

Linguagens Formais e Autômatos (LFA)

Aula de 30/10/2013

Autômatos de Pilha
Modelo Conceitual; JFLAP
Modelos de Implementação: Ruby

Definição e Modelo Conceitual

No JFLAP autômatos de pilha não determinísticos (nondeterministic pushdown automaton, ou NPDA) como a tupla $(Q, \Sigma, \Gamma, \delta, q_s, Z, F)$ onde:

Q é um conjunto finito de estados $\{q_i \mid i \text{ é um inteiro não negativo}\}$
 Σ é um conjunto finito de símbolos que constituem o **alfabeto de entrada**

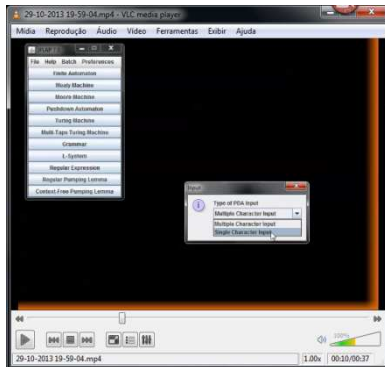
Γ é um conjunto finito de símbolos que constituem o **alfabeto da pilha**
 δ é a **função de transição**, $\delta : Q \times \Sigma^* \times \Gamma^* \rightarrow$ subconjuntos finitos de $Q \times \Gamma^*$
 q_s (membro de Q) é o **estado inicial**

Z é o **símbolo inicial da pilha** (sempre um Z , maiúsculo, no JFLAP)
 F (subconjunto de Q) é o **conjunto de estados finais**

Esta definição dá conta de autômatos de pilha determinísticos (PDA), que nada mais são do que NPDA's onde, para cada elemento de $\delta = Q \times \Sigma^* \times \Gamma^* \rightarrow$ subconjuntos finitos de $Q \times \Gamma^*$, a cardinalidade de $Q \times \Gamma^*$ é 1 (só há **um** estado para o qual a transição leva).

Construindo um NPDA com o JFLAP

No Tutorial do JFLAP pode-se ver como se construir NPDA's. Veja a ilustração no vídeo que acompanha o material da aula.

