



# Teoria da Computação

- ❑ Expressões e linguagens regulares

# Introdução

---

## ❑ Definições

- ❑ Basicamente, podemos dizer algo abrangente de forma específica. Definindo um padrão de busca, temos uma lista de possibilidades.

“Como o brinquedo LEGO, várias pecinhas diferentes, cada uma com sua característica, que juntas compõem estruturas completas e podem ser arranjadas com infinitas combinações diferentes.”

- ❑ Expressões Regulares são **metacaracteres** que casam um padrão. É uma maneira de procurar um texto especificando padrões.

- Padrões para validar data, horário, número IP, e-mail, RG, CPF, telefones... Exemplo: ER para números de telefones => ([0-9]{2})? [0-9]{4}-[0-9]{4}. Casa qualquer uma das strings: “83 3224-1063”, “83 3254-3421”, “82 2343-1212”...

# Introdução

---

## ❑ História

- O termo deriva do que se chamou de álgebra de conjuntos regulares (“*regular sets*”), do matemático Stephen Cole Kleene.
- 1968, Ken Thompson implementava no qed um comando de contexto que aceitava expressões regulares. Sua sintaxe? g/RE/p (Global Regular Expression Print)... Mais tarde deu origem ao famoso grep nos sistemas Unix.
- 1986, criado um pacote pioneiro em C, chamado regex e de graça! Daí a norma IEEE POSIX 1003.2 (POSIX.2) padroniza expressões regulares.

# Expressões regulares

---

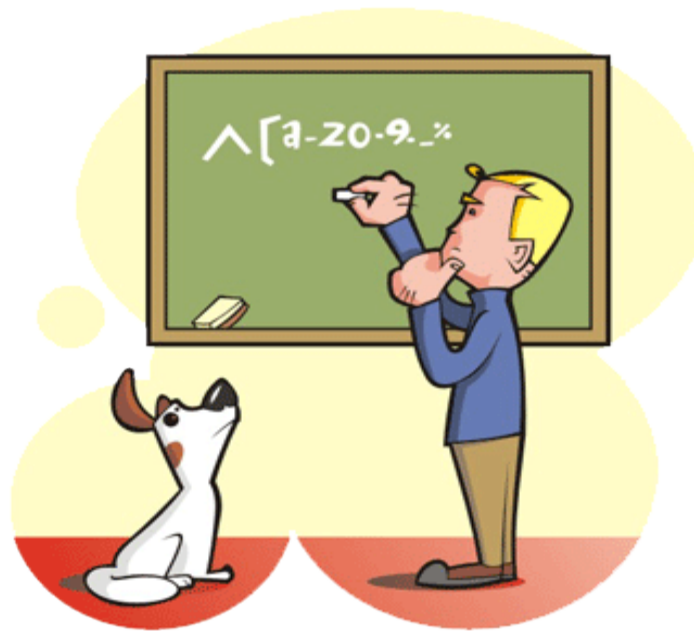
- ❑ Úteis em pesquisa de texto (grep do Unix) e em compiladores (Lex, Flex, analisadores lexicais)
- ❑ Alternativa simpática aos NFA
- ❑ Equivalentes aos autómatos → provar
- ❑ Características algébricas; permitem expressão declarativa das cadeias pretendidas
- ❑ Expressões regulares denotam linguagens
  - **$01^*+10^*$**
  - Linguagem das cadeias binárias que têm um 0 seguido de zero ou mais 1's, ou um 1 seguido de zero ou mais 0's

# Operadores sobre linguagens

---

- ❑ **União** de duas linguagens  $L$  e  $M$  ( $L \cup M$ ), é o conjunto das cadeias que pertencem a  $L$ , a  $M$ , ou a ambas
  - $L = \{001, 10, 111\}$     $M = \{\varepsilon, 001\}$     $L \cup M = \{\varepsilon, 001, 10, 111\}$
- ❑ **Concatenação** de duas linguagens  $L$  e  $M$  ( $LM$  ou  $L.M$ ), é o conjunto de cadeias que se obtém concatenando qualquer cadeia em  $L$  com qualquer cadeia em  $M$ 
  - $LM = \{001, 10, 111, 001001, 10001, 111001\}$
- ❑ **Fecho** de uma linguagem  $L$  ( $L^*$ ) é o conjunto de cadeias que se obtém concatenando um número arbitrário de cadeias de  $L$ , incluindo repetições, isto é,  $L^* = \cup_{i \geq 0} L^i$ , em que  $L^0 = \{\varepsilon\}$ 
  - $L = \{0,1\}$     $L^*$  é a linguagem das cadeias binárias

