

Linguagens Formais e Autômatos (LFA)

Aula de 14/08/2013

Linguagens e Autômatos
Exercícios com JFLAP e Ruby

Retomando o sistema 'pq' da aula anterior

Um sistema com 3 representações apenas: "p", "q" e "-".
É capaz de expressar infinitas "verdades" através das seguintes 2 regras:

Base, fato, axioma

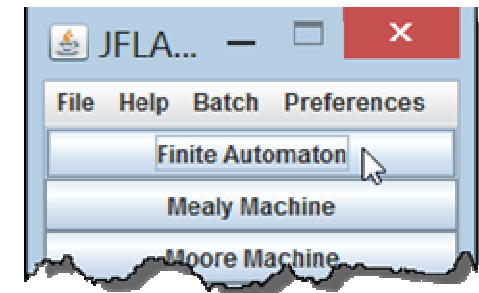
1. **É verdade** que 'X,p,-,q,X,-' para todo X igual a uma sequência de hífens (por exemplo, '-', '--', '---', ...).

1. **Se** for verdade que X,p,Y,q,Z **então** é verdade que 'X,p,Y,-,q,Z,-'

Regra de produção
ou inferência.

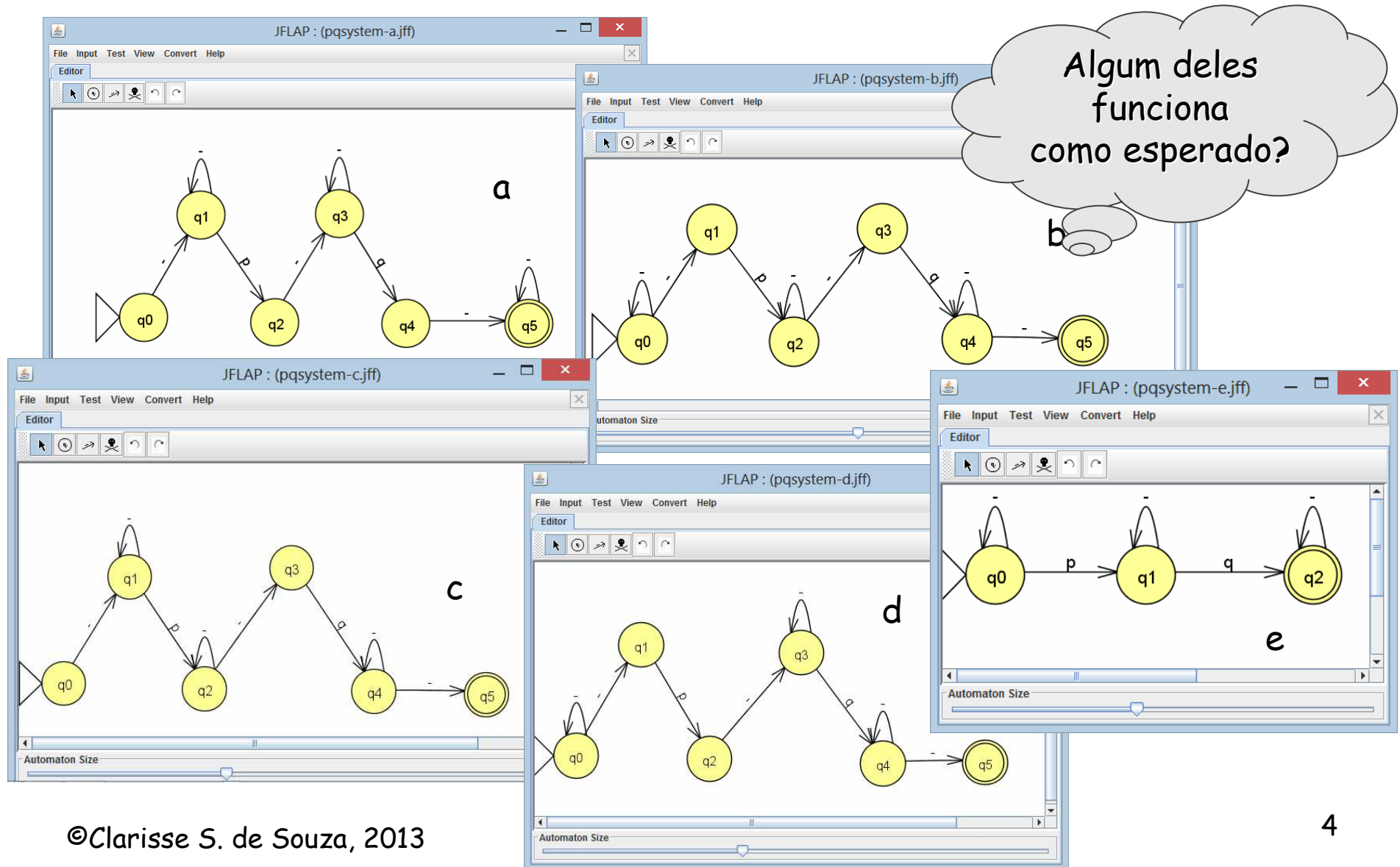
Usemos o construtor "Finite Automaton" do JFLAP