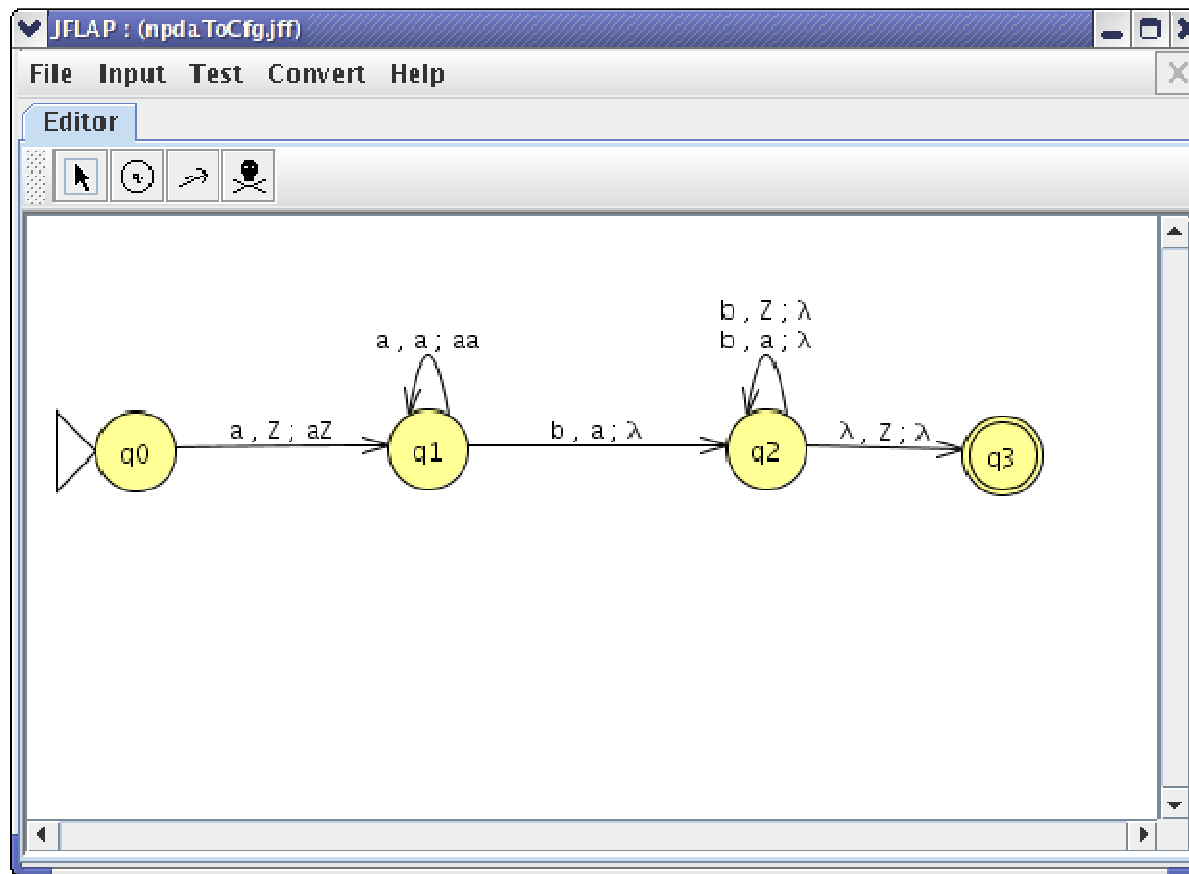


veja "npda-video1.mp4"

No exemplo, o NPDA é construído para a linguagem $L = \{a^n b^n : n > 0\}$.