# Os Metacaracteres



# Sopa de letrinhas

---

Metacaractere	Nome
---------------	------

.	Ponto
---	-------

[ ]	Lista
-----	-------

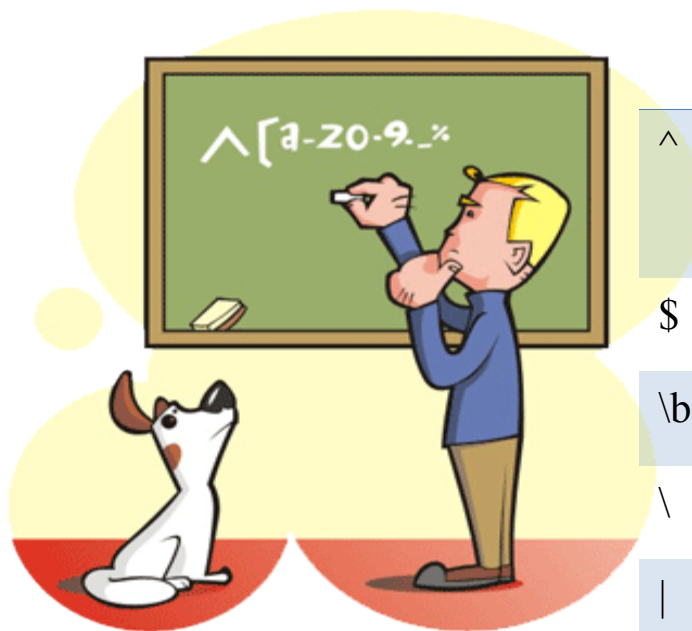
[^]	Lista negada
-----	--------------

?	Opcional
---	----------

*	Asterisco
---	-----------

+	Mais
---	------

{ }	Chaves
-----	--------



Metacaractere	
---------------	--

^	Circunflexo
---	-------------

\$	Cifrão
----	--------

\b	Borda
----	-------

\	Escape
---	--------

	Ou (pipe)
--	-----------

()	Grupo
----	-------

\1	Retrovisor
----	------------

# Sopa de letrinhas – exemplos

---

**Miller, Miler, Müller, Müller, Mueler, Mueller:**

`M(ü|i|ue)ll?er`

**Siglas de 4 letras, começando com UF:**

`UF[A-Z]{2}`

**Tags HTML `<b>`, `</b>`, `<i>`, `</i>`, `<u>`, `</u>`:**

`</?[BbUuli]>`

**Títulos (primeira linha não pontuada):**

`^[A-Za-z0-9 ]+$`

**Palavras:**

`\b[A-Za-z]+\b`



# Padrões POSIX

---

Classe POSIX	Similar	Significa
<code>[ :upper: ]</code>	<code>[ A-Z ]</code>	Letras maiúsculas
<code>[ :lower: ]</code>	<code>[ a-z ]</code>	Letras minúsculas
<code>[ :alpha: ]</code>	<code>[ A-Za-z ]</code>	Maiúsculas/Minúsculas
<code>[ :alnum: ]</code>	<code>[ A-Za-z0-9 ]</code>	Letras e números
<code>[ :digit: ]</code>	<code>[ 0-9 ]</code>	Números
<code>[ :xdigit: ]</code>	<code>[ 0-9A-Fa-f ]</code>	Números Hexadecimais

# (Alguns m|M)etacaracteres tipo barra-letra

---

Classe POSIX	Similar	Significa
\d	[[:digit:]]	Dígito
\D	[^[:digit:]]	Não-dígito
\w	[[:alnum:]]_	Palavra
\W	[^[:alnum:]]_	Não-palavra
\s	[[:space:]]	Branco
\S	[^[:space:]]	Não-branco



Editores de texto e Linguagens

Google Docs

# Google Docs

The image shows a screenshot of the Google Docs web interface. The top menu bar includes File, Edit, View, Insert, Format, Table, Tools, and Help. The 'Edit' menu is open, showing options like Undo, Redo, Cut, Copy, Paste, Select all, Find and replace..., Edit CSS, and Edit HTML. The 'Find and replace...' option is selected, opening a dialog box. The dialog box has a 'Find what:' field containing the regular expression '[A-Z][a-z]+'. A dropdown menu is open below the 'Find what:' field, showing options: 'Match case' (checked), 'Match whole word only', 'Regular expression-style matching [?]' (checked), and 'Find previous'. The 'Replace with:' field is empty. The 'Find next' button is highlighted. The background shows a document with Lorem Ipsum text.

Google docs Untitled

File Edit View Insert Format Table Tools Help

Undo Ctrl+Z  
Redo Ctrl+Y  
Cut Ctrl+X  
Copy Ctrl+C  
Paste Ctrl+V  
Select all Ctrl+A  
Find and replace... Ctrl+F  
Edit CSS  
Edit HTML

Google docs Lorem ipsum dolor sit amet, consectetur...

