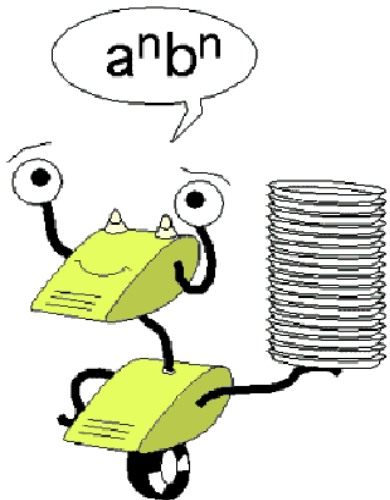


# Autômatos com Pilha

Autômatos com Pilha:  
Reconhecedores de LLCs

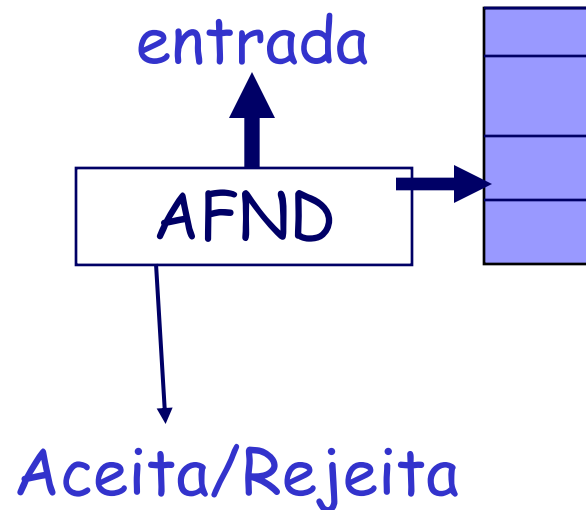


## Autômatos com Pilha (AP)

- Definições alternativas para Linguagens Livres de Contexto
- Extensão de AFND com uma pilha, que pode ser lida, aumentada e diminuída apenas no topo.
- 2 variações de AP:
  - Aceita cadeias por estados de aceitação
  - Aceita cadeias por pilha vazia
- As 2 variações aceitam exatamente as LLC  
(GLC  $\leftrightarrow$  AP)
- AP Determinísticos: aceitam todas as LR e parte das LLC

## AP - Definição Informal

- A pilha permite memorizar uma quantidade infinita de informações. Ao contrário de um computador, que também pode armazenar quantidade arbitrária de informações, o AP tem a restrição do comportamento da pilha (FIFO).



## AP

- O AP decide a mudança de estado baseado no símbolo atual da entrada, no estado atual, e no símbolo do topo da pilha.
- Ele pode, alternativamente, fazer uma transição espontânea, usando  $\epsilon$  como entrada. Em uma transição, o AP:
  - Consome da entrada o símbolo que ele utiliza na transição. Se  $\epsilon$  for usado, nenhum símbolo da entrada é consumido.
  - Vai para um novo estado, que pode ou não ser o mesmo estado anterior.
  - Substitui o símbolo do topo da pilha por qualquer cadeia. A cadeia pode ser  $\epsilon$  (eliminação na pilha); pode ser o mesmo símbolo do topo (nenhuma alteração), ou um ou mais símbolos distintos (eliminação + inserção).

Um AP M é uma sétupla  $(K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  onde:

1.  $K$  é um conjunto finito de estados
2.  $\Sigma$  (sigma) é um alfabeto finito chamado alfabeto de entrada
3.  $\Gamma$  (gama) é um *alfabeto da pilha* finito
4.  $q_0 \in K$  é o estado inicial. A máquina começa nele.
5.  $Z_0 \in \Gamma$  é o símbolo inicial da pilha. Aparece inicialmente na pilha.
6.  $F$  é o conjunto de estados finais  $F \subseteq K$
7.  $\delta$  (delta) é um mapeamento de  
$$K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \text{subconjuntos de } K \times \Gamma^*$$
  
(topo)

O resultado da aplicação de  $\delta$  é um conjunto finito de pares  $(p, \gamma)$  onde  $p$  é o novo estado e  $\gamma$  é a cadeia de símbolos da pilha que substitui o topo da pilha. Se  $\gamma$  é  $\varepsilon$ , o topo é eliminado; senão o topo é substituído por  $\gamma$  de tal forma que o 1º. símbolo de  $\gamma$  é o novo topo.

