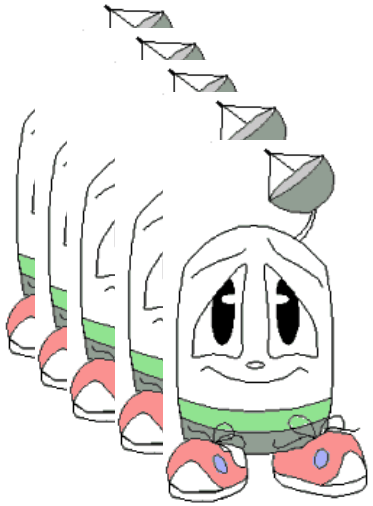
Autômatos Finitos



AF Não-determinísticos

Equivalência entre AFND e AFD

AFs e GRs

Implementação de AFs

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.

Por que isso é interessante?

- Pois possibilita tentar alternativas distintas
- Se cada estado representa uma opção, ir para mais de um estado representa tentar opções diferentes de caminho para a solução

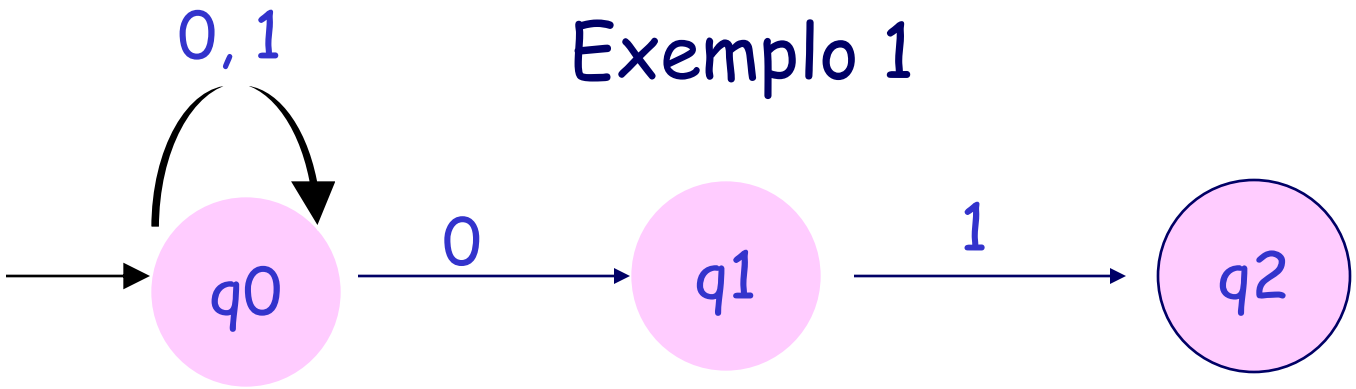
AFND

- Uma cadeia de entrada $a_1a_2...a_n$ é aceita/reconhecida por um AFND
  - se existe AO MENOS UMA sequência de transições que leva do 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,
  - aceitando a cadeia se ao menos uma delas parar num estado final.

- Pensem: Será que o não-determinismo aumenta o poder de reconhecimento de linguagens de um AF?

Exemplo 1



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

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

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

OU

ir para q1 porque aposta que o fim está chegando.

E na verdade ele executa as duas opções!

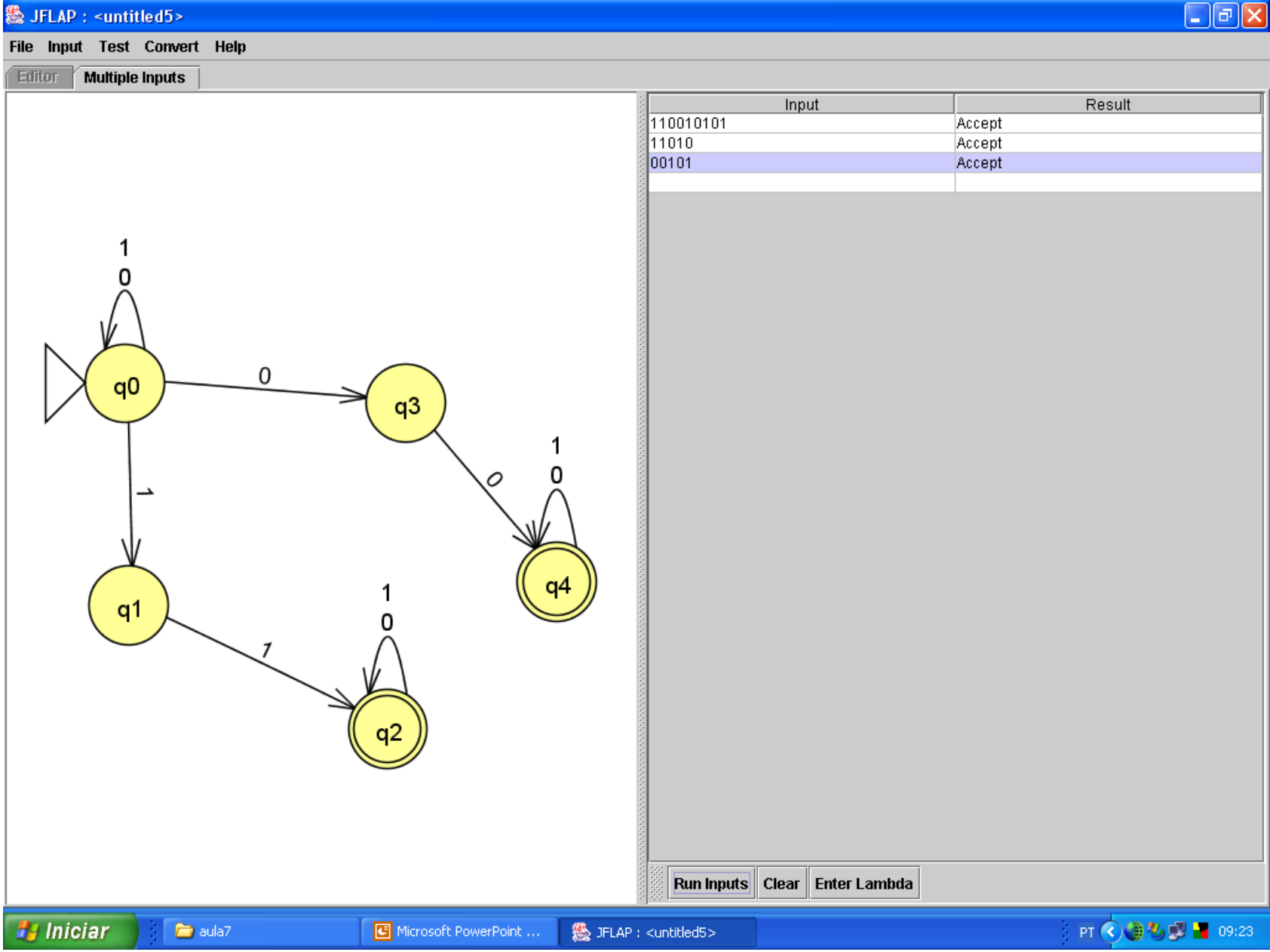
Por isso costumamos pensar que ele "advinha" qual é a alternativa correta entre muitas.