File Edit View Insert Format Table Tools Help

Find what: [A-Z][a-z]+ Find next Replace with: Replace Replace all

- ✓ Match case
- Match whole word only
- ✓ Regular expression-style matching [?]
- Find previous

sit amet, consectetur adipiscing elit. Aliquam auctor risus ac diam molestie viverra. Donec enim metus, pulvinar sit amet vehicula sit in tincidunt massa, id rhoncus urna mattis id. Aenean at enim nibh placerat massa. Aliquam vehicula tortor eget lectus pellentesque id sagittis diam dictum viverra, felis eget auctor pharetra, quam lacus cursus enim, sit amet commodo dui lect libero. Aliquam imperdiet volutpat mattis. Praesent id suscipit nisi. Pellentesque porta dictum. Quisque ultrices tristique massa eget facilisis.

Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae adipiscing sem eget urna facilisis non pharetra neque ultrices. Praesent eleifend, lacus pharetra molestie, turpis augue lobortis lectus, eget tincidunt tortor arcu id ipsum. Prae faucibus mattis ullamcorper. Aenean vitae dui eros. Aliquam quis dolor elit. Suspendisse eleifend ligula. Aenean vulputate turpis vel nunc vehicula aliquet. Aenean magna urna, sodales ac, dictum sit amet sem. Duis odio sapien, feugiat quis commodo vitae, aliquam tellus. Integer lobortis, neque eu elementum adipiscing, tellus lorem commodo massa, aliquam risus orci eu est. Ut sapien magna, elementum ac dignissim ac, pulvinar ac urna nec placerat nibh.

Maecenas eu quam quis nisl pulvinar aliquet ac a augue. Pellentesque eget nisl vitae tempus congue. Mauris cursus varius mi et interdum. Donec sapien turpis, vehicula at a, ornare eget nisi. Maecenas vel sem ac leo rhoncus varius a id arcu. Quisque semper elit mattis auctor. Etiam euismod vestibulum felis vitae eleifend. Nullam iaculis faucibus Duis semper ornare odio. Morbi dignissim dictum felis, vitae lobortis elit condimentum r Quisque quis arcu eget ante bibendum consectetur.

Nam id nibh nec odio rhoncus lobortis. Curabitur ligula risus, tincidunt nec consequat

## Google Docs – observações

- Não há classes POSIX (como `[:alpha:]`), `\w`, e `[A-Z]` não suporta acentuação. Solução?  
`[A-Za-zÀ-u]+` (Posições 65 a 90, 97 a 122 e 192 a 252 da tabela ASCII).
- A opção 'Match whole word only' ('Fazer correspondência somente da palavra inteira') não casará palavras acentuadas ou com 'ç'.
- Retrovisores são referenciados por `$1`, `$2...` na expressão de substituição, mas na ER utiliza-se `\1`, `\2...` Por exemplo, palavras repetidas `([A-Za-z]+) \1`
- Quebras de linha e parágrafos podem ser identificados com `\n`, assim as ERs podem casar múltiplas linhas de uma vez.

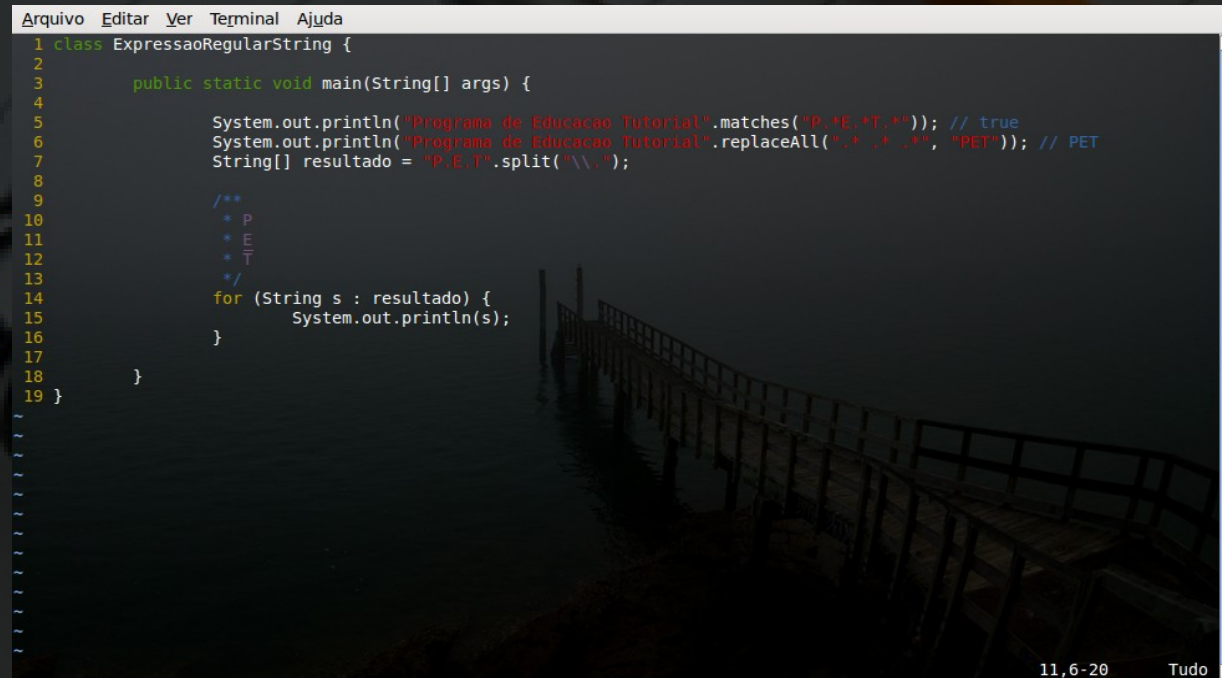
# Editores de texto e Linguagens

A spiral-bound notebook with lined paper is the background. An orange pencil and a silver pen with a black grip are resting on the pages. The notebook is open, showing the spiral binding on the left side.

