Resumo de TC feito por:
Tomás F. D. Simões
+351 963 246 240

Teoria de conjuntos:

- $S_1 \cup S_2 = \{x : x \in S_1 \vee x \in S_2\}$
- $S_1 \cap S_2 = \{x : x \in S_1 \wedge x \in S_2\}$
- $S_1 - S_2 = \{x : x \in S_1 \wedge x \notin S_2\} = S_1 \cap \overline{S_2}$
- $\overline{S} = \{x : x \in U \wedge x \notin S\}$
- $\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S_2}$
- $\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S_2}$
- $S_1 \subseteq S \Rightarrow x \in S, \forall x \in S_1$
- $S = \{a,b,c\}$, $2^S = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a\,c\}, \{b,c\}, \{a,b,c\}\}$
- $S_1 \times S_2 = \{(x,y) : x \in S_1, y \in S_2\}$

Indução Matemática:

1º) Provar para o primeiro
2º) Hipótese indutiva (n)
3º) Passo dedutivo (n+1)

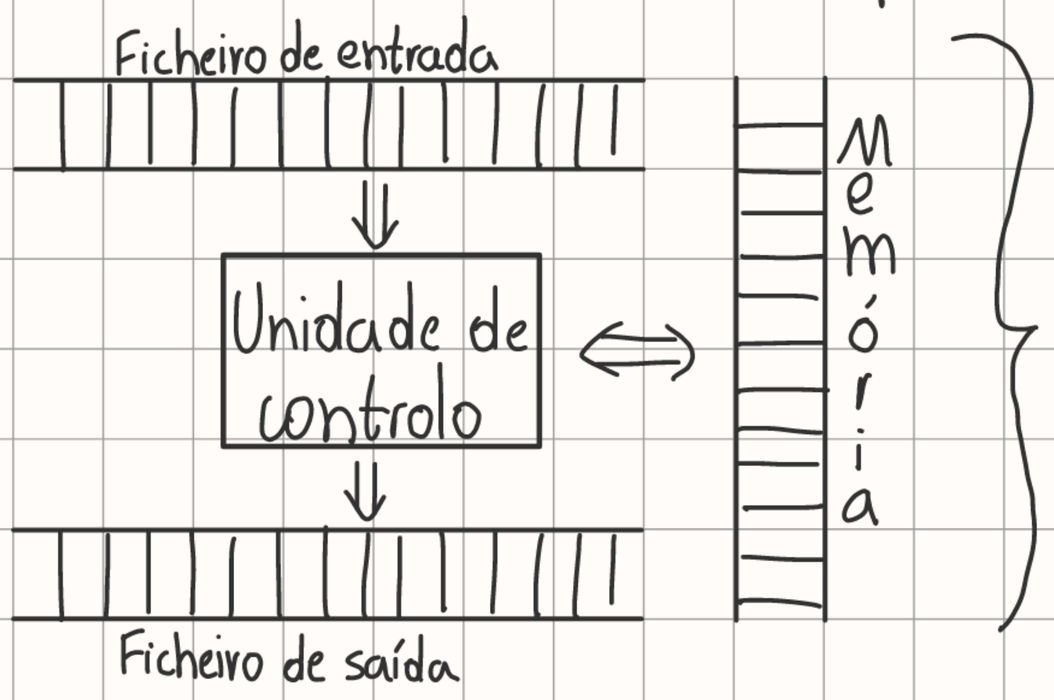
Linguagem formal: Representa-se através de símbolos. É regular se $\exists M$ (autómato): $L = L(M)$.

Alfabeto: Composto por um conjunto não vazio de símbolos. Representa-se por Σ.

- **Notas:** Σ^* contém vazio (λ). Σ^+ contém pelo menos um símbolo.

Linguagem: É qualquer subconjunto de Σ^*, com Σ um alfabeto qualquer. Pode ser finita ou infinita.

Autómato: Modelo abstrato de muitos dispositivos de hardware e software.



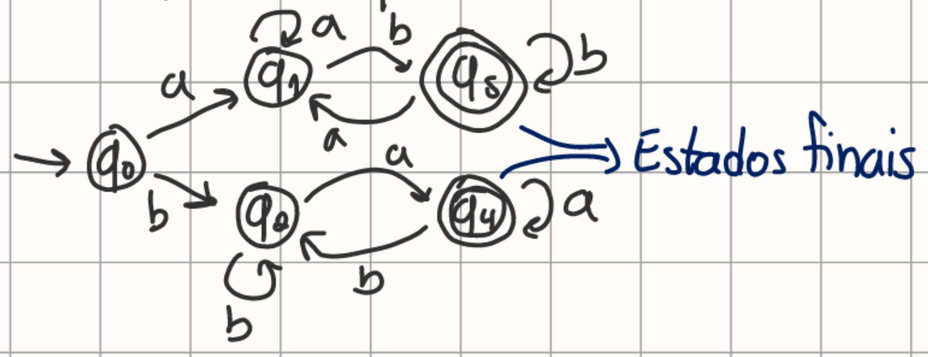
- **Saída:** Aceitadores: Reconhecem ou não uma cadeia na entrada.
  Transdutores: Se produzem uma cadeia por saída.
- **Memória:** Autómato finito: não tem memória temporária.
  Autómato de pilha: tem uma pilha. (LIFO)
  Máquina de Turing: memória em fita de dimensão infinita.