Exemplo 2

- $L(M) = (0+1)^* (00+11) (0+1)^*$ ou seja,

$x \in \{0,1\}^*$ | x tenha dois 0's consecutivos OU dois 1's consecutivos

Ex 2



$L(M) = (0+1)^*\ (00+11)\ (0+1)^*$

$M = (\{q0,q1,q2,q3,q4\},\{0,1\},\delta,q0,\{q2,q4\})$

Definição Formal de um AFND

Denotamos um AFND pela 5-tupla (Q,Σ,δ, q_0,F) onde Q,Σ, q_0 e F são os mesmos de um AFD e

δ: $Q \times \Sigma \rightarrow 2^Q$, conjunto potência de Q, isto é, todos os sub-conjuntos de Q

$L(M) = \{w \mid \delta'(q_0,w) \cap F \neq \emptyset\}$

No exemplo 2:

Estado	Entrada	
	0	1
q_0	$\{q_0,q_3\}$	$\{q_0,q_1\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$

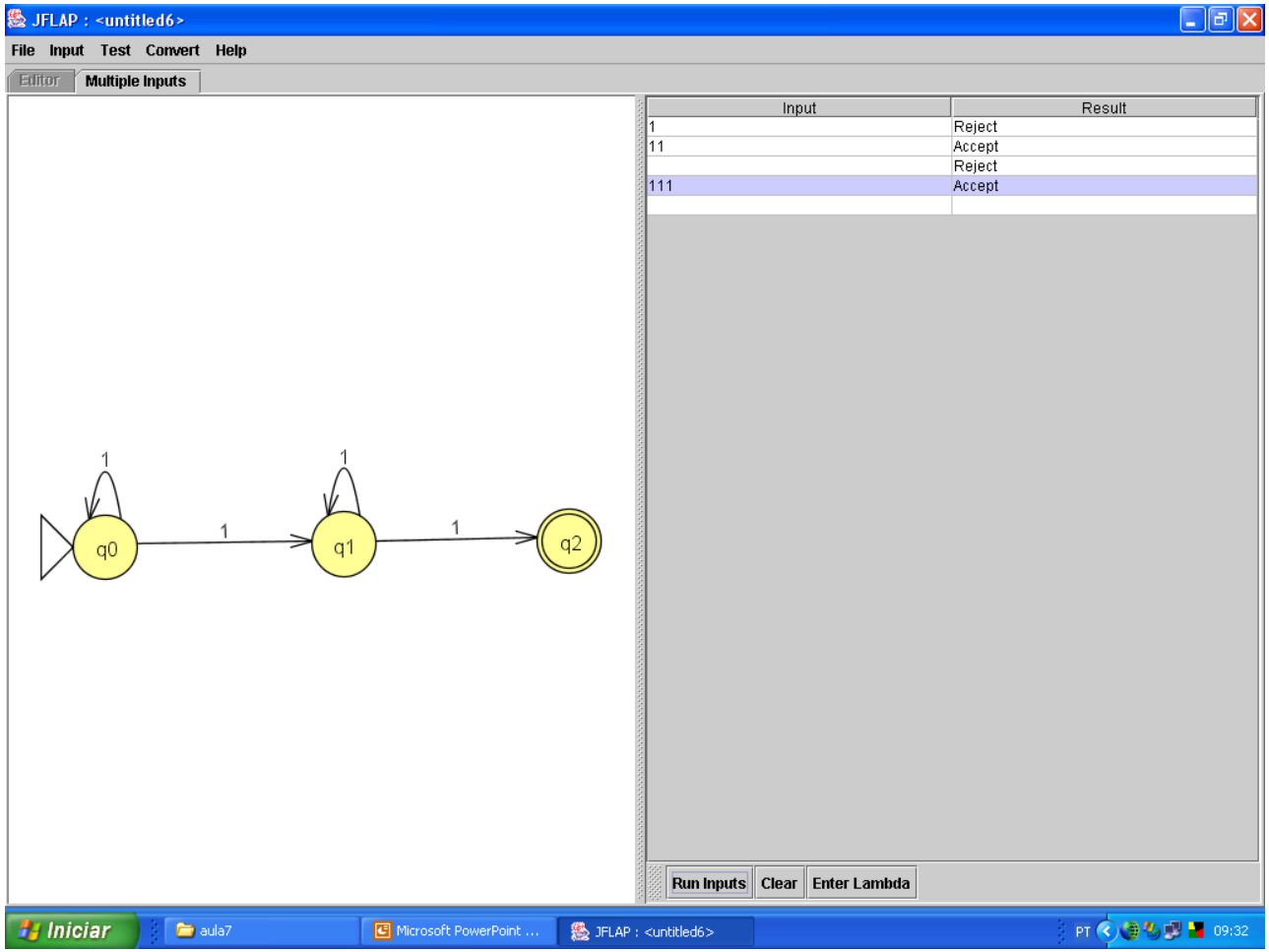
Exemplo 3

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.