Java

# Java

- Na J2SE 1.4 aparecem o pacote `java.util.regex` e classes `Pattern` e `Matcher`, além de algumas melhorias na classe `java.lang.String` (com os métodos `matches`, `replaceFirst`, `replaceAll`, `split`).
- Classe `String`



```
Arquivo  Editar  Ver  Terminal  Ajuda
1 class ExpressaoRegularString {
2
3     public static void main(String[] args) {
4
5         System.out.println("Programa de Educacao Tutorial".matches("P.*E.*T.*")); // true
6         System.out.println("Programa de Educacao Tutorial".replaceAll(".*", ".*", "PET")); // PET
7         String[] resultado = "P.E.T".split("\\.");
8
9         /**
10          * P
11          * E
12          * T
13          */
14         for (String s : resultado) {
15             System.out.println(s);
16         }
17     }
18 }
19 }
```

11,6-20 Tudo

# Java

- Classes Pattern e Matcher

```
Arquivo  Editar  Ver  Terminal  Ajuda
1 import java.util.regex.*;
2
3 class ExpressaoRegularRegex {
4
5     public static void main(String[] args) {
6
7         String pet      = "Programa de Educacao Tutorial";
8         Pattern er      = Pattern.compile("[A-Z].*");
9         Matcher result  = er.matcher(pet);
10
11         System.out.println(result.matches()); // TRUE
12         System.out.println(result.replaceAll("PET")); // PET
13
14         /**
15          * P
16          * E
17          * T
18          */
19         pet = "P.E.T";
20         er = Pattern.compile("\\\\.");
21         String[] matches = er.split(pet);
22
23         for(String s : matches) {
24             System.out.println(s);
25         }
26     }
27 }
```

1,1 Tudo



# Exemplos

## Data (dd/mm/aaaa)

../../..

[0-9]{2}/[0-9]{2}/[0-9]{4}

[0123][0-9]/[01][0-9]/[12][0-9]{3}

([012][0-9]|3[0-1])/([0[0-9]|1[012])/[12][0-9]{3}

(0[1-9]|[12][0-9]|3[01])/0[0-9]|1[012])/[12][0-9]{3}

## Horário (hh:mm)

....

[0-9]{2}:[0-9]{2}

[012][0-9]:[0-5][0-9]

([01][0-9]|2[0-3]):[0-5][0-9]

## Email (email@company.com)

.\*@.\*

[^@]+@[^@]+.[^\*]

[A-Za-z0-9\_.-]+@[A-Za-z0-9\_]+\.[a-z]+

[A-Za-z0-9\_.-]+@[A-Za-z0-9\_]+\.[a-z]{2,3}

# Exemplos de fecho

---

- $L = \{0, 11\}$ 
  - $L^0 = \{\varepsilon\}$
  - $L^1 = L = \{0, 11\}$
  - $L^2 = LL = \{00, 011, 110, 1111\}$
  - ...
  - $L^* = \{\varepsilon, 0, 11, 00, 011, 110, 1111, \dots\}$
  - Apesar de a linguagem  $L$  ser finita, bem como cada termo  $L^i$ ,  $L^*$  é infinita
- $L = \{\text{todas as cadeias só com 0's}\}$ 
  - $L^* = L$
  - $L$  é infinita, tal como  $L^*$
- $L = \emptyset$ 
  - $L^* = L^0 = \{\varepsilon\}$

# Construção de expressões regulares

---

## □ Base

- As constantes  $\varepsilon$  e  $\emptyset$  são expressões regulares
  - $L(\varepsilon) = \{\varepsilon\}$  e  $L(\emptyset) = \emptyset$
- Se  $a$  é um símbolo **a** é uma expressão regular
  - $L(\mathbf{a}) = \{a\}$
- Uma variável (ex:  $L$ ) é uma expressão regular
  - Representa qualquer linguagem

## □ Indução

- Se  $E$  e  $F$  são expressões regulares  $E + F$  é expressão regular
  - $L(E + F) = L(E) \cup L(F)$
- Se  $E$  e  $F$  são expressões regulares  $EF$  é expressão regular
  - $L(EF) = L(E)L(F)$
- Se  $E$  é expressão regular  $E^*$  é expressão regular
  - $L(E^*) = (L(E))^*$
- Se  $E$  é expressão regular  $(E)$  é expressão regular
  - $L((E)) = L(E)$

