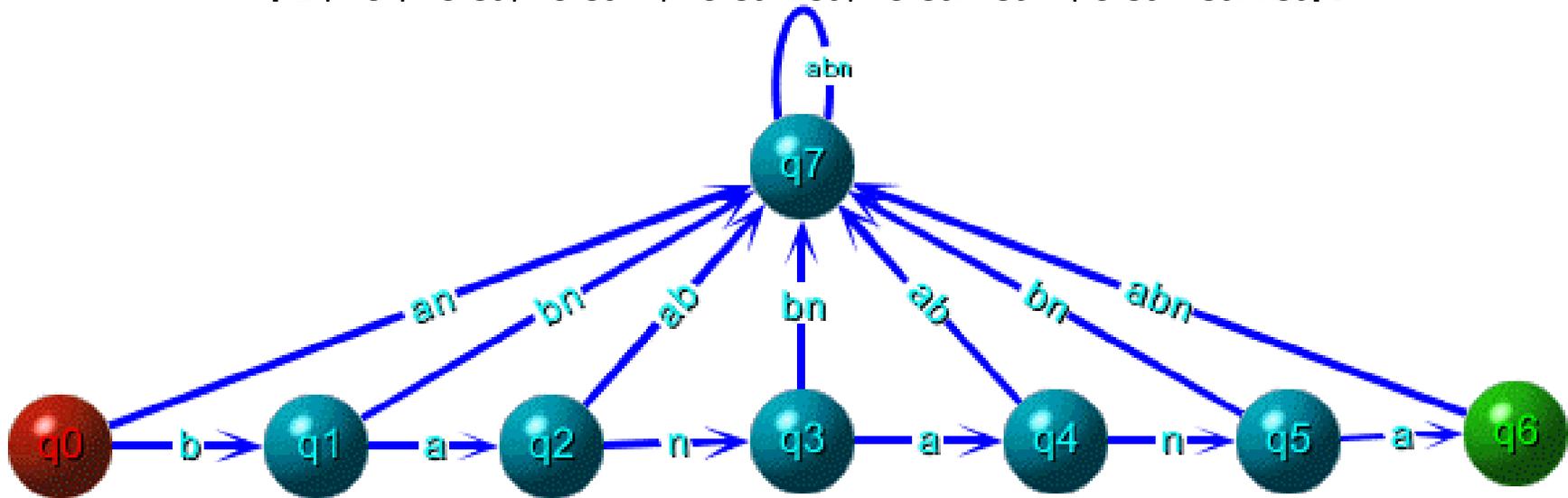


Vamos tratar de não determinismo nas próximas aulas. Então:

Q: Existe uma maneira de corrigir esse autômato, tornando-o determinista?

# Linguagens de Cardinalidade 1

R: Sim, basta adicionar um **estado de falha** q7; I.e., incluir um estado que “suga” todo os strings diferentes de “banana” – exceto strings que são **prefixos** de “banana”  
{ $\epsilon$ , b, ba, ban, bana, banan,banana}.