## Notação gráfica para AP

- O diagrama de transição para APs que aceitam a linguagem por estado final segue:
- Os nós correspondem aos estados do AP
- Existem o estado inicial e os finais
- Um arco rotulado com  $a, X/\alpha$  do estado  $q$  para  $p$  significa que  $\delta(q, a, X)$  contém o par  $(p, \alpha)$  entre os pares.
- Convencionalmente,  $Z_0$  é o símbolo da pilha (no JFLAP é  $Z$ ).
- Leia-se: símbolo, topo\_antigo/novo\_topo

# Linguagem aceita por um AP

1. **Aceitação por Estado Final:** (Similar a AF)  
Conjunto de todas as cadeias para as quais alguma sequência de movimentos faz o AP entrar num estado final  $\rightarrow$  **Linguagem aceita por estado final**,  $L(P) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \alpha); q \in F\}$

2. **Aceitação por Pilha Vazia:** Conjunto de todas as cadeias para as quais alguma sequência de movimentos, a partir do inicial, faz com que a pilha fique vazia  $\rightarrow$  **Linguagem aceita por Pilha vazia**  $N(P) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$

OBS: quando aceitamos por pilha vazia o conjunto de estados finais é irrelevante.

**Os dois métodos acima são equivalentes.**

## Exemplo 1

- Encontre um AP  $M$  que reconheça o conjunto  $L = \{ w \mid w \in \{0,1\}^* \text{ e } w \text{ tem igual número de } 0 \text{ e } 1 \}$ , por pilha vazia.
- Mostre as configurações do AP com a entrada 0101

Dica:

- associar 0 com  $Z$  e 1 com  $U$ ;  $Z, U \in \Gamma$
- "matar" 0 com 1 e 1 com 0
- fazer regra para aceitar cadeia vazia



$$M = (\{q1\}, \{0,1\}, \{B,Z,U\}, \delta, q1, B, \emptyset)$$

1.  $\delta(q1, 0, B) = \{(q1, ZB)\}$

*empilha primeiro 0*

2.  $\delta(q1, 1, B) = \{(q1, UB)\}$

*empilha primeiro 1*

3.  $\delta(q1, 0, Z) = \{(q1, ZZ)\}$

*empilha 0 consecutivos*

4.  $\delta(q1, 1, U) = \{(q1, UU)\}$

*empilha 1 consecutivos*

5.  $\delta(q1, 1, Z) = \{(q1, \epsilon)\}$

*desempilha se 1 com 0*

6.  $\delta(q1, 0, U) = \{(q1, \epsilon)\}$

*desempilha se 0 com 1*

7.  $\delta(q1, \epsilon, B) = \{(q1, \epsilon)\}$

*desempilha no fim da cadeia*

AP para  $L =$  cadeias com igual número de 0s e 1s  
(por pilha vazia)

0, B/ZB

1, B/1B

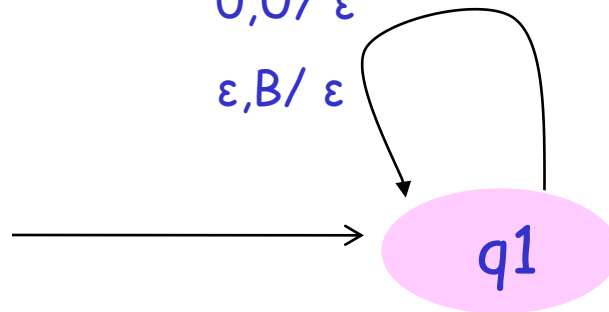
0, Z/ZZ

1, U/UU

1, Z/  $\epsilon$

0, U/  $\epsilon$

$\epsilon$ , B/  $\epsilon$



Repare que o AP é Determinístico

# Configurações para 0101

Entrada	Configuração
	$(q1, B)$
0	$(q1, ZB)$
01	$(q1, B)$
010	$(q1, ZB)$
0101	$(q1, B)$
$\varepsilon$	$(q1, \varepsilon)$

## Exemplo 2

- Encontre um AP  $M$  que reconheça o conjunto  $L = \{ 0^n 1^{2n} \mid n > 0 \}$  por pilha vazia.
- Ideia: para cada 0 lido, empilhar a subcadeia 11. Para cada 1 lido, desempilhar um 1 da pilha. Se ao acabar a cadeia a pilha estiver vazia, a cadeia é aceita.