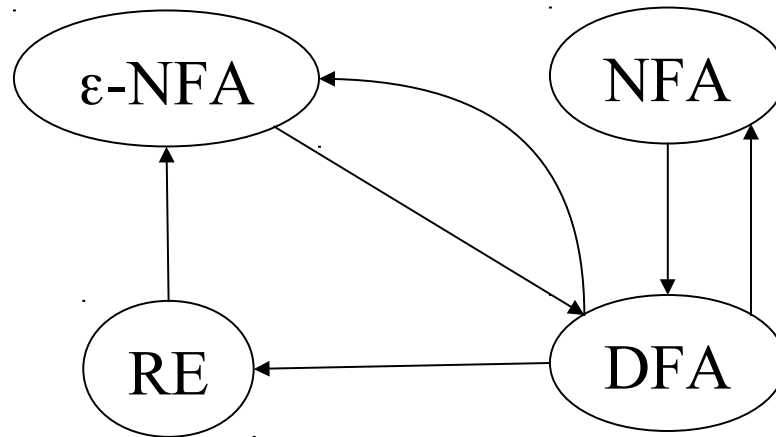
# Exemplo

---

- ❑ Escrever uma expressão regular para o conjunto de cadeias constituídas por 0's e 1's alternados.
  - **01**             $L(\mathbf{01}) = \{01\}$
  - **(01)\***             $\neq \mathbf{01}^*$      $L((\mathbf{01})^*) = \{\epsilon, 01, 0101, 0101, \dots\}$ 
    - ❑ Ainda faltam muitas!
  - **(01)\*+(10)\*+0(10)\*+1(01)\***
    - ❑ Está bem
  - **( $\epsilon+1$ )(01)\*( $\epsilon+0$ )**
    - ❑ Também.

# Equivalência FA - RE

---



- Mostrar que todas as linguagens definidas por autómatos também são definidas por expressões regulares ( $\text{DFA} \rightarrow \text{RE}$ )
- Mostrar que todas as linguagens definidas por RE também são definidas por autómatos ( $\text{RE} \rightarrow \epsilon\text{-NFA}$ )

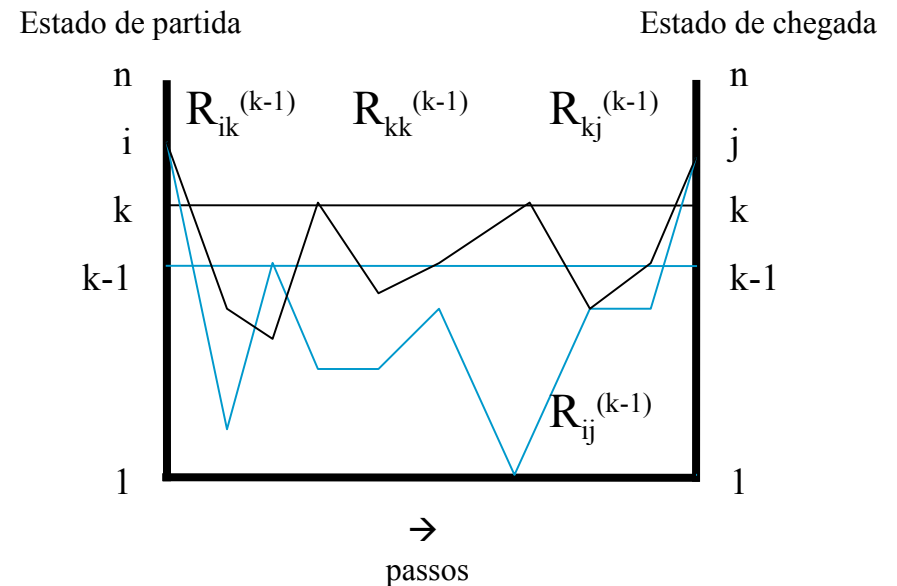
# Dos DFA's às RE's

---

- ❑ **Teorema:** Se  $L=L(A)$  para um DFA  $A$  então existe uma expressão regular  $R$  tal que  $L=L(R)$
- ❑ Dois métodos
  - numerar os estados de 1 a  $n$ ; construir REs que vão descrevendo caminhos sucessivamente mais complexos no DFA, até descrever todos os caminhos da entrada para cada estado final
  - Considerar os arcos etiquetados por RE; eliminar estados internos substituindo o seu “efeito” por REs

# Construção de caminhos

- ❑ Numerar os estados de 1 a  $n$ , começando pelo de entrada
- ❑  $R_{ij}^{(k)}$ 
  - Expressão regular cuja linguagem é o conjunto de cadeias  $w$  tal que  $w$  é a etiqueta de um caminho entre os nós  $i$  e  $j$ , sem passar em nenhum nó intermédio maior do que  $k$  (os extremos podem ser)
- ❑ Indução no número dos nós ( $k$ )