Gostaríamos de ver se dá para construir um autômato que reconheça corretamente cadeias que pertencem e não pertencem ao "sistema pq".

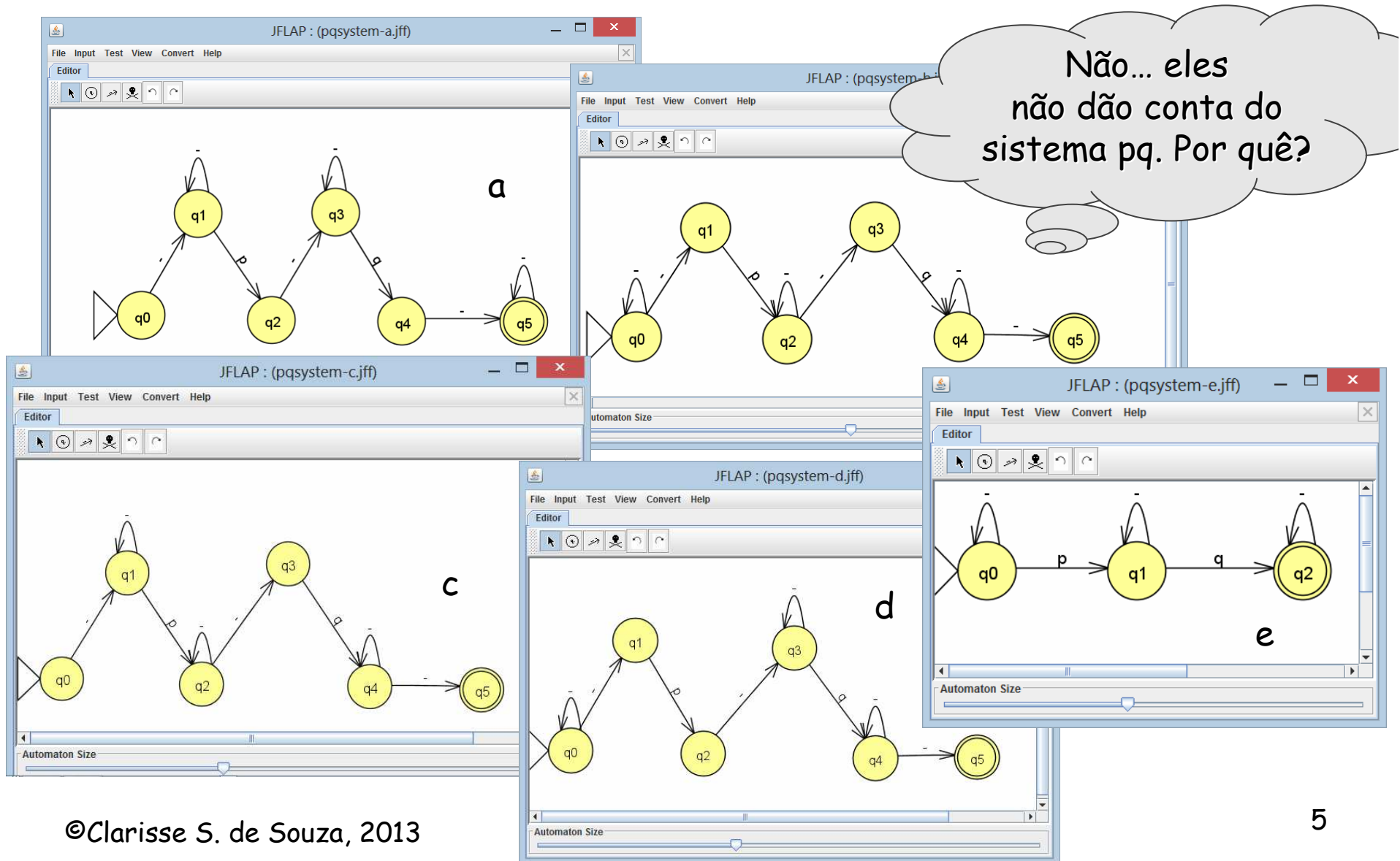


Exemplos de cadeias e resultados esperados

-p-q--	OK	
--p-q---	OK	
p-q-		hmmm...
--p---q-	não OK	
-p-q-----	não OK	
pq--		não OK
--pq--	não OK, mas...	
-----	não OK	
--p-q	não OK	



Não... eles não dão conta do sistema pq. Por quê?



©Clarisse S. de Souza, 2013

Autômatos finitos não dão conta do sistema pq

Porém, examinemos de perto como cada um deles processa o conjunto de cadeias exemplo.

Qual o seu comportamento para cadeias que aceitam?

Qual o seu comportamento para cadeias que não aceitam?

O QUE JÁ DÁ PARA PERCEBER?

Implementações em Ruby

Como poderíamos “programar” autômatos do tipo que estamos examinando?

Vejamos exemplos em Ruby.

Introdução à linguagem Ruby

- <http://www.ruby-lang.org/pt/>
- Criada por Yukihiro Matsumoto (1995)
- Dinâmica, interpretada
- Orientada a objetos
- Multiplataforma
- Possui características de linguagens imperativas e funcionais
- Inspirada em Perl, Smalltalk, Eiffel, Lisp

Por que Ruby ?

- Usada no livro-texto (Ramos, Neto e Vega)
- Fácil de aprender e usar
- Boa expressividade para os modelos usados no curso
- Popularidade

Visão geral de Ruby

```
# hello.rb  
puts "Hello, world !"
```

Strings e operações básicas

```
puts "Hello " + "world"      # "Hello world"  