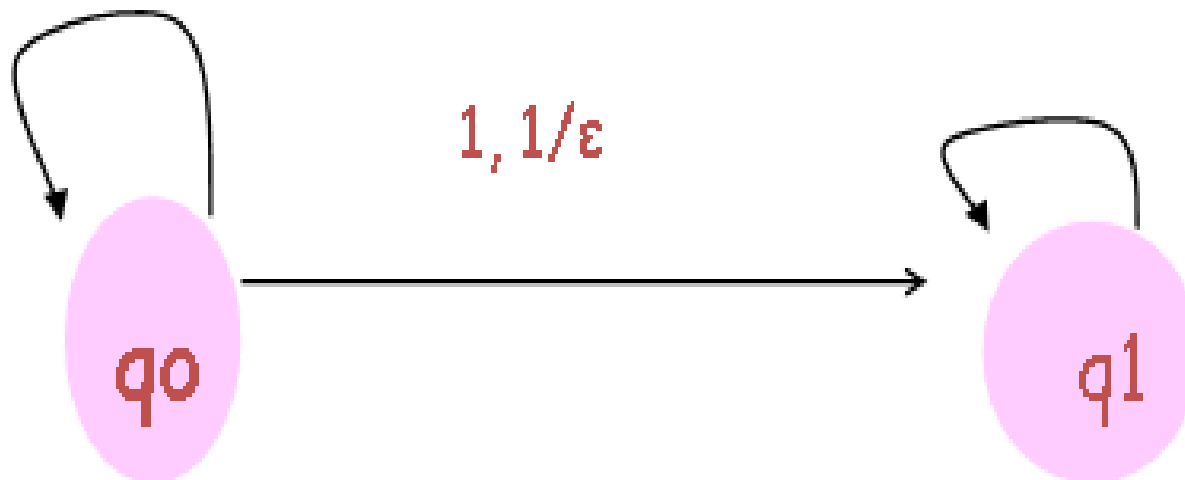
$AP = (\{q_0, q_1\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0)$

$0, Z_0 / 11Z_0$

$\epsilon, Z_0 / \epsilon$

$0, 1 / 111$

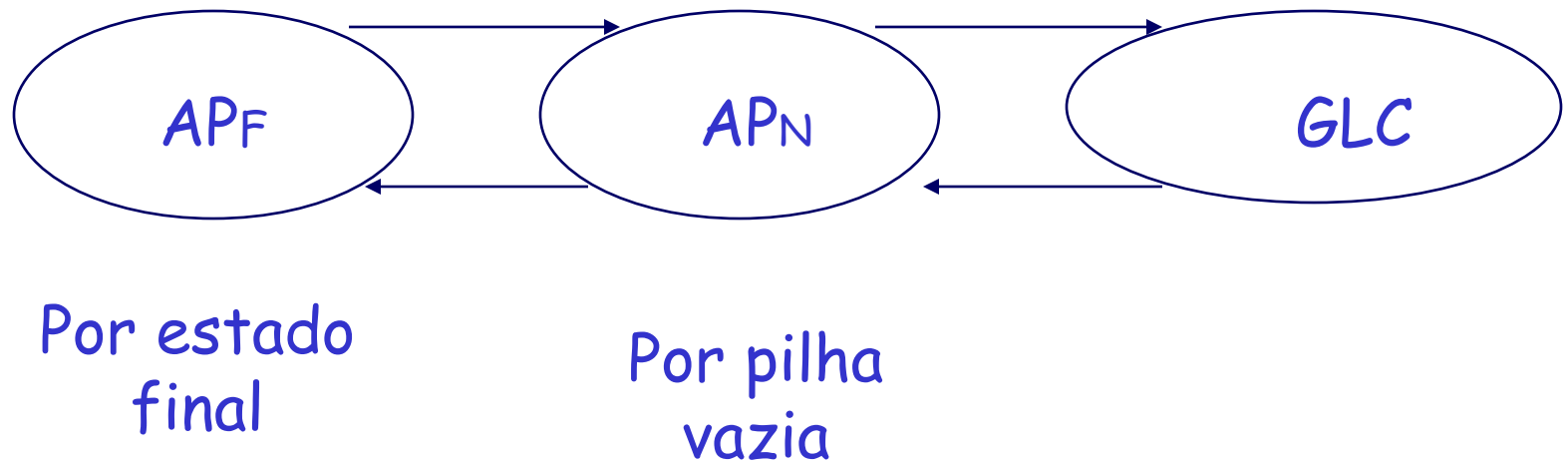
$1, 1 / \epsilon$



Repare que  $L$  é LC, mas não é Regular, e o AP é Determinístico

# Equivalência de AP e Gramática Livre de Contexto

São equivalentes:



## AP Determinísticos (APD)

- Por definição, os AP podem ser não-determinísticos;
- A classe de AP determinísticos são especiais: embora limitada, a classe de linguagens reconhecidas por eles é interessante no contexto de linguagens de programação.

## APD & LLC

- APD não reconhecem todas as LLCs.
- Por exemplo, não existe um APD que reconheça  $L_{wwr} = \{ww^r \mid w \text{ está em } (0+1)^*\}$  - palíndromos de comprimento par sobre  $\{0,1\}$ .

Mas, se inserirmos um "marcador de centro"  $c$ , tornamos a linguagem reconhecível por um APD:

$$L_{wcr} = \{wcw^r \mid w \text{ está em } (0+1)^*\}$$

**Estratégia:** empilhar 0s e 1s até encontrar  $c$ . Então muda de estado em que compara cada novo símbolo com o topo da pilha, desempilhando se forem iguais, até encontrar  $Z_0$  no topo; caso contrário, para num estado não final.



# APND de $Lww^r$

0,  $Z_0/0 Z_0$

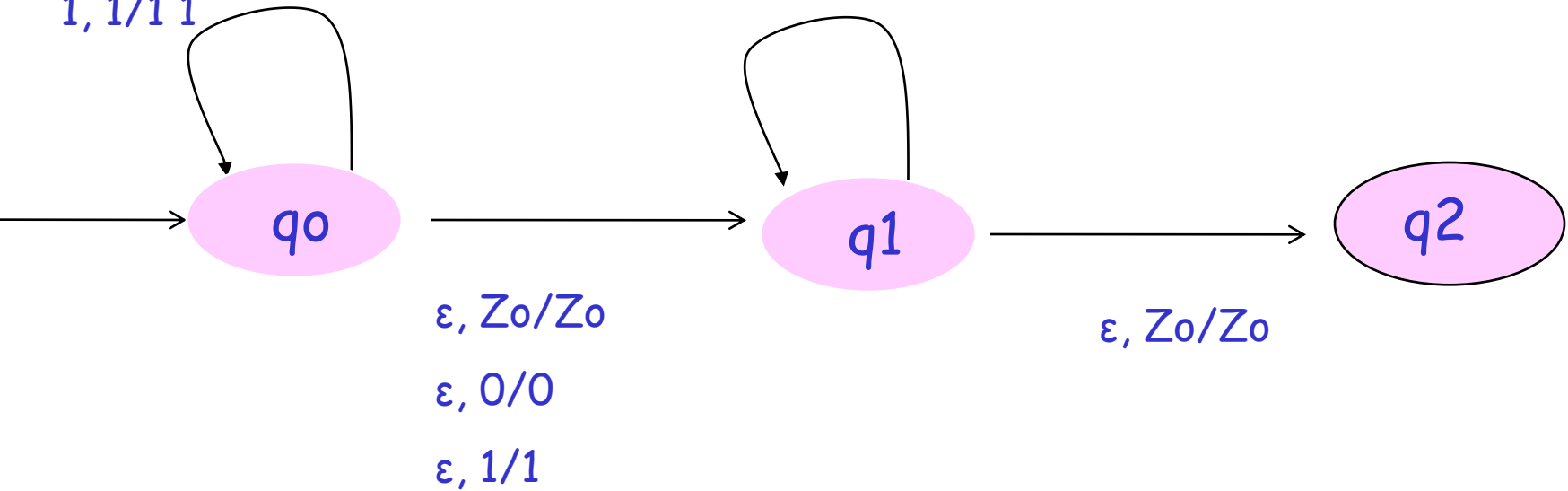
1,  $Z_0/1 Z_0$

0, 0/0 0

0, 1/0 1

1, 0/1 0

1, 1/1 1



APD para  $L_{wcw^r} = \{wcw^r \mid w \text{ está em } (0+1)^*\}$

0, Z0/0 Z0

1, Z0/1 Z0

0, 0/0 0

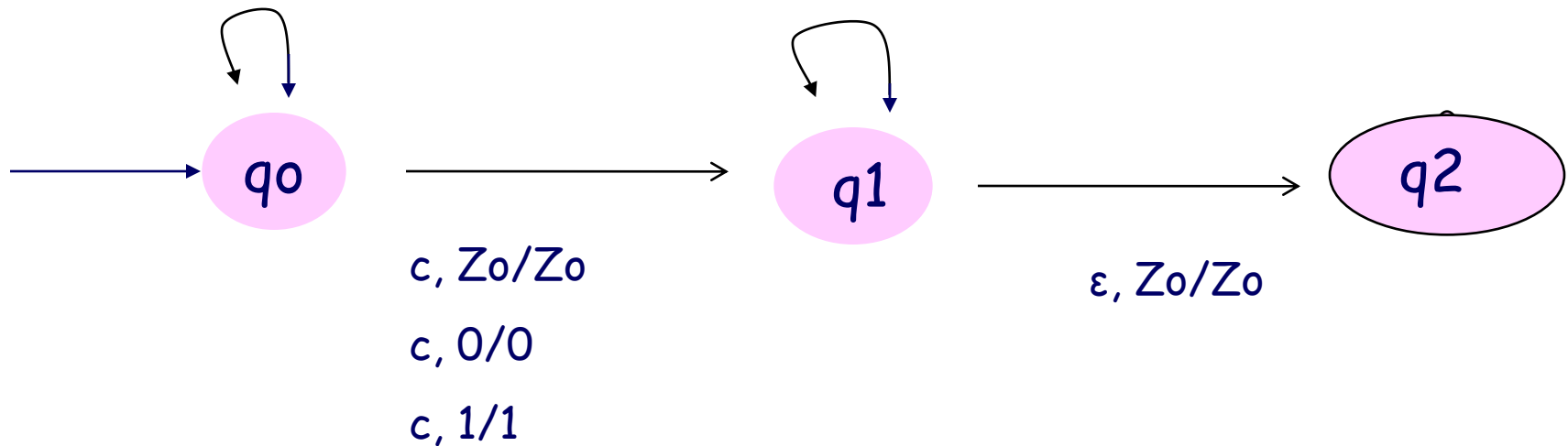
0, 1/0 1

1, 0/1 0

1, 1/1 1