# Demonstração usando JFLAP

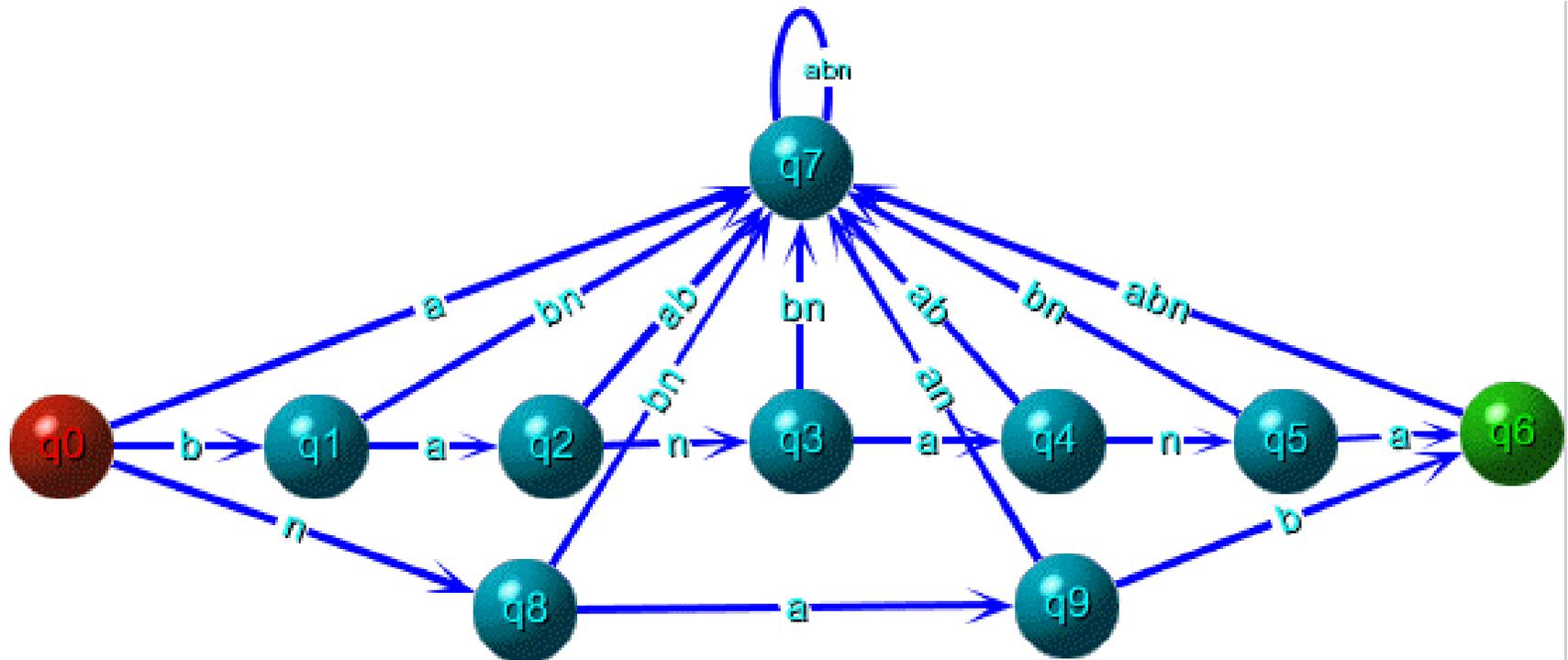
[JFlap](#)

# Dois Strings

Q: E linguagens contendo apenas 2 strings  
são regulares? Por exemplo  
{ banana, nab } ?

# Dois Strings

R: Apenas adicione uma nova rota:



# Número Finito Arbitrário de Strings

Q1: E mais strings? Por exemplo  
{ banana, nab, ban, babba } ?

Q2: Ou menos (o conjunto vazio):  
 $\emptyset = \{ \} ?$



# Número Finito Arbitrário de Strings: Linguagem Vazia

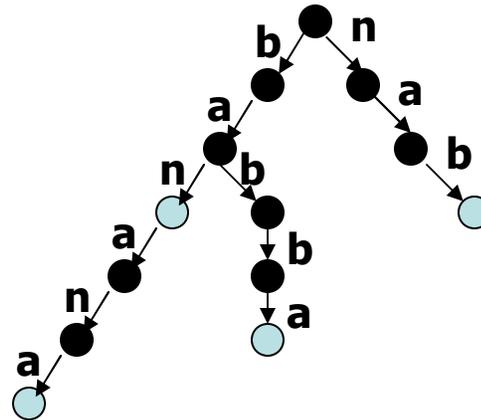
R2: Construa um autômato com um único estado e com conjunto de estados de aceitação  $F$  *vazio* !



# Número Finito Arbitrário de Strings

THM: Toda linguagem finita é regular.

*Prova* : É sempre possível construir uma árvore em que cada ramo representa um string. Por exemplo:



A raiz é o estado inicial; as folhas são estados de aceitação; adicione um estado de falha para finalizar a construção.