Linguagem L ⇒ Gramática G (gera as cadeias de L) ⇒ Autómato (Reconhece as cadeias de L)

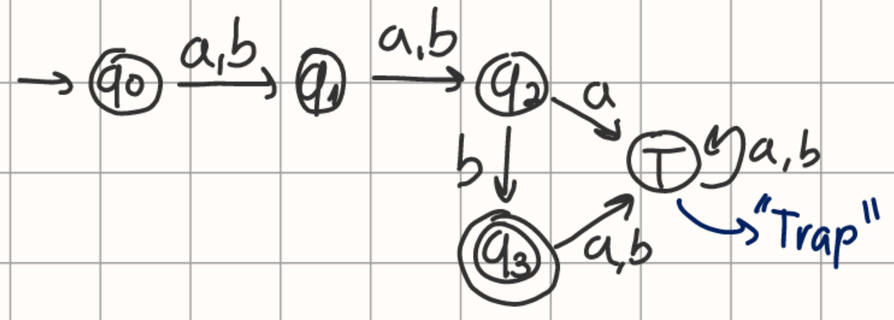
- **DFA (Deterministic Finite Automata):** $M = (Q, \Sigma, \delta, q_0, F)$, $q_0 \in Q$, $F \subseteq Q$
  A mesma cadeia segue sempre o mesmo caminho
  - Q: conjunto de estados
  - Σ: Alfabeto de entrada
  - δ: Função de transição (é função total)
  - q_0: estado inicial
  - F: conjunto de estados finais

- Exemplos:

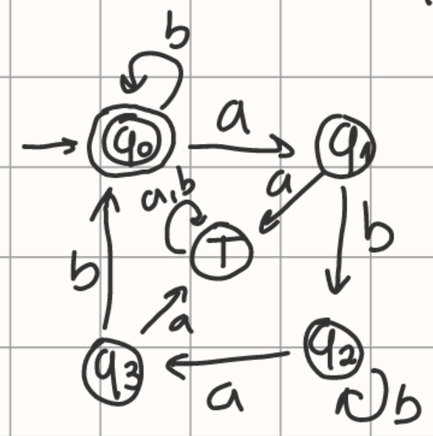
1º) $\Sigma = \{a,b\}$. DFA que aceita todas as cadeias que comecem e acabem com símbolos distintos:



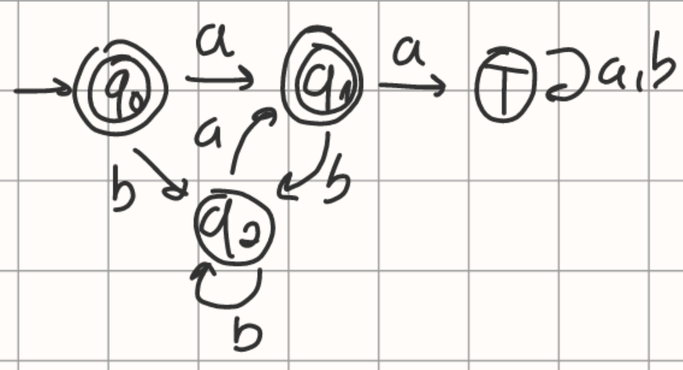
2º) $\Sigma = \{a,b\}$. DFA que aceita apenas cadeias com 3 símbolos, sendo o último um 'b':



3º) $\Sigma = \{a,b\}$. DFA que aceita cadeias com um nº par de 'a', sendo que cada 'a' é seguido por um 'b':



4º) $\Sigma = \{a,b\}$. DFA que não aceita cadeias com 2 'a' seguidos e que acabem com 'b':



5º) $\Sigma = \{cara, coroa\}$. DFA que aceite todas as sequências contenham pelo menos uma cara:

Formalmente:

$$M = (\{q_0, q_1\}, \{cara, coroa\}, q_0 \times cara \to q_1, q_0, q_1)$$
$$q_0 \times coroa \to q_0$$
$$q_1 \times cara \to q_1$$
$$q_1 \times coroa \to q_1$$

Non-Deterministic

→ NFA: Se existir 1 caminho no qual a cadeia é aceite, esta pertence à linguagem. δ é função parcial.

→ Diferenças NFA vs. DFA:

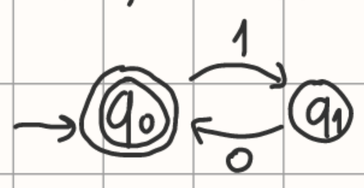
1º) O contradomínio de δ é a potência 2^Q.

