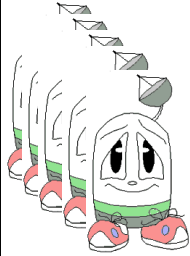


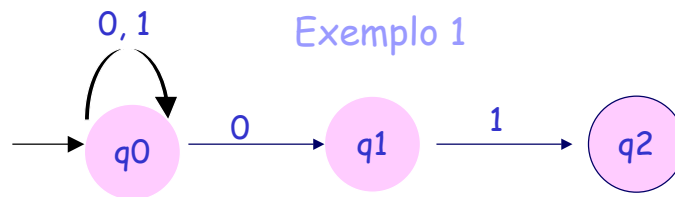
Automatos Finitos



AF Não-determinísticos
Equivalência entre AFND e AFD
Equivalência entre AF e GR
(H&U, 1969 e 1979), (H;M;U, 2001)
e (Menezes, 2002)

AF NÃO-Determinístico (AFND)

- Consideremos uma modificação no modelo do AFD para permitir
 - zero, uma ou mais transições de um estado sobre o **MESMO** símbolo de entrada.
 - Ou visto de outra forma:
 - a função toma um estado e uma entrada e devolve zero, um ou mais estados.
- Esse modelo é chamado AFND.



Esse autômato aceita cadeias de 0's e 1's que terminam em 01.
Analisem a aceitação de "00101"

Quando está em q0 e o símbolo lido é 0 ele tem a opção de:

continuar em q0 no caso do fim da cadeia não estar próximo

OU

ir para q1 porque advinha que o fim está chegando.

E na verdade ele executa as duas opções!

Por isso costumamos pensar que ele "advinha" qual é a alternativa correta de muitas.

AFND

- Uma seqüência de entrada $a_1a_2\dots a_n$ é aceita por um AFND
 - se existe **AO MENOS UMA** seqüência de transições, correspondente à seqüência de entrada, que leva o estado inicial para algum estado final.
- Ele funciona como se houvesse a multiplicação da unidade de controle, uma para cada alternativa,
 - processando **independentemente**, sem compartilhar recursos com as demais!
- **Pensem: Será que o não-determinismo aumenta o poder de reconhecimento de linguagens de uma classe de autômatos?**

Exemplo 2

- $L(M) = \{x \in \{0,1\}^* \mid x \text{ tenha dois } 0\text{'s consecutivos OU dois } 1\text{'s consecutivos}\}$
O "ou" é não-exclusivo

Ex 2

The screenshot shows the JFLAP interface. On the left, a DFA diagram with five states: q0 (start), q1, q2, q3, and q4 (accept). Transitions are: q0 to q0 on 0, q0 to q1 on 1, q1 to q2 on 1, q2 to q2 on 0, q0 to q3 on 0, q3 to q4 on 0, and q4 to q4 on 0 and 1.

Input	Result
110010101	Accept
11010	Accept
00101	Accept

$L(M) = \{x \in \{0,1\}^* \mid x \text{ tenha dois } 0\text{'s consecutivos OU dois } 1\text{'s consecutivos}\}$
 $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0,1\}, \delta, q_0, \{q_2, q_4\})$

Definição Formal de um AFND

Denotamos um AFND pela 5-tupla $(Q, \Sigma, \delta, q_0, F)$

onde Q, Σ, q_0 e F são os mesmos de um AFD e

$\delta: Q \times \Sigma \rightarrow 2^Q$, conjunto potência de Q , isto é, todos os subconjuntos de Q

$L(M) = \{w \mid \delta(q_0, w) \text{ contém um estado em } F\}$

No exemplo 2:

Entrada

Estado	0	1
q0	{q0,q3}	{q0,q1}
q1	\emptyset	{q2}
q2	{q2}	{q2}
q3	{q4}	\emptyset
q4	{q4}	{q4}

Exemplo 3

Construir um AFND que aceita cadeias $\in \{1,2,3\}^*$ tal que o último símbolo na cadeia tenha aparecido anteriormente. **Por exemplo, 121 é aceita; 31312 não é aceita.**

Dica: resolvam para os vocabulários mais simples antes

- 1) Construir um AFND que aceita cadeias $\in \{1\}^*$ tal que o último símbolo na cadeia tenha aparecido anteriormente
- 2) Construir um AFND que aceita cadeias $\in \{1,2\}^*$ tal que o último símbolo na cadeia tenha aparecido anteriormente

JFLAP -> untitled6

Input	Result
112121	Accept
2212	Accept
1112	Reject
12	Reject
21	Reject

JFLAP -> untitled6

Input	Result
1	Reject
11	Accept
111	Accept

JFLAP -> untitled6

Input	Result
121	Accept
31312	Reject
123	Reject

Não vou explorar o Teo 2.1

- Por questão de tempo, neste curso, será importante saber que AFD's são equivalentes a AFND e não construir o AFD a partir do não-determinístico.