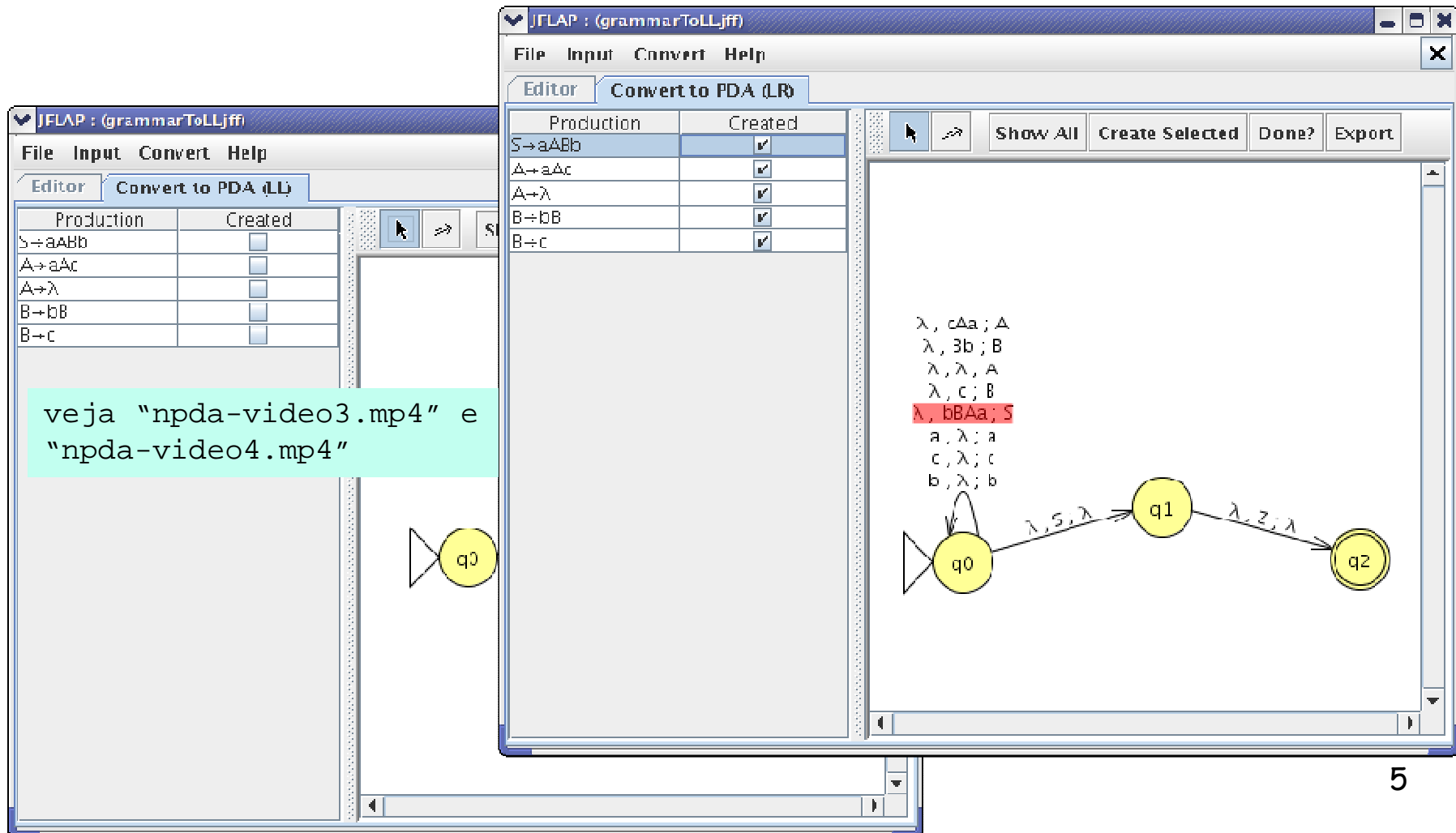
A opção dos tutores é `push` um "a" para a pilha a cada vez que for lido um "a" na cadeia de entrada e, depois, `pop` um "a" para fora da pilha a cada vez que for lido um "b" na cadeia de entrada.

Convertendo um NPDA para CFG com o JFLAP



veja "npda-video2.mp4"

Convertendo LL(k) e LR(k) para PDA no JFLAP



veja "npda-video3.mp4" e "npda-video4.mp4"

JFLAP : (grammarToLLjff)

File Input Convert Help

Editor Convert to PDA (LL)

Production	Created
$S \rightarrow aABb$	<input type="checkbox"/>
$A \rightarrow aAc$	<input type="checkbox"/>
$A \rightarrow \lambda$	<input type="checkbox"/>
$B \rightarrow bB$	<input type="checkbox"/>
$B \rightarrow c$	<input type="checkbox"/>

JFLAP : (grammarToLLjff)

File Input Convert Help

Editor Convert to PDA (LR)

Production	Created
$S \rightarrow aABb$	<input checked="" type="checkbox"/>
$A \rightarrow aAc$	<input checked="" type="checkbox"/>
$A \rightarrow \lambda$	<input checked="" type="checkbox"/>
$B \rightarrow bB$	<input checked="" type="checkbox"/>
$B \rightarrow c$	<input checked="" type="checkbox"/>

Show All Create Selected Done? Export

$\lambda, cAa; A$
 $\lambda, 3b; B$
 λ, λ, A
 $\lambda, c; B$
 $\lambda, bBAa; S$
 $a, \lambda; a$
 $c, \lambda; c$
 $b, \lambda; b$

$q_0 \xrightarrow{\lambda, S, \lambda} q_1 \xrightarrow{\lambda, Z, \lambda} q_2$

Implementação de Autômatos de Pilha em Ruby (livro-texto, p.411-430)