2º) O conjunto $\delta(q_i, a)$ pode ser vazio. Se um símbolo não tem "saída" de um estado é como uma trap.

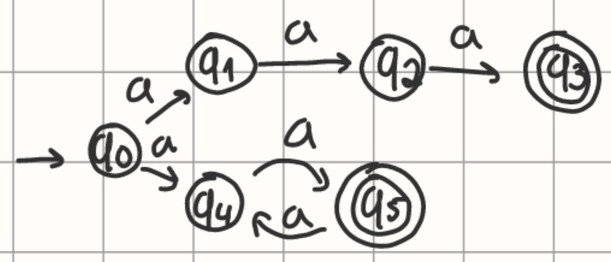
3º) λ pode também ser argumento de δ.

- Exemplos:

1º $L(M) = \{(10)^n : n \geq 0\}$. Nota: $a^0 = \lambda$



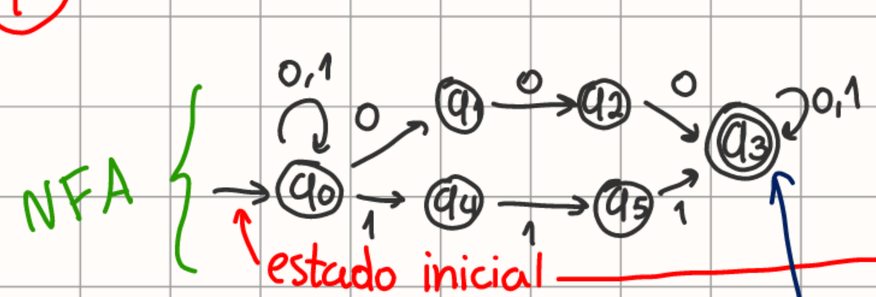
2º $L(M) = \{a^3\} \cup \{a^{2n} : n \geq 0\}$:



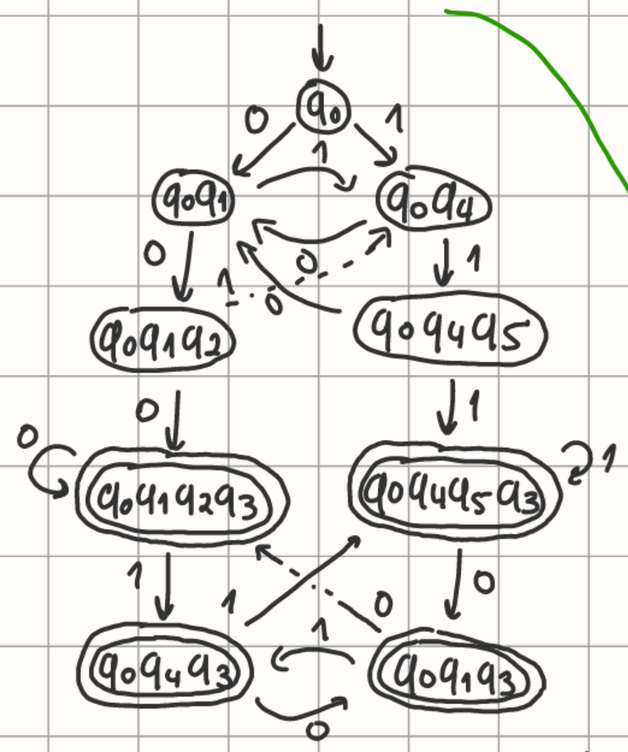
→ DFA → NFA: DFA já é um caso particular de NFA.

→ NFA → DFA: Exemplos:

1º

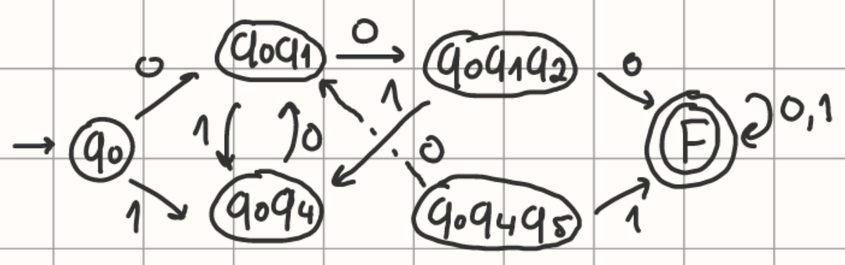


	0	1
→ q_0	q_0q_1	q_0q_4
q_0q_1	$q_0q_1q_2$	q_0q_4
q_0q_4	q_0q_1	$q_0q_4q_5$
$q_0q_1q_2$	$q_0q_1q_2q_3$	q_0q_4
$q_0q_4q_5$	q_0q_1	$q_0q_4q_5q_3$
* $q_0q_1q_2\underline{q_3}$	$q_0q_1q_2q_3$	$q_0q_4q_3$
* $q_0q_4q_5\underline{q_3}$	$q_0q_1q_3$	$q_0q_4q_5q_3$
* $q_0q_4\underline{q_3}$	$q_0q_1q_3$	$q_0q_4q_5q_3$
* $q_0q_1\underline{q_3}$	$q_0q_1q_2q_3$	$q_0q_4q_3$