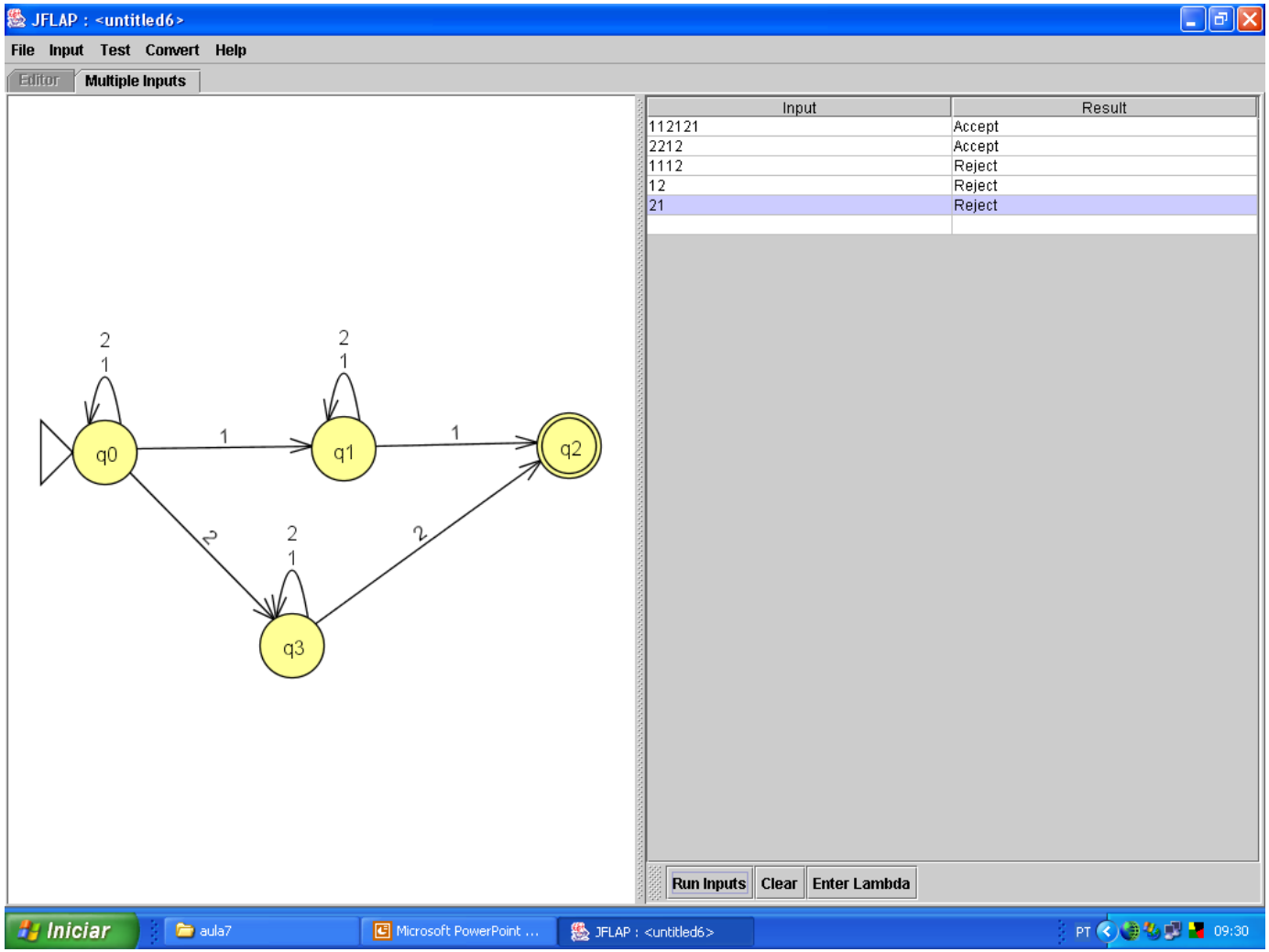
Dica: resolvam para os seguintes vocabulários mais simples antes

1) Construir um AFND que aceita cadeias $\in \{1\}^*$ tal que o último símbolo na cadeia tenha aparecido anteriormente ➔ 1*11*1
2) Construir um AFND que aceita cadeias $\in \{1,2\}^*$ tal que o último símbolo na cadeia tenha aparecido anteriormente ➔ ((1+2)*1(1+2)*1)+ ((1+2)*2(1+2)*2)

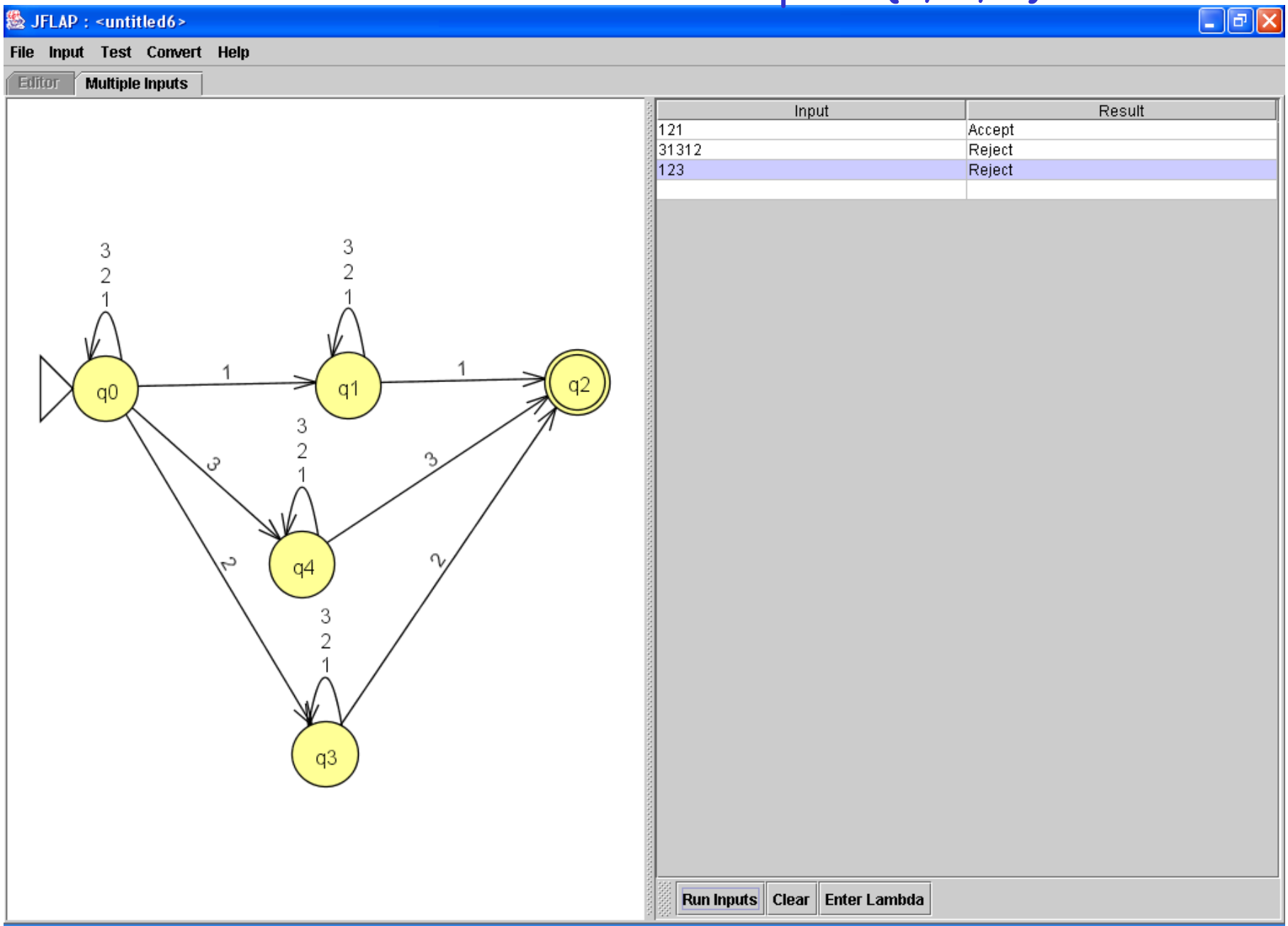
1) Construir um AFND que aceita cadeias $\in \{1\}^*$ tal que o último símbolo na cadeia tenha aparecido anteriormente