# Construção de caminhos

---

## □ Base

- $k=0$  significa sem nós intermédios (o menor é 1)
  - arco de  $i$  para  $j$  (RE é o respectivo símbolo; ou  $\emptyset$ , se não existir; ou  $a_1+a_2+\dots+a_m$ , se houver  $m$  arcos em paralelo)
  - nó  $i$  ( $i$  para  $i$ ) (RE é  $\varepsilon+a_1+a_2+\dots+a_m$ )

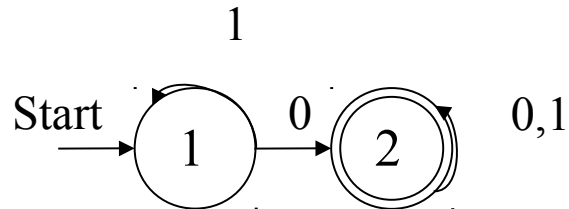
## □ Indução

- Hipótese: os caminhos que só usam nós até  $k-1$  já estão convertidos
- Existe caminho de  $i$  para  $j$  sem passar no estado  $k$ 
  - $R_{ij}^{(k-1)}$
- O caminho passa uma ou mais vezes em  $k$ :
  - $R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} (R_{kk}^{(k-1)})^* R_{kj}^{(k-1)}$
- Terminar:  $R_{ij}^{(n)}$  caminhos entre  $i$  e  $j$  usando todos os estados

- A RE da linguagem do autómato é a soma das expressões  $R_{ij}^{(n)}$  tais que  $j$  é um estado de aceitação.



# Exemplo DFA $\Rightarrow$ RE



❑ Autómato que reconhece cadeias com pelo menos um 0

❑  $R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} (R_{kk}^{(k-1)})^* R_{kj}^{(k-1)}$

❑  $R_{ij}^{(1)} = R_{ij}^{(0)} + R_{i1}^{(0)} (R_{11}^{(0)})^* R_{1j}^{(0)}$

$R_{11}^{(0)}$	$\epsilon+1$
$R_{12}^{(0)}$	0
$R_{21}^{(0)}$	$\emptyset$
$R_{22}^{(0)}$	$\epsilon+0+1$

$R_{11}^{(1)}$	$\epsilon+1+(\epsilon+1)(\epsilon+1)^*(\epsilon+1)$	$1^*$
$R_{12}^{(1)}$	$0+(\epsilon+1)(\epsilon+1)^*0$	$1^*0$
$R_{21}^{(1)}$	$\emptyset+\emptyset(\epsilon+1)^*(\epsilon+1)$	$\emptyset$
$R_{22}^{(1)}$	$\epsilon+0+1+\emptyset(\epsilon+1)^*0$	$\epsilon+0+1$

Simplificação:

$$(\epsilon+1)^* = 1^*$$

$$\emptyset R = R \emptyset = \emptyset$$

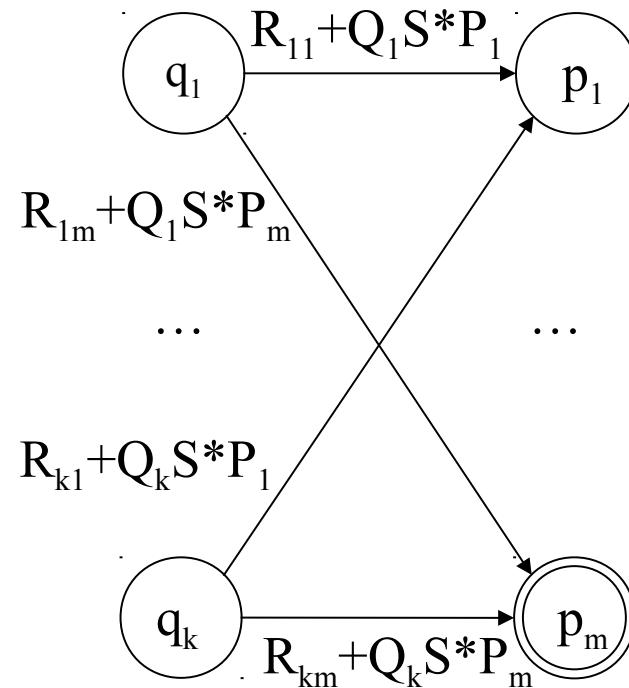
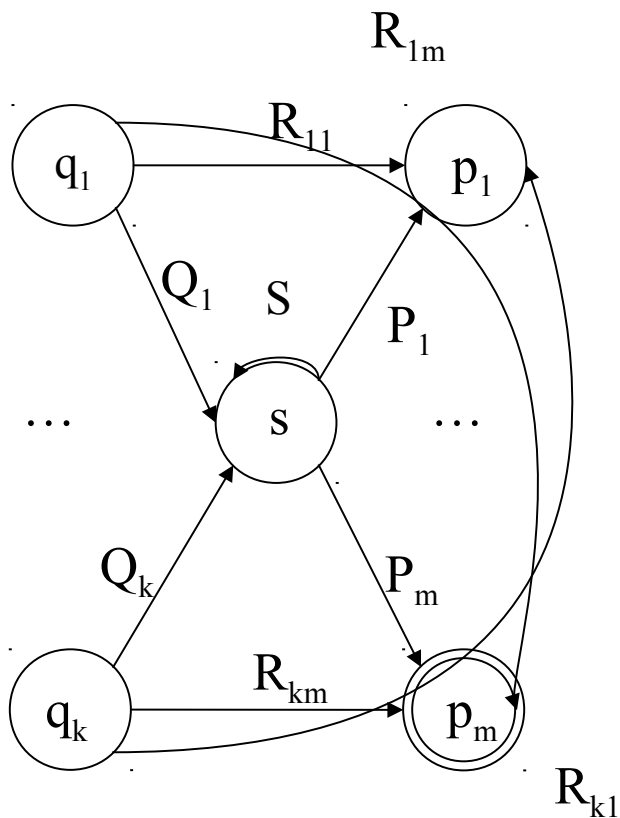
$$\emptyset + R = R + \emptyset = R$$