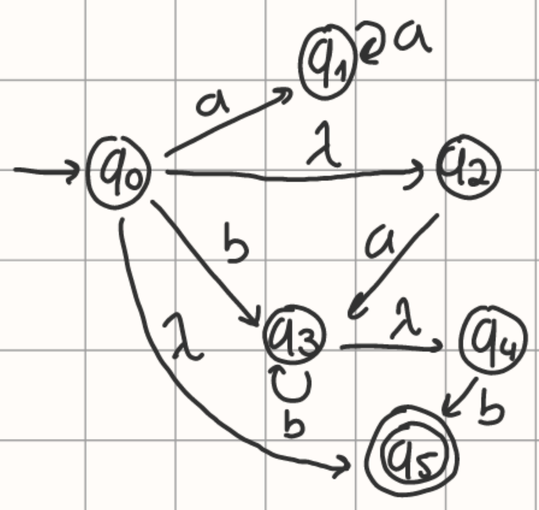


DFAs

⇓ otimização



2º



	a	b
* → $q_0q_2q_5$	$q_1q_3q_4$	q_3q_4
$q_1q_3q_4$	q_1	$q_3q_4q_5$
q_3q_4	∅	$q_3q_4q_5$
q_1	q_1	∅
* $q_3q_4q_5$	∅	$q_3q_4q_5$
Trap → ∅	∅	∅

Só pode haver 1 inicial no DFA

Expressão Regular (ER): Define uma linguagem. Utiliza símbolos do alfabeto, os operadores de união (+), concatenação (.), e fecho estrela (* → ≥0), e parênteses.

→Regras:

-Comulativas e Associativas:

- $L+M=M+L$
- $(L+M)+W=L+(M+W)$
- $(LM)W=L(MW)$
- $LM \neq ML$

-Distributivas:

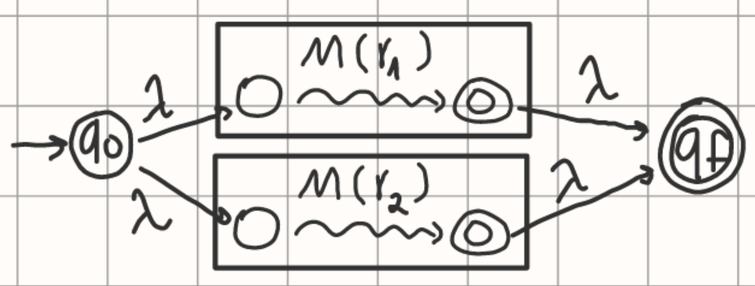
- $L(M+W)=LM+LW$
- $(M+W)L=ML+WL$

→Exemplos:

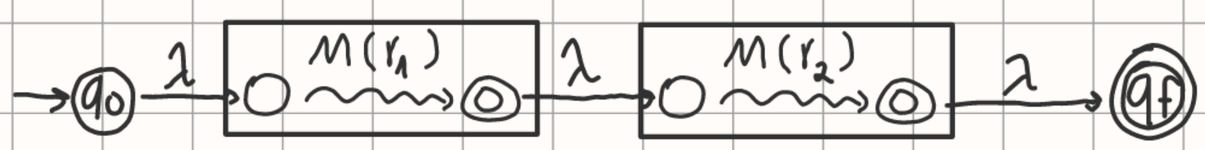
1) $\Sigma=\{0,1\}$. W: tenha pelo menos um par de 0s consecutivos: $(0+1)^*00(0+1)^*$.
2) $\Sigma=\{0,1\}$. W: não tenha qualquer par de 0s consecutivos: $(01+1)^*(\lambda+0)$.

→ER→NFA:

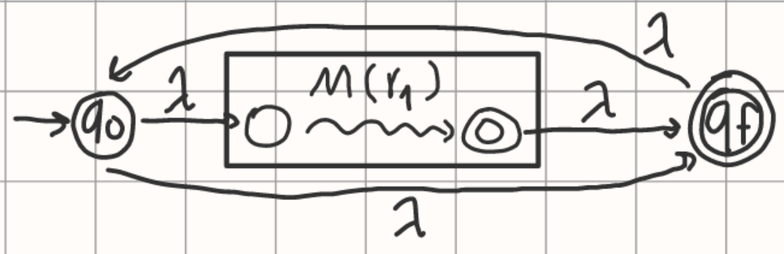
•$L(r_1+r_2)$:



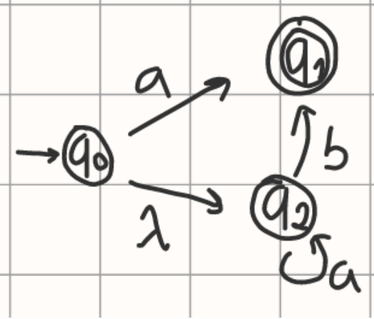
•$L(r_1.r_2)$:



•$L(r_1^*)$:



•Exemplo: $\Sigma=\{a,b\}$. ER=a+a*b:

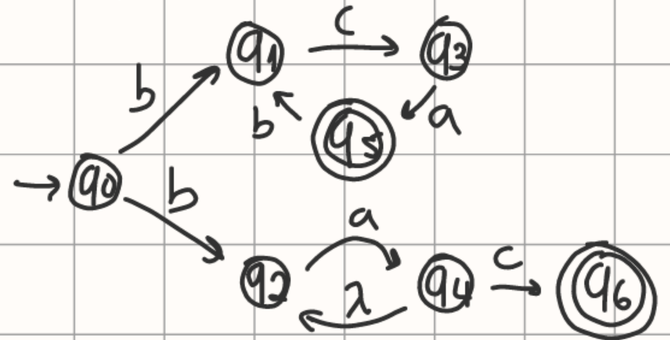


→NFA→ER: Algoritmo de eliminação de estados: Ficar com um estado inicial e final quando terminado:

• (P Q) = Etiqueta(P,Q) + Etiqueta(P,R). Etiqueta(R,R)*. Etiqueta(R,Q).

entrada: P; saída: Q; estado a eliminar: R

• Exemplo: $\Sigma = \{a,b,c\}$:



A eliminar: q_3; saída: q_5; entrada: q_1

$q_1q_5 = \emptyset + c\lambda a$

• q_1:

$q_0q_5 = \emptyset + b\lambda ca$

$q_5q_5 = \lambda + b\lambda ca$

• q_4:

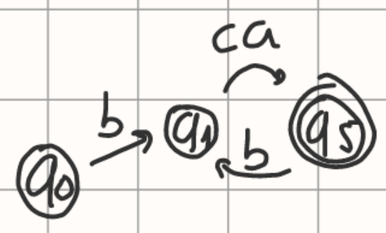
$q_2q_2 = \lambda + aa^*\lambda$

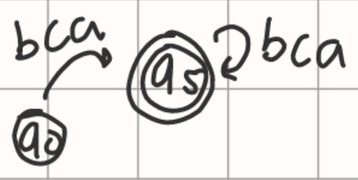
$q_2q_6 = \emptyset + aa^*c$

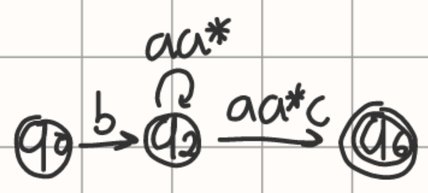
• q_2:

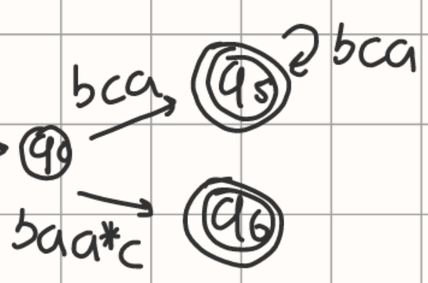
$q_0q_6 = \emptyset + b(aa^*)^*aa^*c$

$= \emptyset + baa^*c$









• ER = $bca(bca)^* + baa^*c$

Gramáticas Regulares: Formalmente $G = (V,T,S,P)$ (V: conj.º variáveis; T: conj.º símbolos terminais; S: símbolo inicial; P: produções)

→ Linear:

• À esquerda: $X \in T; A,B \in V$: $A \to BX, B \to AX, \ldots$. G e L regulares

• À direita: $A \to XB, B \to XA, \ldots$. G e L regulares

• À esquerda e à direita: $A \to BX, B \to xA, \ldots$. G e L irregulares

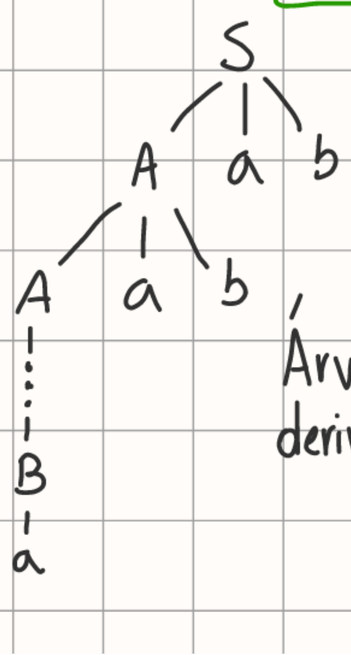
→ Não linear: $A \to AxB, B \to ABx, \ldots$