2) Construir um AFND que aceita cadeias ∈ {1,2}* tal que o último símbolo na cadeia tenha aparecido anteriormente



Estendendo o vocabulário para {1, 2, 3}:



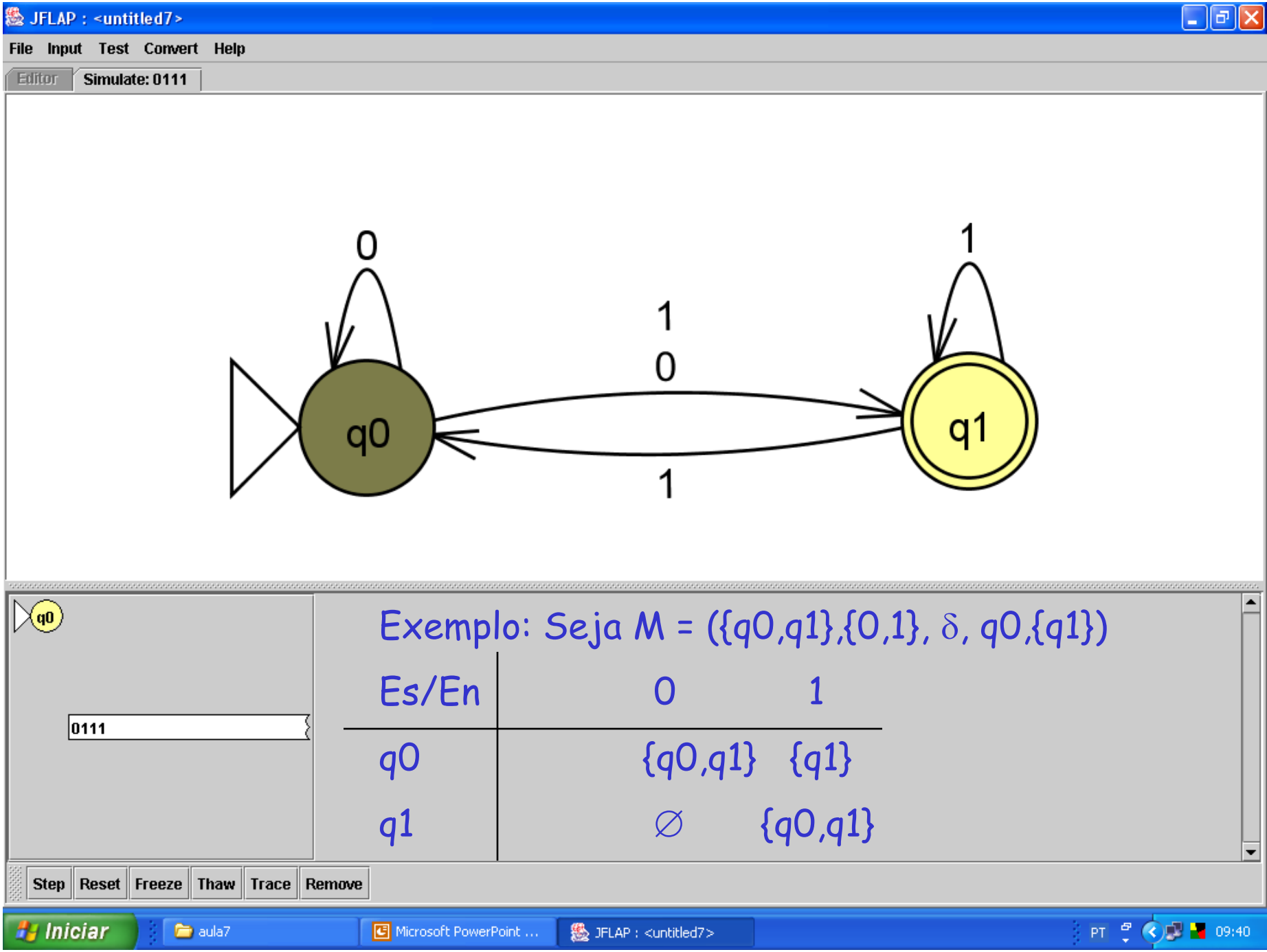
Equivalência entre AFD e AFND

Teorema: Se L é reconhecida por um AFND, então ela é reconhecida por um AFD.

Esse teorema responde a primeira pergunta desses slides.

Vamos propor um método para construir um AFD a partir do AFND. Daí se pode provar que as linguagens reconhecidas por ambos são iguais (veja essa prova na bibliografia).