- Novas classes:
 - AutomatoPilha
 - AutomatoPilhaDeterministico
 - AutomatoPilhaNaoDeterministico
 - ConsultaAPD
 - MovimentacaoAPD
 - MovimentacaoAPND
 - Pilha
 - ReconhecedorAPD
 - ReconhecedorAPND

Arquivo apd/ReconhecedorAPD.rb

```
class ReconhecedorAPD < Reconhecedor
  def instanciarAutomato( estadoInicial, estadosFinais )
    @automato = AutomatoPilhaDeterministico.new(
      estadoInicial, estadosFinais )
  end

  def condicaoDoAutomato?( automato )
    return ( automato.consulta.atingiuEOF?( ) &&
      automato.consulta.pilhaVazia?( ) )
  end
end
```

Obs: este método implementa a aceitação por pilha vazia. Se comentado, a aceitação é por estado final (implementada na classe base)

Arquivo apd/AutomatoPilha.rb

```
class AutomatoPilha < Automato
  attr_accessor :pilha

  def instanciarEstruturaEspecificica()
    @pilha = Pilha.new()
  end

  def adicionarTransicao( transicao )
    @transicoes.update( transicao )
  end

  def instanciarEntrada()
    @entrada = FitaLimitada.new()
  end

  def instanciarConsulta()
    @consulta = ConsultaAPD.new( self )
  end
end
```


Arquivo apd/AutomatoPilhaDeterministico.rb

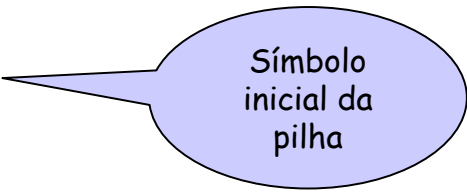
```
class AutomatoPilhaDeterministico < AutomatoPilha
  def instanciarMovimentacao()
    @movimentacao = MovimentacaoAPD.new( self )
  end
end
```

Exemplo de transição:

```
{ ['q0', 'b', 'Z0'] => ['q1', ['Z0']] }
```

Arquivo apd/Pilha.rb

```
class Pilha
  def initialize()
    @conteudo = ["Z0"]
  end
  def pop()
    return @conteudo.pop
  end
  def top()
    return @conteudo.last
  end
  def push( lista )
    @conteudo += lista
  end
  def vazia?()
    return @conteudo == []
  end
  def configuracao?()
    return "(" + @conteudo.inspect() + ")"
  end
  def clonar()
    return Clonagem.new().clonar( self )
  end
end
```




Símbolo
inicial da
pilha

Arquivo apd/servico/MovimentacaoAPD.rb (1)

```
class MovimentacaoAPD < MovimentacaoDeterministica

  def calcularOndaDeClones()
    ondaDeClones = {}
    e = @automato.consulta.estadoCorrente?()
    s = @automato.entrada.ler()
    z = @automato.pilha.top()
    @automato.transicoes.each do |condicaoEsperada, instrucao|
      #tentativa de transicao com consumo de entrada
      if( condicaoEsperada == [ e,s,z ] )
        clone = @automato.clonar()
        ondaDeClones[ clone ] = (instrucao << "A")
      #tentativa de transicao sem consumo de entrada
      elsif( condicaoEsperada == [ e,"",z ] )
        clone = @automato.clonar()
        ondaDeClones[ clone ] = instrucao
      end
    end
    return ondaDeClones
  end
end
```



Indica que
deve avançar
na fita de
entrada

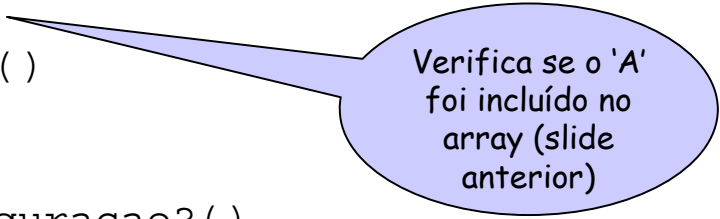
Arquivo apd/servico/MovimentacaoAPD.rb (2)

```
def mover( instrucao )
  @automato.estadoCorrente = instrucao[0]
  @automato.pilha.pop()
  @automato.pilha.push( instrucao[1] )

  if( instrucao.size() > 2 )
    @automato.entrada.avancar()
  end

  puts @automato.consulta.configuracao?()
end

end
```