Equivalência entre AFD e AFND

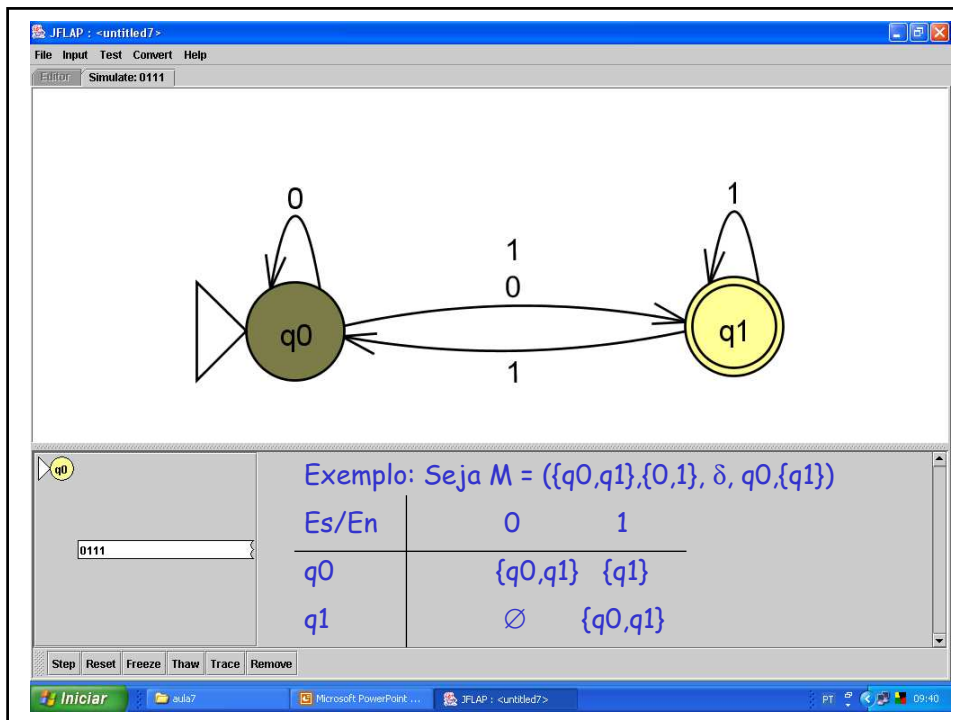
Teo 2.1 (H&U,79): Seja L o conjunto aceito por um AFND, então existe um AFD que aceita L . Isto é, eles são equivalentes.

Esse teorema responde a primeira pergunta desses slides.

Embora muitas vezes seja mais fácil construir um AFND para uma LR, o AFD tem na prática quase o mesmo número de estados que o AFND, embora ele tenha mais transições.

No pior caso, o menor AFD pode ter 2^n estados enquanto que o menor AFND para a mesma linguagem tenha somente n estados.

OBS: ACPD e ACPND (relacionados a GLC) NÃO aceitam a mesma classe de linguagem, já as MTD e MTND (GDC e GEF) são equivalentes.



Nós podemos construir um AFD $M' = (Q', \{0, 1\}, \delta', [q_0], F')$ aceitando $L(M)$ pela construção chamada de "construção de subconjuntos" porque ela envolve a construção de todos os subconjuntos do conjunto de estados do AFND. Ela é tal que Σ é o mesmo do AFND e o estado inicial é o conjunto contendo somente o estado inicial do AFND.