Esse método é chamado de *construção de subconjuntos*


JFLAP : <untitled7>
File Input Test Convert Help
Editor Simulate: 0111
0
1
0
1
1
q0
q1
q0
0111
Exemplo: Seja M = ({q0,q1},{0,1}, δ, q0,{q1})
Es/En 0 1
q0 {q0,q1} {q1}
q1 ∅ {q0,q1}
Step Reset Freeze Thaw Trace Remove
Iniciar
aula7
Microsoft PowerPoint ...
JFLAP : <untitled7>
PT
09:40

Nós podemos construir um AFD M'= (Q', {0,1}, δ', {qo}, F') aceitando L(M) pela construção chamada de "construção de subconjuntos" (dos estados de 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 {qo,q1}:

  eo e1 e2 e3

  {qo}, {q1}, {qo,q1}, e $\varnothing$.

- $\delta'(\{q_1 \ldots q_n\}, a) = {}_{i=1}\cup^{n} \delta(qi, a)$;

- F' = {p | p $\subseteq$ Q' e p contém ao menos 1 elemento de F }

Assim, no exemplo:

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

$\delta'(\{q_1\},0) = \varnothing$ $\delta'(\{q_1\},1) = \{q_0,q_1\}$

$\delta'(\{q_0,q_1\},0) = \{q_0,q_1\}$

Pois $\delta(\{q_0,q_1\},0) = \delta(q_0,0) \cup \delta(q_1,0) = \{q_0,q_1\} \cup \varnothing = \{q_0,q_1\}$

$\delta'(\{q_0,q_1\},1) = \{q_0,q_1\}$

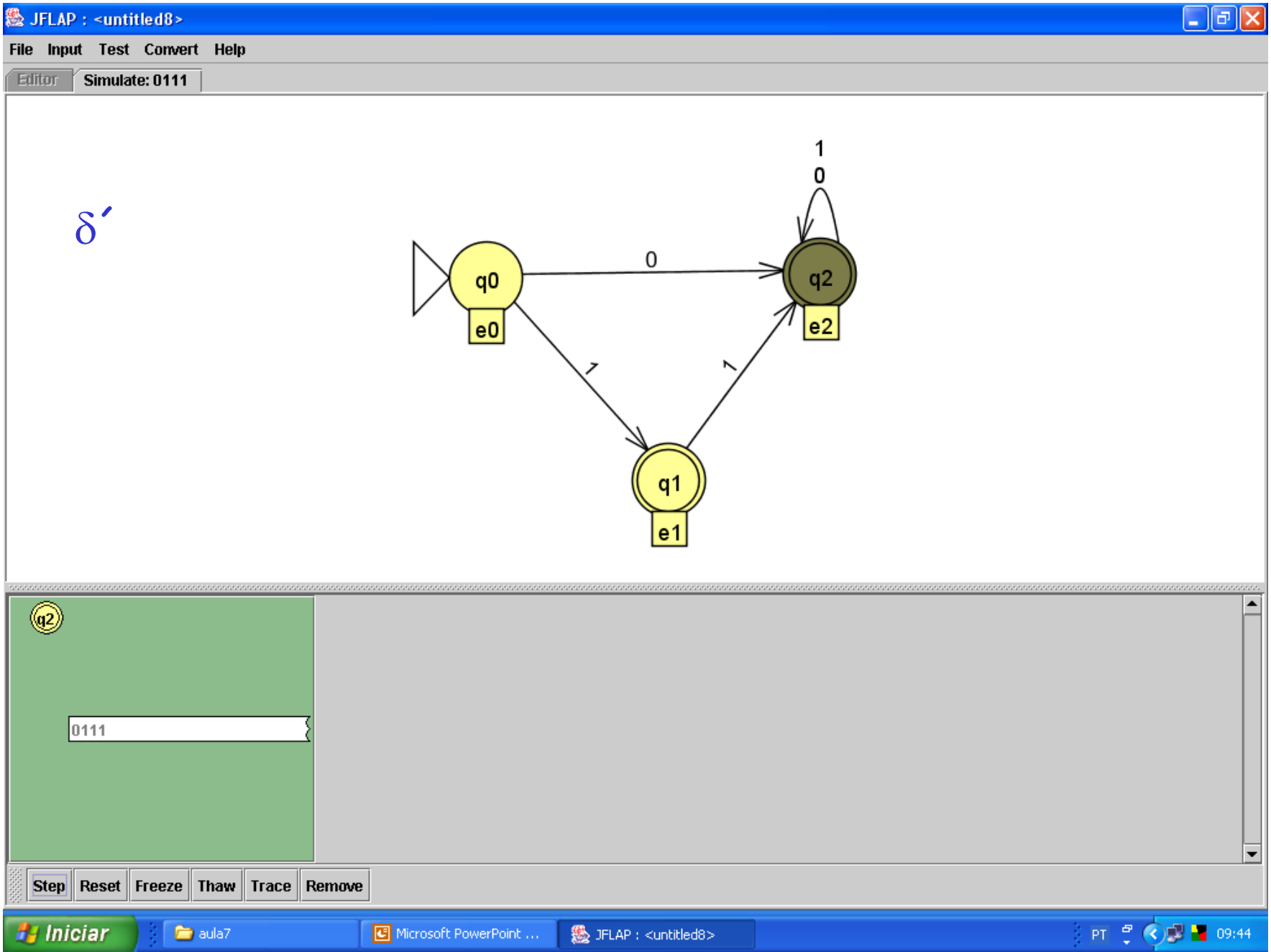
Pois $\delta(\{q_0,q_1\},1) = \delta(q_0,1) \cup \delta(q_1,1) = \{q_1\} \cup \{q_0,q_1\} = \{q_0,q_1\}$

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

- F' = {{q1},{q0,q1}} isto é, estados onde F antigo estava presente

δ	0	1
$\varnothing$	$\varnothing$	$\varnothing$
$\{q_0\}$	$\{q_0,q_1\}$	$\{q_1\}$
$\{q_1\}$	$\varnothing$	$\{q_0,q_1\}$
$\{q_0,q_1\}$	$\{q_0,q_1\}$	$\{q_0,q_1\}$