Verifica se o 'A'
foi incluído no
array (slide
anterior)

Arquivo apd/CasoUso1.rb

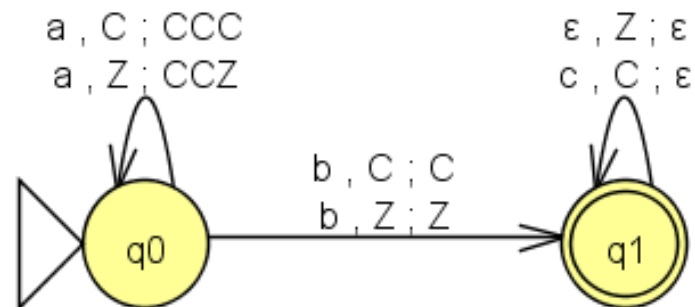
```

rpd = ReconhecedorAPD.new( 'q0', ['q1'] )

rpd.automato.adicionarTransicao({ ['q0','a','Z0'] => ['q0',['Z0','C','C']] })
rpd.automato.adicionarTransicao({ ['q0','a','C']   => ['q0',['C','C','C']] })
rpd.automato.adicionarTransicao({ ['q0','b','Z0'] => ['q1',['Z0']] })
rpd.automato.adicionarTransicao({ ['q0','b','C']   => ['q1',['C']] })
rpd.automato.adicionarTransicao({ ['q1','c','C']   => ['q1',[]] })
rpd.automato.adicionarTransicao({ ['q1','','Z0']  => ['q1',[]] })

rpd.iniciar( 'aabccc' )
automatos = rpd.analisar()
puts rpd.reconheceu?()

```




Arquivo apnd/ReconhecedorAPND.rb

```
class ReconhecedorAPND < Reconhecedor

  def instanciarAutomato( estadoInicial, estadosFinais )
    @automato = AutomatoPilhaNaoDeterministico.new(
      estadoInicial, estadosFinais )
  end

  def condicaoDoAutomato?(automato)
    return (automato.consulta.atingiuEOF?( ) &&
      automato.consulta.pilhaVazia?( ))
  end

end
```



Assim como no
APD, define o
critério de
aceitação

Arquivo apnd/AutomatoPilhaNaoDeterministico.rb

```
class AutomatoPilhaNaoDeterministico < AutomatoPilha
  def instanciarMovimentacao()
    @movimentacao = MovimentacaoAPND.new(self)
  end
end
```

Exemplo de transição:

```
{ ['q0', 'a', 'Z0'] => [ ['q1', ['Z0', 'C', 'C']],
                             ['q2', ['Z0', 'C']] ] }
```

Arquivo apnd/servico/MovimentacaoAPND.rb (1)

```
class MovimentacaoAPND < MovimentacaoNaoDeterministica
  def calcularOndaDeClones()
    ondaDeClones = {}
    e = @automato.consulta.estadoCorrente?()
    s = @automato.entrada.ler()
    z = @automato.pilha.top()
    @automato.transicoes.each do |condicaoEsperada, instrucoes|
      #tentativa de transicao com consumo de entrada
      if( condicaoEsperada == [ e, s, z ] )
        instrucoes.each do |instrucao|
          clone = @automato.clonar()
          ondaDeClones[ clone ] = (instrucao << "A")
        end
      #tentativa de transicao sem consumo de entrada
      elsif( condicaoEsperada == [ e, "", z ] )
        instrucoes.each do |instrucao|
          clone = @automato.clonar()
          ondaDeClones[ clone ] = instrucao
        end
      end
    end
    return ondaDeClones
  end
end
```