→ Exemplo: Símbolo inicial: S; produções:

$S \to Aab$

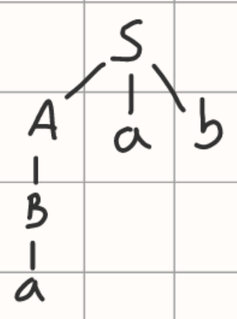
$A \to Aab \mid B$ (Vars)

$B \to a$ (símbolos terminais)



Árvore de derivação

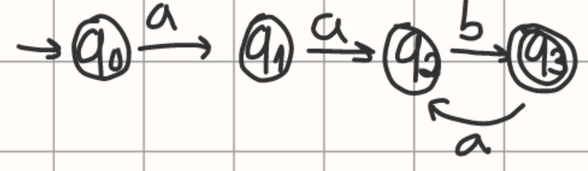
Árvore para a cadeia aab:



$ER = a(ab)^*ab$

$L = \{a(ab)^n, n \geq 1\}$

NFA:

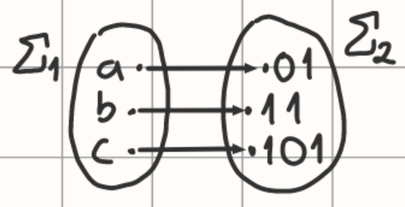


Linguagens Regulares:

→ L_1 e L_2 linguagens regulares:

- $L_1 \cap L_2$, $L_1 \cup L_2$, $L_1 L_2$, $L_1 - L_2$, complementar(L_1), L_1^*, L_1^R são linguagens regulares. (ou L_2)

→ **Homomorfismo:** Correspondência biunívoca (bijeção) entre dois conjuntos. L regular ⇒ imagem holomórfica de L também o é:



$h(abc) = h(a)h(b)h(c) = (01)(11)(101) = 0111101$

→ **Quociente à direita:** L_1, L_2 linguagens regulares do mesmo alfabeto, $\forall w \in L_1 : w = XY \wedge Y \in L_2 \Rightarrow X \in L_1/L_2$ e L_1/L_2 é regular.

→ Para ver se uma cadeia pertence a uma linguagem basta recorrer ao seu autómato.

→ Uma linguagem regular é finita se o seu autómato não tiver ciclos, é infinita caso contrário.

→ $L = (L_1 - L_2) \cup (L_2 - L_1)$ é linguagem vazia se $L_1 = L_2$.

→ Se for possível criar um autómato/expressão regular/gramática então a linguagem é regular.

→ **Princípio do Pombal:** Se tivermos n caixas e m objetos: $m > n$, $\exists$ (existe pelo menos 1) caixa com mais do que 1 objeto. (Serve para provar que uma linguagem que necessita de memória é irregular)

→ **Lema da Bombagem:** L uma linguagem Regular infinita $\Rightarrow \exists m \in \mathbb{N}^{*}$ (inteiros positivos) $: w = XYZ \wedge |XY| \leq m \wedge |Y| \geq 1 : w_i = XY^iZ \in L, \forall w \in L : |w| \geq m$.
(↳ contém a parte infinita)

- Pela negativa provamos que uma linguagem infinita é irregular. Exemplo:

$L = \{baba^p b^{3p} aba, p > 0\}$

Assumindo $p = m > 3$:

$|w| = 3 + m + 3m + 3 = 4m + 6 \geq m$ ✓

$XYZ = \overbrace{baba^{m-3}}^{X} \overbrace{a^3}^{Y^1} \overbrace{b^{3m} aba}^{Z} = baba^m b^{3m} aba = w$ ✓

bab	a^{m-3}	$a^3 b^{3m} aba$
X	Y^1	Z

$|XY| = 3 + m - 3 = m \Rightarrow |XY| \leq m$ ✓

$|Y| = m - 3 > 3 - 3 = 0 \Rightarrow |Y| > 0 \Rightarrow |Y| \geq 1$ ✓

bab	a^{m-2}	$a^3 b^{3m} aba$
X	Y^2	Z

→ $W_2 = baba^{m+1} b^{3m} aba \notin L$ ⇒ Se bombearmos $n > 0$ a's em $baba^i a^3 b^{3m} aba$, com $i > m-3$, a cadeia sai fora da linguagem, e portanto prova que a linguagem é não regular.

Linguagens Livres de Contexto: Linguagens Regulares ⊂ Linguagens Livres de Contexto.