M'= ({eo,e1,e2}, {0,1}, δ', eo, {e1,e2}). L(M') = 0(0+1)* + 1 + 11(0+1)*

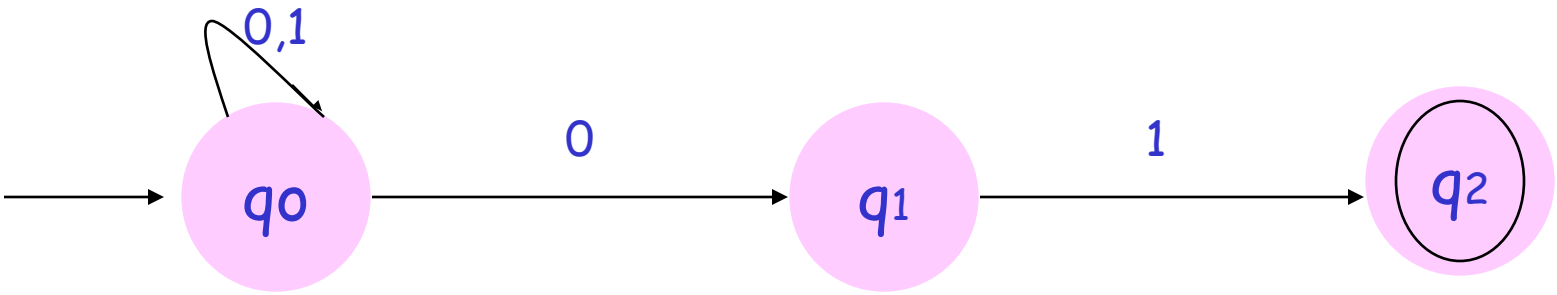


Estados Acessíveis e Não Acessíveis

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 AFD pode ter 2^n estados enquanto que o AFND para a mesma linguagem tem somente n estados.

Ex. seja o AFND que aceita todas as cadeias em {0,1} que terminam em 01.



AFND

	0	1
→ q0	{q0, q1}	{q0}
q1	∅	{q2}
q2	∅	∅

AFD

	0	1
∅	∅	∅
→ {q0}	{q0,q1}	{q0}
*{q1}	∅	{q2}
*{q2}	∅	∅
{q0, q1}	{q0,q1}	{q0,q2}
* {q0,q2}	{q0,q1}	{q0}
*{q1,q2}	∅	{q2}
*{q0,q1,q2}	{q0,q1}	{q0,q2}

Renomeando os estados:

AFND

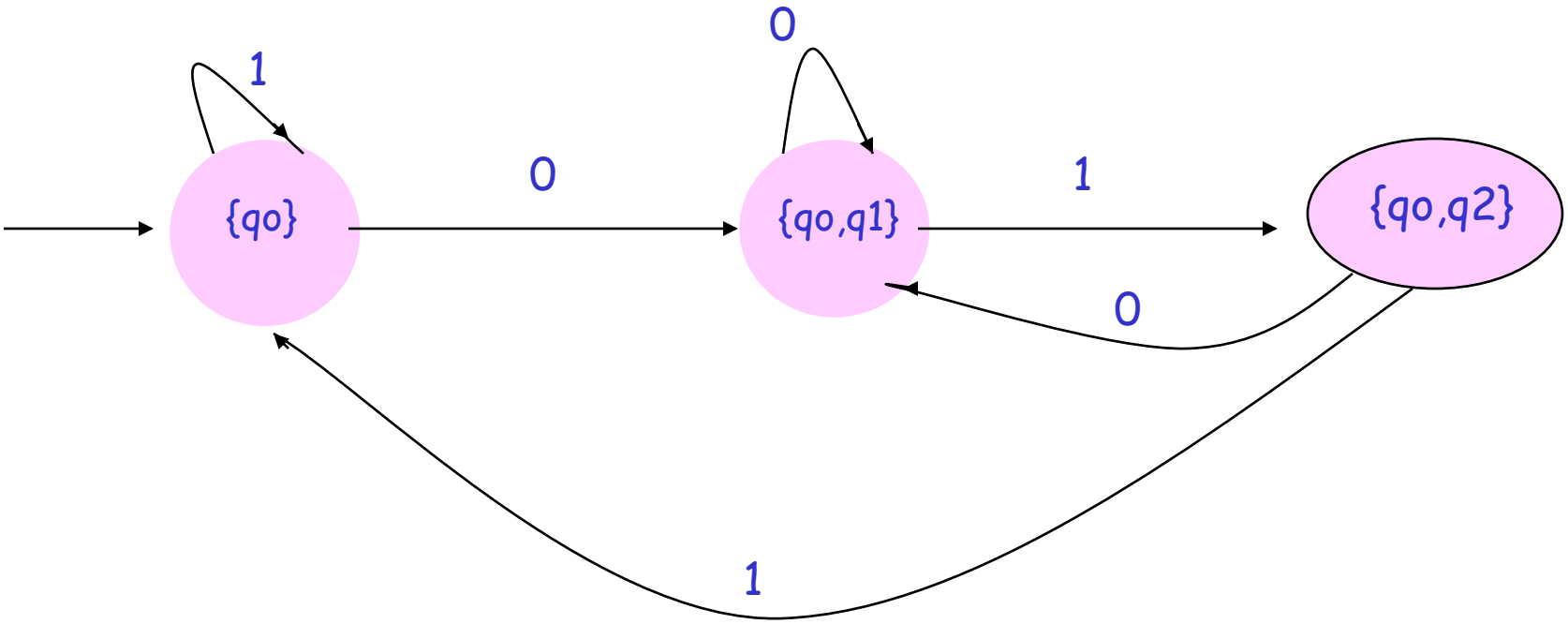
	0	1
→ q0	{q0, q1}	{q0}
q1	∅	{q2}
q2	∅	∅

AFD

	0	1
A	A	A
→ B	E	B
* C	A	D
*D	A	A
E	E	F
*F	E	B
*G	A	D
*H	E	F

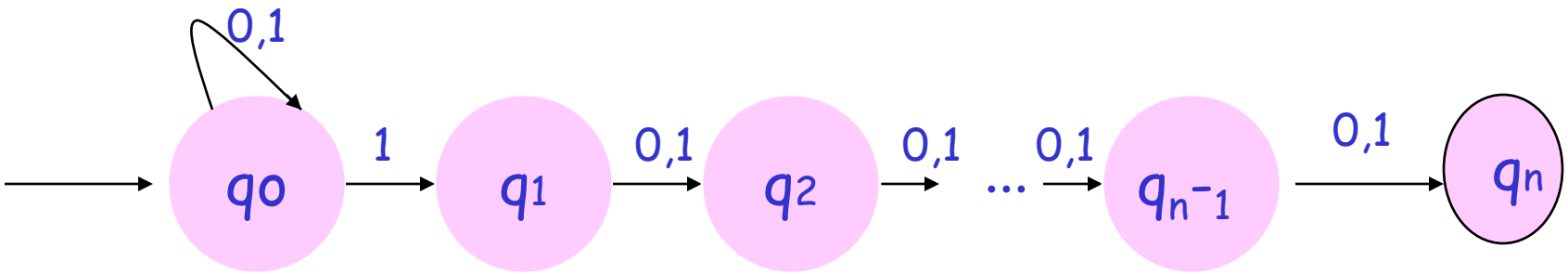
Repare que os únicos estados acessíveis a partir de B são: B, E e F. Os demais são inacessíveis e não precisam constar da tabela.