$R_{11}^{(2)}$	$1^* + 1^*0(\epsilon+0+1)^*\emptyset$	$1^*$
$R_{12}^{(2)}$	$1^*0 + 1^*0(\epsilon+0+1)^*(\epsilon+0+1)$	$1^*0(0+1)^*$
$R_{21}^{(2)}$	$\emptyset + (\epsilon+0+1)(\epsilon+0+1)^*\emptyset$	$\emptyset$
$R_{22}^{(2)}$	$\epsilon+0+1+(\epsilon+0+1)(\epsilon+0+1)^*(\epsilon+0+1)$	$(0+1)^*$

$$R = 1^*0(0+1)^*$$

# Eliminação de estados

## □ Eliminação do estado $s$



# Propriedades de fecho

---



- ❑ A classe das linguagens regulares é fechada para a operação
- ❑ Exemplos de operações
  - União, Intersecção e Complemento
  - Diferença
  - Reverso
  - Fecho (\*) e Concatenação
  - Homomorfismo e Homomorfismo Inverso

# Propriedades de decisão das LR

---

- ❑ Como estudar uma linguagem?
  - Infinita  $\rightarrow$  não dá para análise exaustiva
  - Representação finita: DFA, NFA,  $\epsilon$ -NFA, ER  $\rightarrow$  LR
- ❑ Responder a uma questão sobre uma linguagem
  - Encontrar um algoritmo que responda sim ou não
  - Em muitas situações, existem algoritmos para LR mas não existem para as não regulares, mesmo que exista representação finita para algumas delas
- ❑ Três questões
  - A linguagem é vazia?
  - A cadeia  $w$  pertence à linguagem?
  - Duas descrições de linguagens correspondem de facto à mesma?

# Conversão entre representações

---

- ❑ Qual a complexidade dos respectivos algoritmos?
- ❑ NFA  $\rightarrow$  DFA
  - Função no número de estados  $n$  do NFA
  - Cálculo do fecho- $\epsilon$ :  $O(n^3)$
  - Construção de subconjuntos:  $O(2^n)$  (número de estados do DFA)
  - Cálculo de uma transição da função  $\delta$ :  $O(n^3)$
  - Considera-se o alfabeto constante, portanto só influencia a constante escondida na notação  $O(\cdot)$
  - Conversão completa:  $O(n^3 2^n)$
  - Como o número de estados  $s$  do DFA é frequentemente muito menor que exponencial:  $O(n^3 s)$

# Conversão entre representações

---

- ❑ DFA  $\rightarrow$  NFA
  - $O(n)$
- ❑ DFA  $\rightarrow$  ER
  - Algoritmo de introdução sucessiva de estados:  $n$  passos
  - Em cada passo temos  $n^2$  expressões e uma expressão é construída à custa de 4 do passo anterior:  $O(n^3 4^n)$
  - No caso NFA  $\rightarrow$  ER, se se começar por converter primeiro para DFA obtém-se um algoritmo duplamente exponencial:  $O(n^3 4^{n^{32^n}})$

# A linguagem é vazia?

---

- ❑ Resposta:  $L = \emptyset$  é vazia; as outras linguagens não.
- ❑ A questão é mais interessante se  $L$  for representada por uma expressão regular ou por um autômato
- ❑ Autômato
  - Questão resume-se à acessibilidade no respectivo grafo: se nenhum estado final for acessível a partir do inicial, a resposta é positiva
  - Algoritmo de complexidade proporcional ao número de arcos  $O(n^2)$
- ❑ Expressão regular (comprimento  $n$ )
  - Converter para  $\epsilon$ -NFA, resultado  $O(n)$  estados, algoritmo  $O(n)$ , ou
  - Inspeccionar a expressão regular (no caso de conter  $\emptyset$ )