puts "Hello " * 2            # "Hello Hello "  
puts "Hello".reverse()       # "olleH"  
puts "Hello".length()        # 5  
puts :Hello                   # "Hello" (Symbol)
```

Conversões entre tipos

```
puts 'len=' + 'abc'.length()           # erro !
```

```
puts 'len=' + 'abc'.length().to_s()     # len=3
```

```
puts '1' + 1                           # erro !
```

```
puts '1'.to_i + 1                       # 2
```

```
puts 1.to_s + 'a'                       # 1a
```

Arrays

```
a = [ 1, 'hi', 3.14, 1, 2, [4, 5] ]
```

```
puts a[2]           # 3.14
```

```
puts a.reverse      # [[4, 5], 2, 1, 3.14, 'hi', 1]
```

```
puts a.flatten.uniq # [1, 'hi', 3.14, 2, 4, 5]
```

```
a.each do |x| puts x end # imprime elementos de a
```

Hashes

```
hash = { :inf1626 => 'LFA', :inf1005 => 'Programacao I' }  
puts hash[ :inf1626 ]      # LFA
```

```
hash.each_pair do |key, value|  
  puts "#{key} : #{value}"  
end
```

```
hash.delete :inf1005  
puts hash.inspect      # { :inf1626=>"LFA" }
```

Classes

```
class Automato
  attr_accessor :nome

  def initialize( nome )
    @nome = nome
  end
end

a = Automato.new( 'aut01' )

puts a.nome      # aut01
```

Conjuntos

A = [1 , 2 , 3]

B = [1, 2, 3, 4, 5]

pertinência

puts A.include?(2) #true

puts A.include?(5) #false

união

puts (A | B).inspect # [1, 2, 3, 4, 5]

#interseção

puts (A & B).inspect # [1, 2, 3]

#diferença

puts (B - A).inspect # [4, 5]