AFD resultante com n=3 estados



Há casos, no entanto, em que o AFD correspondente não pode ter menos estados do que 2^n, sendo n o número de estados do AFND correspondente.

Verifique para o seguinte AFND:



Aceita cadeias cujo n-ésimo símbolo, a partir da direita, é 1.

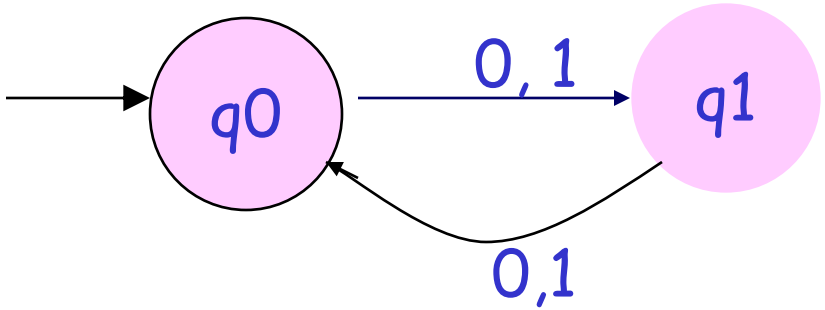
Cuidado com os estados de aceitação!

Muitas vezes, ao construir AFD, impedimos que ele aceite cadeias válidas.

Por outro lado, ao construir AFND, as vezes fazemos com que eles aceitem cadeias que não deveriam aceitar.

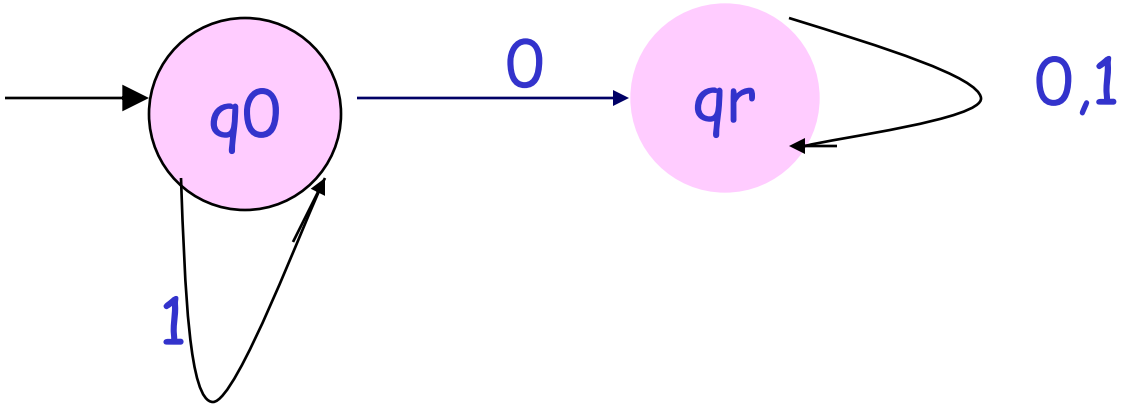
Exemplo

- Seja um AFD A que aceita cadeias sobre {0,1}* de comprimento par



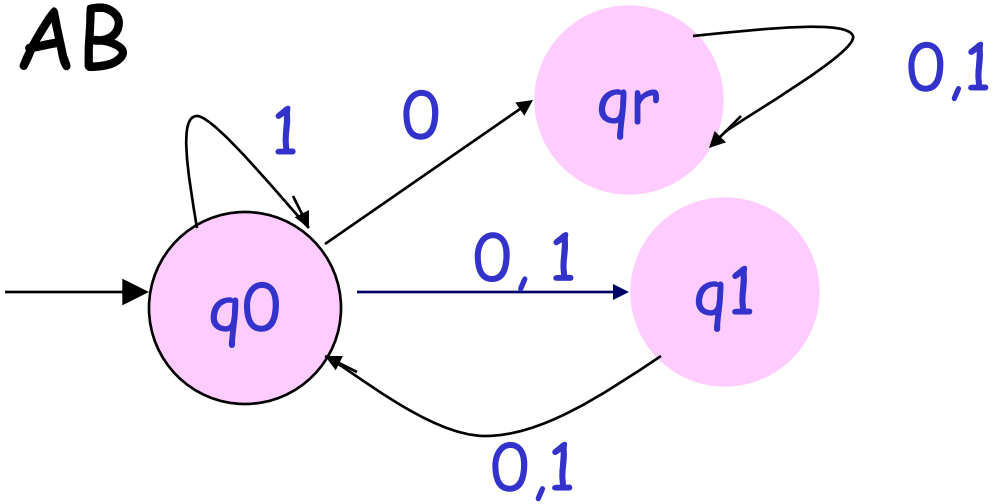
$L(A) = \{(0+1)^{2n} \mid n=0,1,\ldots\}$

- Seja um AFD B que aceita cadeias sobre {0,1}* que não contêm 0´s



L(B)= 1*

- Coloque A e B juntos, formando um AFND AB



- Poderíamos esperar que AB aceitasse a linguagem união das duas anteriores.
- Mas vemos que ela aceita não apenas essa união, mas qualquer cadeia exceto aquelas com número ímpar de 0´s e nenhum 1.
- $L(AB)=\{0,1\}^* - \{0^{2n+1} \mid n \geq 0\}$

Estados de não-aceitação em AFD

- A definição estrita de um AFD EXIGE que todo estado tenha uma transição para cada símbolo de entrada.
- Se seguirmos esta definição muitos exemplos vistos na seção de AFD não eram 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.