# Cardinalidade Infinita

Q: Toda linguagem regular é finita?

# Cardinalidade Infinita

R: Não! Muitas linguagens infinitas são regulares.

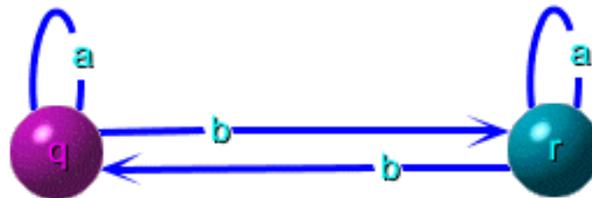
Erro comum 1: Os strings de uma linguagem regular são finitos, portanto a linguagem deve ser finita.

Erro comum 2: Linguagens regulares são – por definição – aceitas por um autômato *finito*, portanto são finitas.

Q: Dê um exemplo de uma linguagem *infinita* mas regular.

# Cardinalidade Infinita

- bitstrings com número par de b's



- O exemplo mais simples é  $\Sigma^*$



- muitos, muitos outros

Exercício: Pense num critério para não finitude

# Operações Regulares

Você provavelmente já usou operações regulares ao especificar pesquisas avançadas utilizando programas como `emacs`, `egrep`, `perl`, `python`, **etc.**

Vamos trabalhar com três operações básicas:

1. União
2. Concatenação
3. Kleene-star (fecho de Kleene)

E uma quarta definida em termos dessas:

4. Kleene-plus (fecho positivo de Kleene)

# Operações Regulares – Tabela Resumo

Operação	Simbolo	Versão UNIX	Significado
União	∪		casa um dos padrões
Concatenação	•	<i>implicito em UNIX</i>	casa os padrões em sequencia
Kleene-star	*	*	casa o padrão 0 ou mais vezes
Kleene-plus	+	+	casa o padrão 1 ou mais vezes

# Operações Regulares – União

UNIX: para pesquisar todas as linhas contendo vogais em um texto, podemos usar o comando

```
egrep -i `a|e|i|o|u`
```

O padrão “*vogal*” é casado por qualquer linha que contenha um dos símbolos a, e, i, o ou u.

Q: O que é um padrão de string?

# Padrão de String

A: Uma boa maneira de definir um padrão de string é como um conjunto de strings, i.e. uma linguagem. A linguagem para um dado padrão é o conjunto de todos os strings que satisfazem o predicado do padrão.

EX: *padrão-vogal* =

{ strings que contenham pelo menos um dos símbolos: a e i o u }

# Padrões UNIX vs. Padrões em Computabilidade

Em UNIX, supõe-se implicitamente que um string cada com um padrão se esse padrão ocorre como *substring* deste.

Nesse curso, entretanto, um padrão deve especificar o string *completo*, e não apenas um substring.

# Operações Regulares – União

Linguagens formais: Dados os padrões

$A = \{\text{aardvark}\}$ ,  $B = \{\text{bobcat}\}$ ,

$C = \{\text{chimpanzee}\}$

a união desses padrões resulta em

$A \cup B \cup C = \{\text{aardvark, bobcat, chimpanzee}\}$

# Operações Regulares - Concatenação

UNIX: pra pesquisar todas as ocorrências duplas consecutivas de vogais, usamos:

```
egrep -i `(a|e|i|o|u)(a|e|i|o|u)`
```

O padrão “*vogal*” foi repetido.

Usamos parênteses para especificar onde exatamente ocorre a concatenação.

# Operações Regulares - Concatenação

Linguagens formais. Considere o resultado anterior:

$L = \{\text{aardvark, bobcat, chimpanzee}\}$

Q: Qual é a linguagem resultante de concatenar  $L$  com ela própria:

$L \bullet L$  ?

# Operações Regulares - Concatenação

A:  $L \bullet L =$

$\{\text{aardvark, bobcat, chimpanzee}\} \bullet \{\text{aardvark, bobcat, chimpanzee}\}$

$=$

$\{\text{aardvarkaardvark, aardvarkbobcat, aardvarkchimpanzee,}$   
 $\text{bobcataardvark, bobcatbobcat, bobcatchimpanzee,}$   
 $\text{chimpanzeeaardvark, chimpanzeebobcat, chimpanzeechimpanzee}\}$

Q1: O que é  $L \bullet \{\epsilon\}$  ?