# w pertence à linguagem?

---

- ❑ L representada por um autómato
  - w é sempre explícita
  - DFA
    - ❑ Simular o processamento da cadeia; sim, se terminar num estado de aceitação;  $O(|w|)$
  - NFA,  $\epsilon$ -NFA
    - ❑ Converter para DFA e aplicar método anterior; algoritmo pode ser exponencial no tamanho da representação
    - ❑ Mais simples e mais eficiente simular o NFA directamente, mantendo o conjunto dos s estados em que o autómato pode ficar em cada transição  $O(|w|s^2)$
- ❑ L representada por expressão regular de comprimento s
  - Converter para  $\epsilon$ -NFA com até  $2s$  estados em tempo  $O(s)$  e aplicar o método anterior

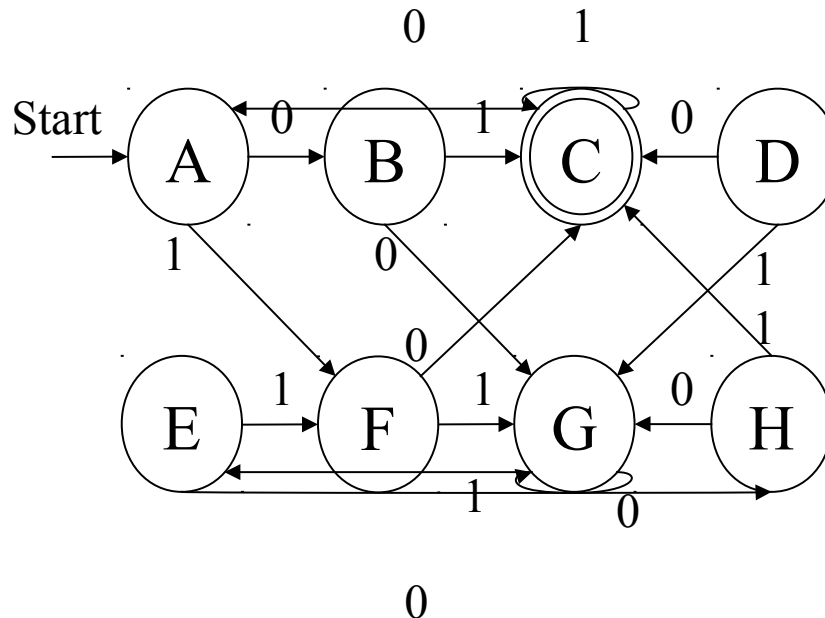


# $L_1$ e $L_2$ são equivalentes?

---

- ❑ Duas descrições de LR são equivalentes se definirem a mesma linguagem
  - Pode encontrar-se uma representação mínima, única a menos de renomeação dos estados
- ❑ Equivalência de estados de um DFA
  - Dois estados  $p$  e  $q$  são equivalentes se para todas as cadeias  $w$   $\delta^*(p,w)$  é um estado de aceitação se e só se  $\delta^*(q,w)$  também o for
    - ❑ Não se consegue distinguir  $p$  e  $q$  só a partir do resultado de aceitação ou não de quaisquer cadeias
    - ❑ Não se exige que  $\delta^*(p,w)$  e  $\delta^*(q,w)$  sejam o mesmo estado mas apenas que sejam ambos de aceitação ou ambos de não aceitação
  - Se dois estados não forem equivalentes, são distinguíveis
    - ❑ Há pelo menos uma cadeia em que um de  $\delta^*(p,w)$  e  $\delta^*(q,w)$  é de aceitação e o outro não

# Equivalência de estados



- ❑  $\delta^*(C, \epsilon) = C$  é de aceitação e  $\delta^*(G, \epsilon)$  não  $\rightarrow$  C e G não são equivalentes
- ❑ A e G:  $\epsilon, 0, 1$  não permitem distinguir, mas  $\delta^*(A, 01) = C$  e  $\delta^*(G, 01) = E$  sim
- ❑ A e E: nenhum é de aceitação; 1 leva ambos para F, portanto  $w=1x$  não permite distinguir; 0 também não;  $\delta^*(A, 00) = G = \delta^*(E, 00)$ ;  $\delta^*(A, 01) = C = \delta^*(E, 01)$ ; portanto A e E são equivalentes

# Algoritmo de preenchimento de tabela

---

- Dado um DFA  $A=(Q,\Sigma,\delta,q_0,F)$  encontrar estados equivalentes
  - Base: se  $p$  for de aceitação e  $q$  não, o par  $\{p,q\}$  é distinguível
  - Indução: sejam  $p$  e  $q$  estados tais que, para um símbolo  $a$ ,  $r=\delta^*(p,a)$  e  $s=\delta^*(q,a)$  são distinguíveis; então  $\{p,q\}$  é distinguível

B	X						
C	X	X					
D	X	X	X				
E		X	X	X			
F	X	X	X		X		
G	X	X	X	X	X	X	
H	X		X	X	X	X	X
	A	B	C	D	E	F	G

# Trabalho

---

Apresentar uma implementação que utilize de recursos de expressões regulares.