- Q' consiste de todos os subconjuntos de $\{q_0, q_1\}$: $[q_0]$, $[q_1]$, $[q_0, q_1]$, e \emptyset . Note que freqüentemente nem todos os estados são acessíveis do estado inicial e podem ser descartados.

$$\delta'([q_0], 0) = [q_0, q_1] \quad \delta'([q_0], 1) = [q_1]$$

$$\delta'([q_1], 0) = \emptyset \quad \delta'([q_1], 1) = [q_0, q_1]$$

$$\delta'([q_0, q_1], 0) = [q_0, q_1]$$

$$\text{Pois } \delta'([q_0, q_1], 0) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$$

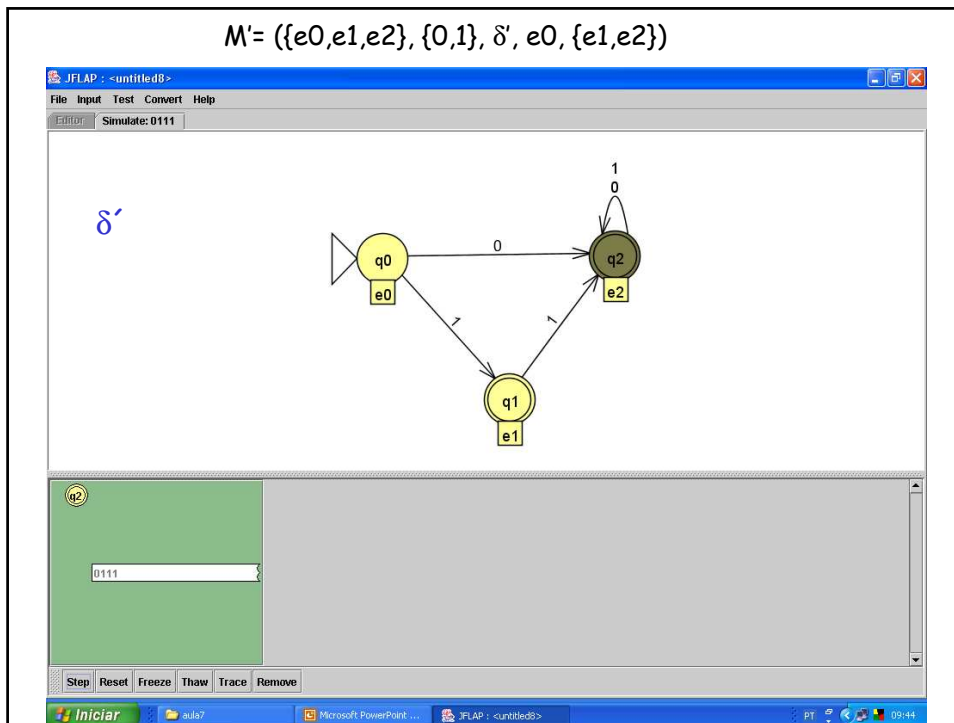
$$\delta'([q_0, q_1], 1) = [q_0, q_1]$$

$$\text{Pois } \delta'([q_0, q_1], 1) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_1\} \cup \{q_0, q_1\} = \{q_0, q_1\}$$

$$\delta'(\emptyset, 0) = \delta'(\emptyset, 1) = \emptyset$$

- $F' = \{[q_1], [q_0, q_1]\}$ isto é, estados onde F antigo estava presente

$$M' = (\{e0, e1, e2\}, \{0, 1\}, \delta', e0, \{e1, e2\})$$



Fim do Teo 2.1

Estados de não-aceitação em AFD (H,M,U, 2001) pg 67

- A definição estrita de um AFD EXIGE que todo estado tenha uma transição para cada símbolo de entrada.
- Se seguirmos esta definição o exemplo de AFD na seção de equivalência não é AFD no senso estrito.
- Porém, temos a facilidade de criar um estado de não-aceitação (**erro**) para o qual podem ir todas as entradas não necessárias na definição de uma linguagem.
- Por esta razão, relaxamos esta exigência e dizemos que um AFD tem **NO MÁXIMO UMA** transição em cada estado para cada símbolo em vez de exatamente uma transição.