Q2: O que é  $L \bullet \emptyset$  ?

# Álgebra de Linguagens

A1:  $L \bullet \{\varepsilon\} = L$ . De modo geral,  $\{\varepsilon\}$  é a ***identidade*** da “álgebra” de linguagens. I.e., se pensarmos na concatenação como sendo multiplicação,  $\{\varepsilon\}$  age como o número 1.

A2:  $L \bullet \emptyset = \emptyset$ . Dualmente a  $\{\varepsilon\}$ ,  $\emptyset$  age como o número zero, obliterando qq string com o qual é concatenado.

Nota: Na analogia entre números e linguagens, a adição corresponde à união e a multiplicação corresponde à concatenação, formando assim uma “álgebra”.

# Operações Regulares – Kleene- \*

UNIX: pesquisar por linhas que consistem puramente de vogais (incluindo a linha vazia):

```
egrep -i `^(a|e|i|o|u)*$`
```

NOTA:  $\wedge$  and  $\$$  são símbolos especiais em expressões regulares UNIX que ligam o padrão ao *início* e ao *fim* da linha, respectivamente. Isso pode ser usado para converter uma operação regular de Linguagens Formais em uma expressão regular UNIX equivalente.

# Operações Regulares – Kleene- \*

Linguagens formais : Considere a  
linguagem

$$B = \{ ba, na \}$$

Q: Qual é a linguagem  $B^*$  ?

# Operações Regulares – Kleene- \*

A:

$$B^* = \{ ba, na \}^* =$$

{  $\epsilon$ ,

ba, na,

baba, bana, naba, nana,

bababa, babana, banaba, banana,

nababa, nabana, nanaba, nanana,

babababa, bababana, ... }

# Operações Regulares – Kleene-

+

Kleene-+ é tal como Kleene-\* exceto que o padrão deve ocorrer *pelo menos uma vez*.

UNIX: pesquisar por linhas que consistem puramente de vogais (exceto linha vazia):

```
egrep -i `^(a|e|i|o|u)+$`
```

Linguagens formais :  $B^+ = \{ba, na\}^+ =$

{ ba, na,

baba, bana, naba, nana,

bababa, babana, banaba, banana,

nababa, nabana, nanaba, nanana,

babababa, bababana, ... }

# Gerando Linguagens Regulares

A razão pela qual linguagens regulares são chamadas regulares é a seguinte:

THM: Linguagens regulares são aquelas que podem ser geradas a partir de linguagens finitas pela aplicação de operações regulares.

Esse teorema será provado adiante.

Q: Podemos começar a partir de linguagens ainda mais básicas que linguagens finitas arbitrárias?

# Gerando Linguagens Regulares

R: Sim. Podemos começar com linguagens que consistem de um único string, os quais consistem de um único caractere. Essas são chamadas linguagens regulares “atômicas”.

EX: Para gerar a linguagem finita

$$L = \{ \text{banana, nab} \}$$

Podemos começar com as linguagens atômicas

$$A = \{a\}, B = \{b\}, N = \{n\}.$$

Então podemos expressar  $L$  como:

$$L = (B \bullet A \bullet N \bullet A \bullet N \bullet A) \cup (N \bullet A \bullet B)$$

# Exercício

1. Expressar as linguagens a seguir na forma de uma expressão regular, no estilo de expressões regulares UNIX, e usando operações regulares.
  - a. A linguagem  $L$  sobre o alfabeto  $\{0,1\}$  cujos strings possuem tamanho múltiplo de 3 ou terminam com 1.
  - b. A linguagem  $L$  sobre o alfabeto  $\{0,1\}$  cujos strings começam com 0 e terminam com 10 ou com 11
2. Construa o AFD dos itens anteriores usando o JFlap