Diferença em
relação ao APD (não
determinismo)

Arquivo `apnd/servico/MovimentacaoAPND.rb` (2)

```
def mover( instrucao )

  @automato.estadoCorrente = instrucao[0]
  @automato.pilha.pop()
  @automato.pilha.push( instrucao[1] )

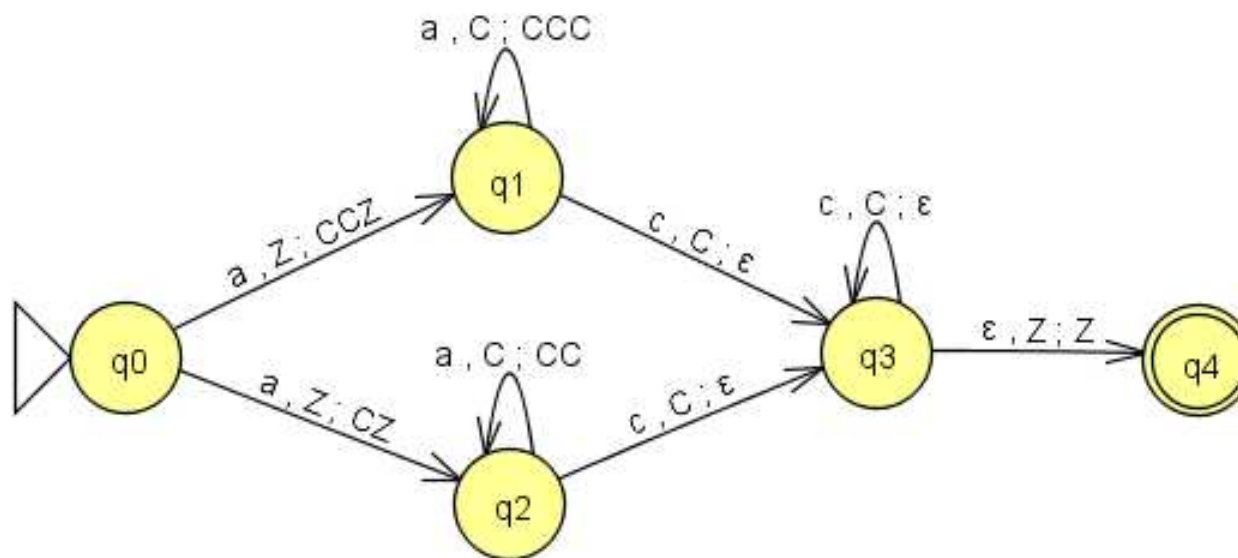
  if( instrucao.size() > 2 )
    @automato.entrada.avancar()
  end

  puts @automato.consulta.configuracao?()
end

end
```

Igual a `MovimentacaoAPD.mover`

Arquivo apnd/CasoUso1.rb



Arquivo apnd/CasoUso1.rb

```

rpnd = ReconhecedorAPND.new( 'q0', ['q4'] )

rpnd.automato.adicionarTransicao({['q0','a','Z0'] =>
  [ ['q1', ['Z0','C','C']], ['q2',['Z0','C']] ]})

rpnd.automato.adicionarTransicao({['q1','a','C'] =>
  [ ['q1', ['C','C','C']] ]})

rpnd.automato.adicionarTransicao({['q1','c','C'] => [ ['q3',[]] ]})
rpnd.automato.adicionarTransicao({['q2','a','C'] => [ ['q2',['C','C']] ]})
rpnd.automato.adicionarTransicao({['q2','c','C'] => [ ['q3',[]] ]})
rpnd.automato.adicionarTransicao({['q3','c','C'] => [ ['q3',[]] ]})
rpnd.automato.adicionarTransicao({['q3','', 'Z0'] => [ ['q4',['Z0']] ]})

rpnd.iniciar( 'ac' )
automatos = rpnd.analisar()
puts rpnd.reconheceu?()

```