0, 0/ $\epsilon$

1, 1/ $\epsilon$



# APD e Linguagens Regulares

Os APD aceitam uma classe de linguagens que estão entre as LR e as LLC.

- **Teorema 1:** Se  $L$  é uma LR, então  $L=L(P)$  para algum APD  $P$ .
- Mas os APD tb aceitam LLC que não são Regulares (vide exemplo 2 anterior)

## APD por estado final e as LLC

- As linguagens aceitas por APDs incluem as LRs, mas também incluem outras LLCs.
- Ex.  $L_{wcw}^r$
- Ou seja, há LLCs para as quais não existe um APD que as reconhece (há apenas APND).
- Ex.  $L_{ww}^r$

## Observação

- D. Knuth definiu, em 1965, as gramáticas  $LR(k)$ , uma subclasse das  $GLC$  que geram exatamente as linguagens reconhecidas por APDs.
- As gramáticas  $LR(k)$  formam a base para o YACC (*yet another compiler compiler*) do Unix, que gera um programa em C, que é um analisador sintático para a linguagem definida por uma  $LR(k)$ , dada como entrada.

## APD e Gramáticas Ambíguas

- **Teorema:** Se  $L = N(P)$  para algum APD  $P$  (por pilha vazia), então  $L$  tem uma GLC não-ambígua.
- **Teorema:** Se  $L = L(P)$  para algum APD  $P$  (por estado final) então  $L$  tem uma GLC não-ambígua.