AFD com estado de não-aceitação

$M = (\{q_0, q_1, \text{erro}\}, \{A..Z, a..z, 0..9, _ \}, q_0, \delta, \{q_1\})$

```
graph LR; q0((q0)) -- L --> q1((q1)); q0 -- D --> erro((erro)); q1 -- "L-bar, L" --> q1; style erro fill:#f9d5e5
```

Input	Result
LLL	Accept
LDD	Accept
DD	Reject
LLL	Accept
D*D	Reject

AF que reconhece identificadores em C

Run Inputs Clear Enter Lambda

Equivalência entre AFD e GR

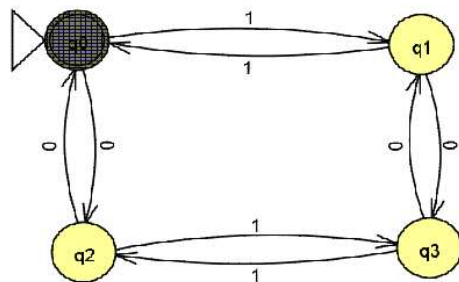
- Teo 3.5 (H&U, 69) Dado um AF M , existe uma gramática do tipo 3 tal que $L(G) = L(M)$.
- Prova: Sem perda de generalidade seja $M = (K, \Sigma, \delta, q_0, F)$ um AFD. Definimos um GR = (K, Σ, P, q_0) :

1. $B \rightarrow aC$ está em P se $\delta(B,a) = C$
2. $B \rightarrow a$ está em P se $\delta(B,a) = C$ e C está em F .

- Se q_0 está em F então λ está em $T(M)$. Neste caso, $L(G) = T(M) - \lambda$. Pelo Teo 2.1 (H&U, 69) podemos obter a partir de G uma nova GR onde:

$$L(G_1) = L(G) \cup \lambda = T(M)$$

Exemplo: $L(M) = \{w \mid w \text{ possui um nro par de } 0\text{'s e de } 1\text{'s}\}$



Como q_0 é final e aparece do lado direito, temos que acrescentar λ à $L(G)$ com as regras:

$S \rightarrow 0Q_2, S \rightarrow 1Q_1, S \rightarrow \lambda$

E

$G_1 = (\{S, Q_0, Q_1, Q_2, Q_3\}, \{0, 1\}, P_1, S)$

- $G = (\{Q_0, Q_1, Q_2, Q_3\}, \{0, 1\}, P, Q_0)$
- $\delta(q_0, 0) = q_2$ $Q_0 \rightarrow 0Q_2$ $\delta(q_0, 1) = q_1$ $Q_0 \rightarrow 1Q_1$
- $\delta(q_1, 0) = q_3$ $Q_1 \rightarrow 0Q_3$ $\delta(q_1, 1) = q_0$ $Q_1 \rightarrow 1Q_0$
- $\delta(q_2, 0) = q_0$ $Q_2 \rightarrow 0Q_0$ $\delta(q_2, 1) = q_3$ $Q_2 \rightarrow 1Q_3$
- $\delta(q_3, 0) = q_1$ $Q_3 \rightarrow 0Q_1$ $\delta(q_3, 1) = q_2$ $Q_3 \rightarrow 1Q_2$

JFLAP e a conversão AF \rightarrow GR

- O algoritmo que o JFLAP usa é diferente do Teo 3.5 visto,
 - pois permite que a cadeia nula apareça numa regra da GR sem que seja, unicamente, para derivar a cadeia nula quando esta pertence à linguagem.
- Isto quer dizer que a **concepção de GR para o JFLAP é mais flexível** do que a vista em sala.

Ex1: Usando o JFLAP para converter AF para GR

The screenshot shows the JFLAP software interface. On the left, a finite automaton is displayed with four states: q0 (start state, green), q1 (yellow), q2 (yellow), and q3 (yellow). Transitions are labeled with 'A', 'B', and 'C'. On the right, a table lists the grammar rules generated by the software:

A	\rightarrow 1S
S	\rightarrow 1A
A	\rightarrow 0C
C	\rightarrow 0A
B	\rightarrow 1C
C	\rightarrow 1B
S	\rightarrow λ
B	\rightarrow 0S
S	\rightarrow 0B