- **Gramáticas Livres de Contexto:** Gramática sem restrições para as produções: $G = (V, T, S, P) : A \to X$. $X \in (V \cup T)^*$, $e \vee$

- **Extrema esquerda/direita:** Coordena a substituição das variáveis: $S \to aAB$; $A \to bBb$; $B \to A \mid \lambda$, gerar "abbbb":
  - E.E.: $S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abbB \Rightarrow abbA \Rightarrow abbbBb \Rightarrow abbbb$ ($B \to \lambda$, $B \to \lambda$)
  - E.D.: $S \Rightarrow aAB \Rightarrow aAA \Rightarrow aAbBb \Rightarrow aAbb \Rightarrow abBbbb \Rightarrow abbbb$ ($B \to \lambda$, $B \to \lambda$)

→ **Ambiguidade:** Várias derivações possíveis para a mesma cadeia, gerar estas árvores todas é parsing exaustivo. $aA \mid bB$ ✓

→ **Gramática Simples (S. Grammar):** $A \to aX, A \in V, a \in T, X \in V^*$. $\exists^1 A \to aX, A \in V, a \in T, \forall X \in V^* \Rightarrow A \to aA \mid aB$ ✗

→ **Simplificação de GLC:** Remoção de produções:

1º) lambda.

2º) unidade.

3º) inúteis.

→ **Forma Normal Chomsky:** $A \to BC \mid a, B, C \in V, a \in T$. } Necessário simplificação GLC

→ **Forma Normal Greibach:** $A \to aX, a \in T, X \in V^*$. } Necessário simplificação GLC

• Remoção de produções lambda:

1º) $V_N = \{\}$

2º) Adicionar a V_N variáveis que produzam lambda diretamente. → ex.: $A \to ab|\lambda$

3º) Adicionar a V_N variáveis que produzam lambda indiretamente. → ex.: $B \to bc|A$

ex.: $C \to AaB$

$S \to ABC$ | $A \to ab|\lambda$ | $B \to bc|A$ ⇒ $S \to ABC|BC|AC|C$ | $A \to ab$ | $B \to bc|A$ | $C \to AaB|aB|Aa|a$

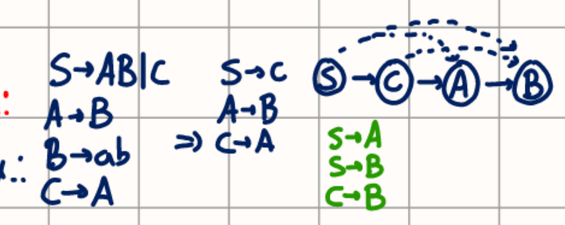
4º) Retirar λ e as produções que incluam variáveis em V_N ficam, adicionando a mesma produção sem essa variável.

Nota: Caso $S \to \ldots|\lambda$, fazemos $S' \to S|\lambda$

$S \to \ldots$

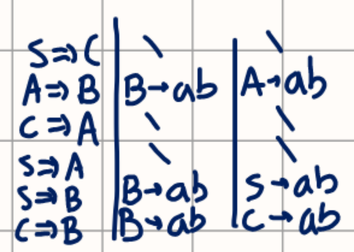
• Remoção de produções unidade:

1º) Identificar pares unidade. → ex.: $S \to AB|C$, $A \to B$, $B \to ab$, $C \to A$ ⇒ $S \to c|ab$, $A \to B$, $C \to A$



$S \to A$, $S \to B$, $C \to B$

ex.:



2º) Criar a tabela pares unidade, produções da variável produzida exceto unidades, produções adicionais da produtora.

3º) Remover produções unidade e adicionar as novas produções. → ex.: $S \to AB|ab$, $A \to ab$, $B \to ab$, $C \to ab$

• Remoção produções inúteis:

Remover ciclos infinitos:

1º) $V_i = \{\}$

2º) Adicionar a V_i variáveis com produções com apenas símbolos terminais. → ex.: $A \to aB|a$, $D \to d$

ex.: $S \to AB$, $B \to Ab|aA|CaD$

3º) Adicionar a V_i variáveis que indiretamente tenham produções apenas com símbolos terminais.

4º) Remover todas as produções que contenham $X \in V : X \notin V_i$ e remover X. → ex.: $S \to AB$, $A \to aBa$, $B \to Ab|aA|CaD$, $C \to aCb$, $D \to d$ ⟹ $V_i = \{S, A, B, D\}$ ⟹ $S \to AB$, $A \to aBa$, $B \to Ab|aA$, $D \to d$

5º) Remover variáveis inatingíveis. → ex.: $S \to AB$, $A \to aBa$, $B \to Ab|aA$, $D \to d$ ⟹ $S \to AB$, $A \to aBa$, $B \to Ab|aA$

push-down

Autómato de Pilha: DPDA + NPDA. $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$. Apenas pode acabar se a pilha estiver vazia.