*(isso nos indica que as linguagens inerentemente ambíguas só podem ser reconhecidas por APND)*

# Gramáticas e reconhecedores

Linguagens/Gramáticas	Reconhecedores
Rec. Enumerável	Máquina de Turing
Sensível ao contexto	Máquina de Turing com memória limitada ou Autômato Linearmente Limitado
Livre de contexto	Autômato a pilha
Regular	Autômato finito

Gramáticas Sensíveis ao Contexto  
(GSC)

Linguagens Sensíveis ao Contexto  
(LSC)

Autômatos Linearmente  
Limitados  
(ALL)



# Autômatos Linearmente Limitados -ALL: Reconhecedores de LLCs

Para que um autômato seja capaz de reconhecer LSCs que não sejam LLCs, ele deve ser uma Máquina de Turing, mas pode ter a seguinte restrição:

- O espaço de fita ocupado é **linear** em relação ao comprimento da cadeia de entrada a ser reconhecida.

Assim, se  $|w|=n$ , então  $O(n)$  células da fita são utilizadas no reconhecimento de  $w$ .

Na verdade, é possível mostrar que apenas  $n$  células são necessárias.

# Ex. Um ALL para reconhecer a LSC

$$L = \{ a^n b^n c^n \mid n \geq 0 \}$$

Exemplos:

*Pertence à L:*

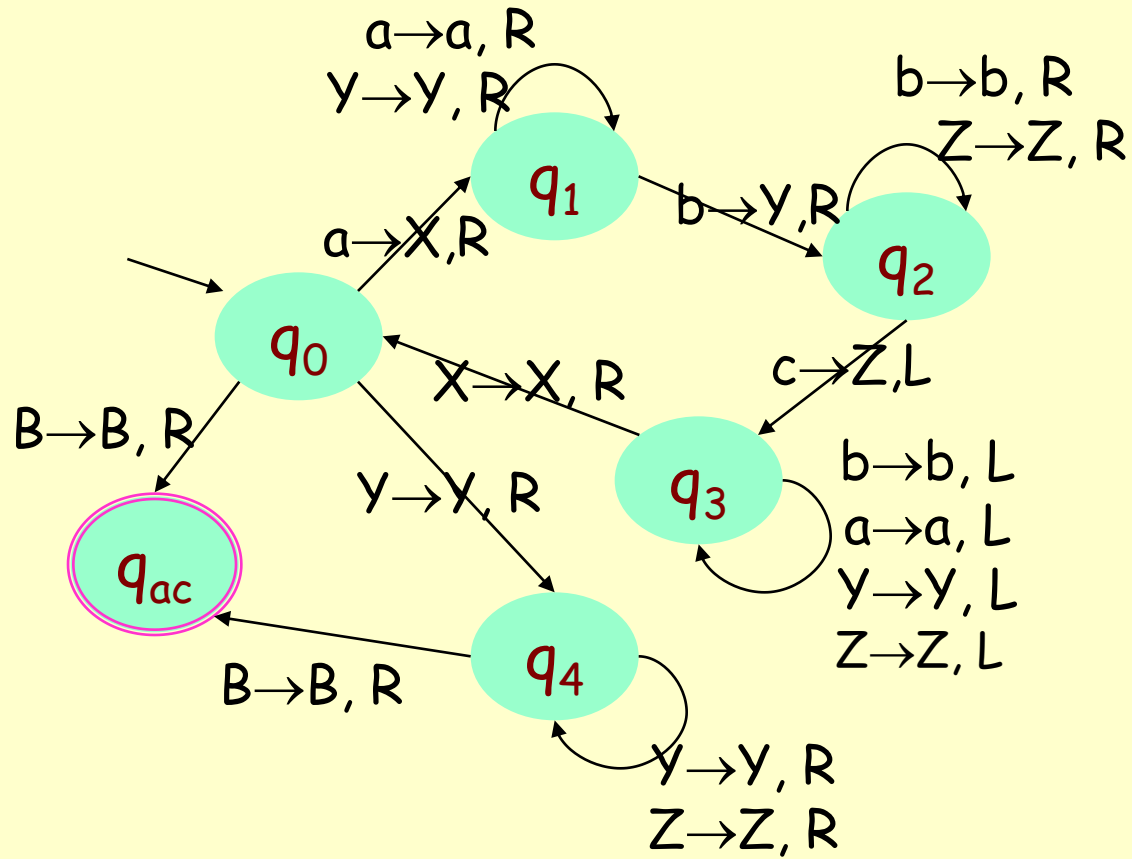
aaabbbccc

*Não Pertence à L:*

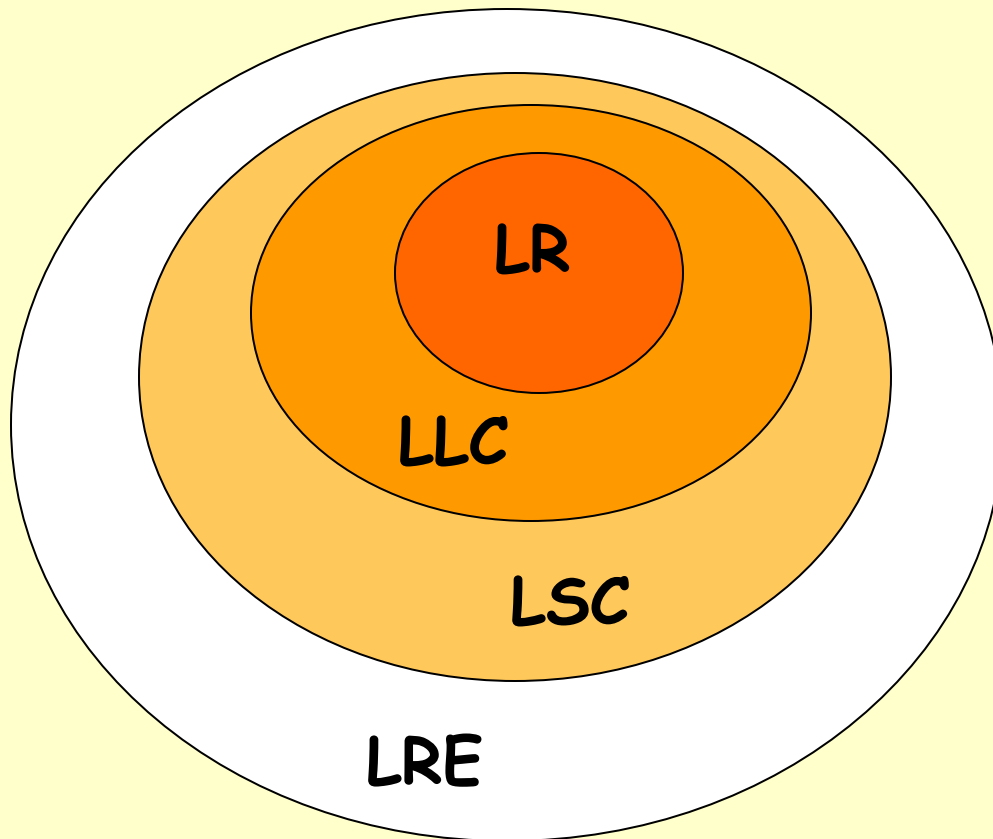
aaabbbcccc

Ideia: em cada passo, reconhecer um a, um b e um c, substituindo-os por X, Y e Z, respectivamente.

1.  $Q = \{q_0, q_1, q_2, q_3, q_4, q_{ac}\}$
2.  $\Sigma = \{a, b, c\}$
3.  $\Gamma = \{a, b, c, B, X, Y, Z\}$
4.  $\delta$  a seguir.
5.  $q_0$  - o estado inicial
6.  $F = \{q_{ac}\}$



## Hierarquia de Chomsky



**LR = Linguagens  
Regulares**

**LLC = Linguagens  
Livres de Contexto**

**LSL = Linguagens  
Sensíveis ao Contexto**

**LEF = Linguagens  
Recursivamente  
Enumeráveis**

## Linguagens e Reconhecedores

Linguagem	Gramática	Reconhecedor	Tempo para reconhecer $w$ ; $ w =n$
Tipo 0: Linguagens Computáveis ou Recursivamente Enumeráveis	Gramáticas com Estrutura de Frase	Máquinas de Turing	NP-completo
Tipo 1: Sensíveis ao Contexto	Gramáticas Sensíveis ao Contexto	Máquinas de Turing com memória limitada	Exponencial: $O(2^n)$
Tipo 2: Livres de Contexto	Gramáticas Livres de Contexto	Autômatos à Pilha	Polinomial Espaço: $O(n)$ ; Tempo: Geral: $O(n^3)$ ; Não-ambíguas: $O(n^2)$
Tipo 3: Conjuntos Regulares	Gramáticas Regulares	Autômatos Finitos	Linear: $O(n)$ e $O( E )$ (no tamanho do AF)