#cardinalidade

puts A.size() # 3

Representação de autômatos finitos

Configuração:

- Estado inicial
- Conjunto de estados de aceitação
- Transições de estados

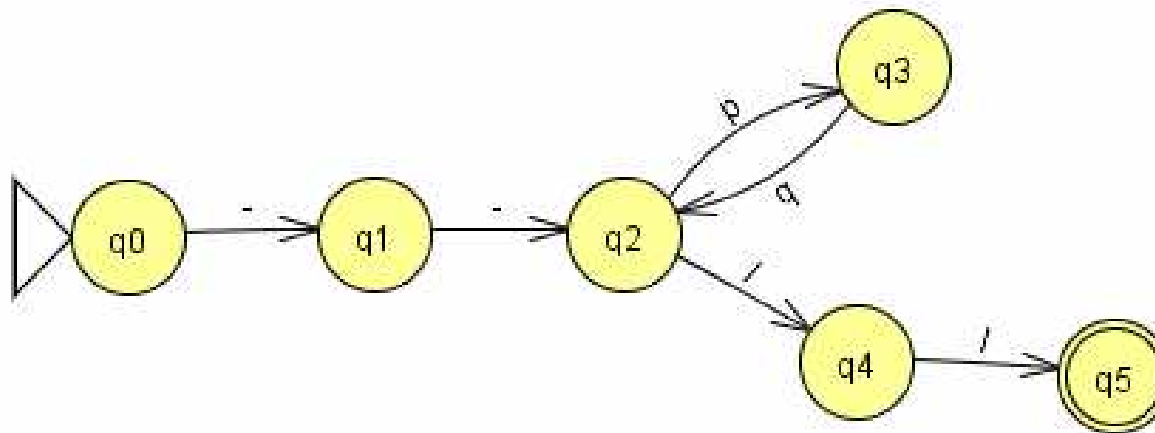
Execução:

- Início: cadeia de entrada
- Fim: estado final e cadeia de entrada resultante
- Reconhecimento: estado final de aceitação e toda a cadeia de entrada consumida

Primeiro exemplo : aut01.rb

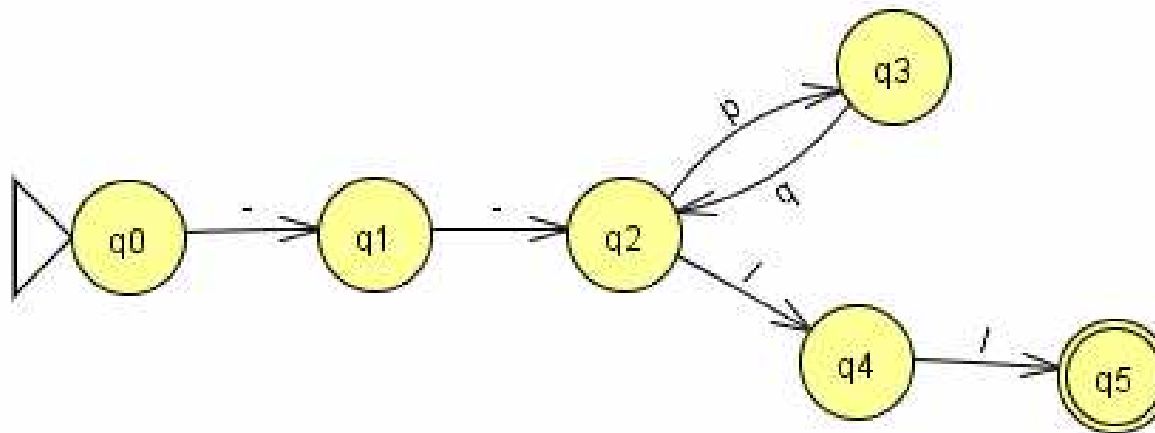
Baseado no exemplo da aula 1 (aut01.jff)

- Primeira abordagem



Segundo exemplo (aut01-v2.rb)

Implementação parametrizada do mesmo autômato



Recursos adicionais

Documentação de Ruby

<http://www.ruby-doc.org/>

Try Ruby

<http://tryruby.org>

Ruby from other languages

<http://www.ruby-lang.org/en/documentation/ruby-from-other-languages/>