$\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$, é função parcial

Não há qualquer equivalência

Γ: alfabeto da pilha; q_0: estado atual; z_0: símbolo inicial da pilha (#)

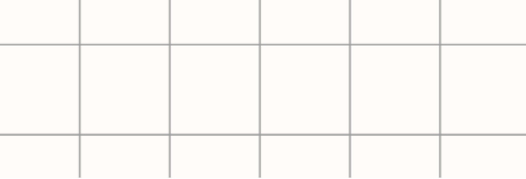
δ: char de entrada, símb. topo da pilha ⟹ estado seguinte, operação a efetuar na pilha

operação a efetuar na pilha: push: $G, t_1 : t_2 t_1$; pop: $G, t_1 : \lambda$; substituição: $G, t_1 : t_2$

→ DPDA:

- $\delta(q, a, b)$ contém, no máximo, uma transição.
- $\delta(q, \lambda, b) \neq \{\} \Rightarrow \delta(q, a, b) = \{\}, \forall a \in \Sigma$

→ Exemplo: $L = \{a^n b^n : n \geq 1\}$:

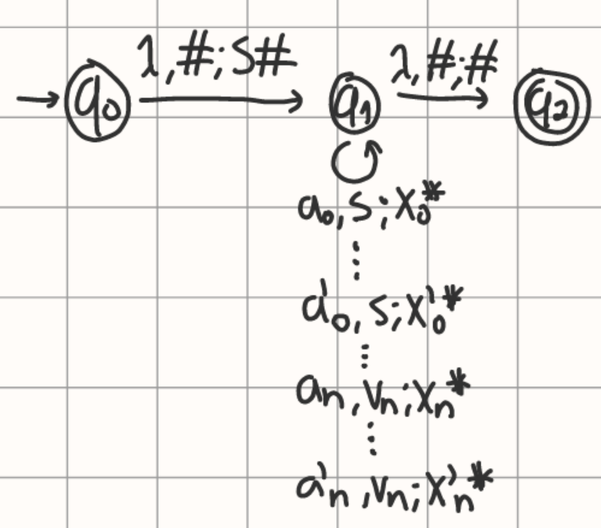


→ GLC → PDA: G na F.N. Greibach: $\Gamma = V$, $\Sigma = T$:

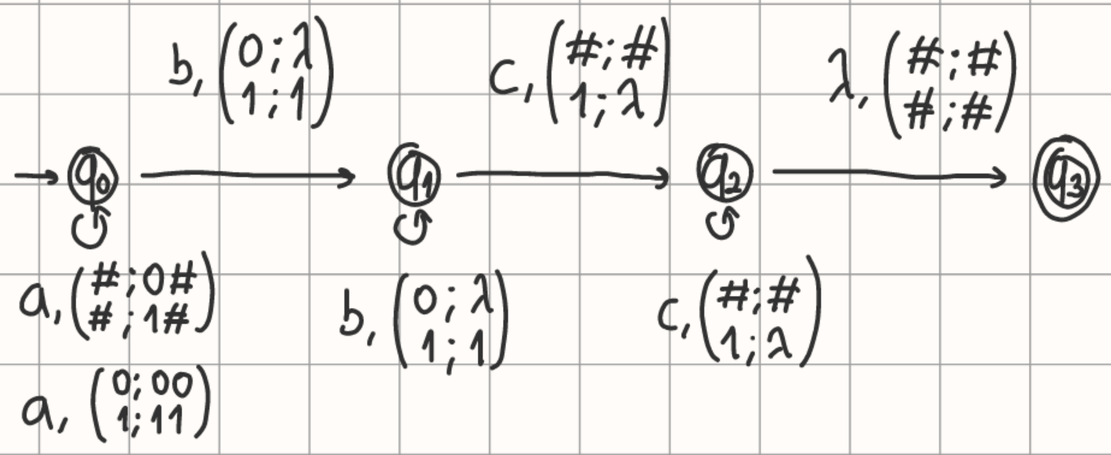
$S \to a_0 X_0^* | \ldots | a'_0 X'^*_0$

$\vdots$

$V_n \to a_n X_n^* | \ldots | a'_n X'^*_n$



→ Duas pilhas: $L = \{a^n b^n c^n, n \geq 1\}$:



Máquina de Turing: $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$

- Σ: conjunto de símbolos da fita
- Γ: "caracter branco" → "baliza" a string
- δ: Estado atual + Char da fita a ser lido → Próx. estado + Escrever na fita um novo símbolo + Mover para a esquerda ("L") ou direita ("R")

→ Características:

- Fita ilimitada nas duas direções.
- MT é determinista.
- $\nexists$ ficheiro de entrada especial.

→ Condições iniciais:

- A cadeia w está escrita na fita.
- w está rodeada de chars brancos "□".
- MT inicia-se no estado q_0 e a apontar para o char mais à esquerda.

→ Paragem:

- MT está num estado final e pára ⇒ w é aceite.
- MT não está num estado final e pára ⇒ w não é aceite.
- MT não pára ⇒ w não é aceite.

→ MT como transdutor: $\hat{w} = j(w)$.
(saída, entrada)

- Unário: Apenas um símbolo: "1" (sem contar com "□").
- Estratégia geral: Marcar o dígito atual / grupo de dígitos já considerados (geralmente com "*").

Resumo de TC feito por:
Tomás F. D. Simões
+351 963 246 240