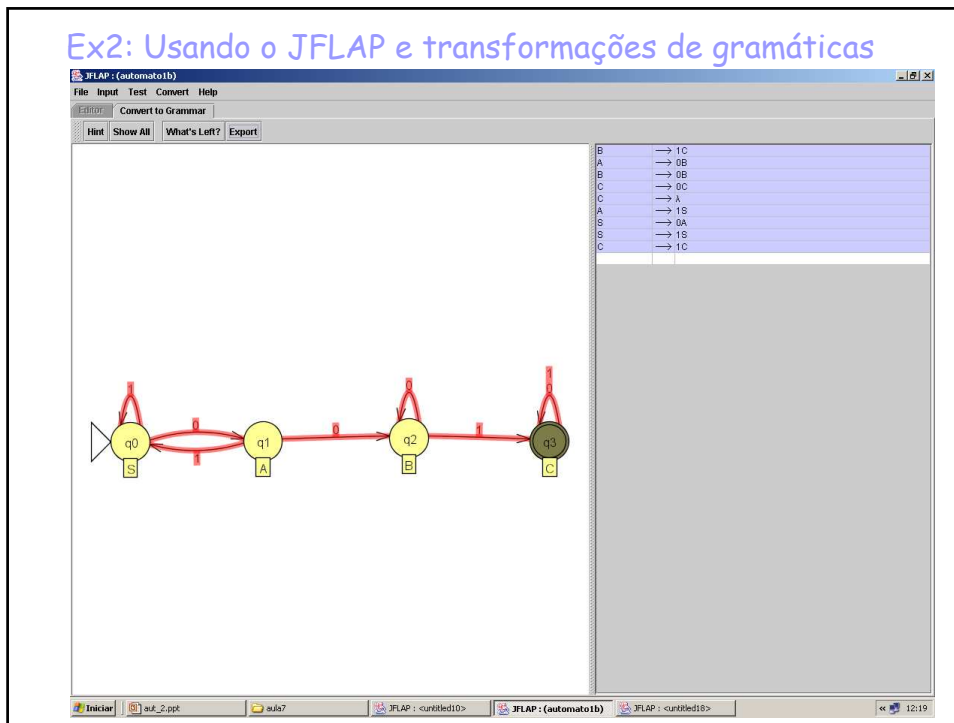
Below the table, a text box contains the following text:

Observem que só temos regras nos formatos:

$B \rightarrow \lambda$

$B \rightarrow aB$

Ex2: Usando o JFLAP e transformações de gramáticas



Remoção de regras com cadeia nula do Ex2

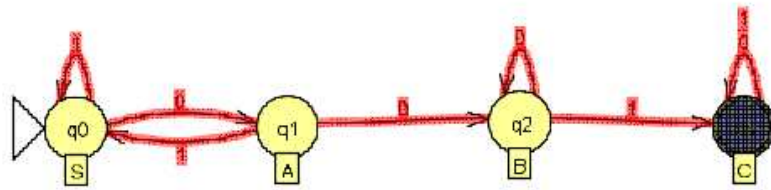
The screenshot shows the 'Lambda Removal' dialog box in JFLAP. The dialog box contains the following text: "Lambda removal complete. 'Proceed' or 'Export' available. Set that derives lambda: [C]". Below the dialog box, the grammar rules are updated to reflect the removal of the lambda rule for C.

Non-terminal	Terminal	Production
S	→	0A
S	→	1S
C	→	1C
A	→	1S
C	→	0C
B	→	0B
A	→	0B
B	→	1C
C	→	1
C	→	0
B	→	1

Como o símbolo inicial não leva em λ
Podemos remover ele da gramática

Exercício 1

- Usem o Teo 3.5 com a Ex2 para comparar com a saída do JFLAP após a remoção da cadeia nula.



Exercício 2

1. Fazer um AFD M que reconhece:
 - a) $L(M) = \{x \in \{0,1\}^* \mid \text{nro de } 1\text{'s em } x \text{ é múltiplo de } 3\}$
 - b) $L(M) = \{x \in \{0,1\}^* \mid x \text{ contém a subcadeia } 001\}$
2. Fazer um AFD M que reconhece:
 - a) a formação de identificadores em Pascal
 - b) números inteiros **sem sinal** em Pascal
 - c) Operadores relacionais em Pascal
3. Para $\Sigma = \{0,1\}$ faça AFD's que reconheçam $L1 = \emptyset$ e $L2 = \Sigma^*$
4. Construa um AFD M que reconhece $L(M) = \{ab^nc \mid n \geq 0\}$
5. Construa um AFD M que reconhece $L(M) = \{a^n b^m \mid n, m \geq 0 \text{ e } n+m > 0\}$. Veja que $n+m > 0$ implica na não possibilidade da vazia nula.
6. Construa o AF e depois a GR para $L1 = \{a^n b^m \mid n, m \geq 0\}$ e $L2 = \{a,b\}^*$