AFD com estado de não-aceitação

JFLAP : <untitled3>

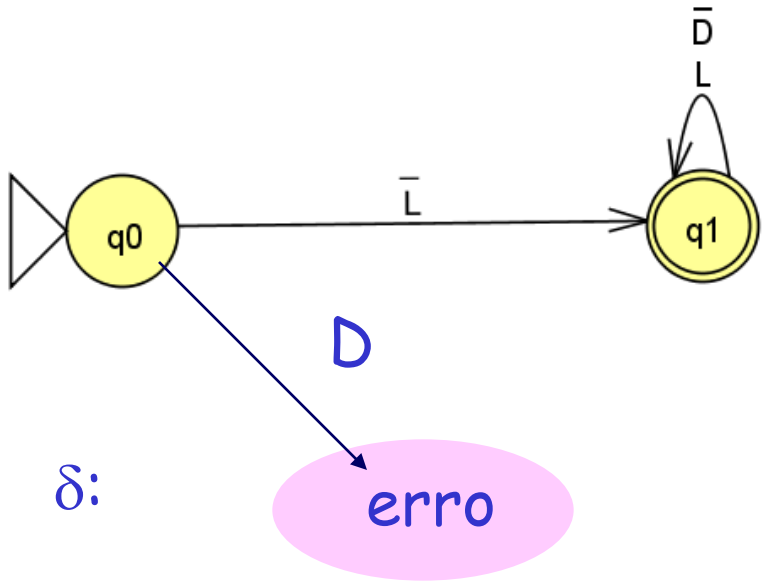
File Input Test Convert Help

Editor | Multiple Inputs

Input	Result
_LLL	Accept
LDD	Accept
DD	Reject
LLL	Accept
D+D	Reject

Run Inputs | Clear | Enter Lambda

$M = (\{q0, q1, erro\}, \{A..Z, a..z, 0..9, \_\}, q0, \delta, \{q1\})$



AF que reconhece identificadores em Pascal:

Letra seguida de Letra ou Dígito:

L (L+D)*

Iniciar | SCE_521_185 | Microsoft PowerPoint ... | JFLAP : <untitled3> | PT | 10:51

AF e Gramática Regular

Teorema 2: As linguagens aceitas por AF são exatamente aquelas geradas por Gramáticas Regulares.

a) Para cada GR existe um AF tal que L(GR) = L(AF)

b) Para cada AF existe uma GR tal que L(AF) = L(GR)

- ***Prova:***

a) Para cada GR existe um AF tal que L(GR) =L(AF)

- Se $A \in Vn$ então $f(q_A, a) = q_B$ sempre que $A \rightarrow aB$ (não determinismo: $A \rightarrow aB$; $A \rightarrow aC$):
- Se $A \rightarrow a$ então $f(q_A, a) = q_{final}$
- Se $S \rightarrow \lambda$ então $F = \{q_{final}, q_S\}$

Assim: $Q = Vn \cup \{q_{final}\}$

$\Sigma = Vt$ {Voca

$q_0 = q_S$

$F = \{q_S, q_{final}\} \Leftrightarrow S \rightarrow \lambda$

ou $F = \{q_{final}\}$, c.c.

$f(q_{final}, a) = q_{rej}$; $\forall\ a \in Vt$

b) Para cada AF existe uma GR tal que L(AF) = L(GR)

Para cada estado $q_i \in Q$, criamos um $Q_i \in Vn$;

Se $f(q_i, a) = q_j$ então $Q_i \rightarrow aQ_j$

Se $f(q_i, a) = q_j$ e $q_j \in F$ então $Q_i \rightarrow a$

$\therefore$ $Vn = Q$; $Vt = \Sigma$; $S = q_0$

Exemplo

- Ex.: GR ➔ AFND ➔ AFD ➔ GR'

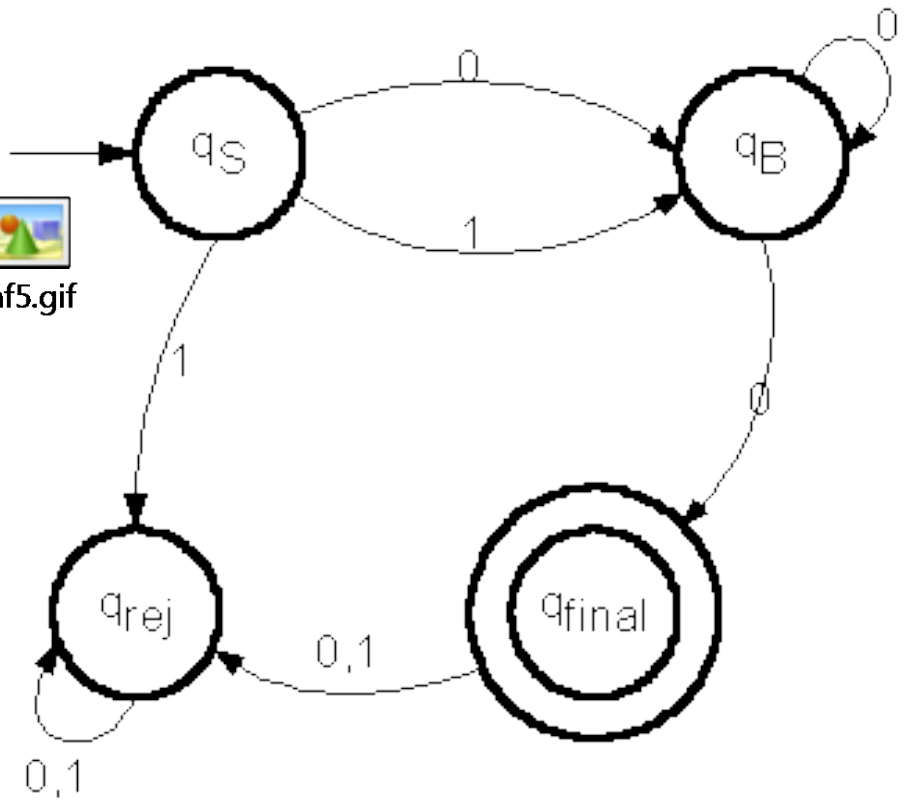
  $G = (\{S, B\}, \{0, 1\}, P, S)$

  $S \rightarrow 0B$

  $B \rightarrow 0B \mid 1S \mid 0$

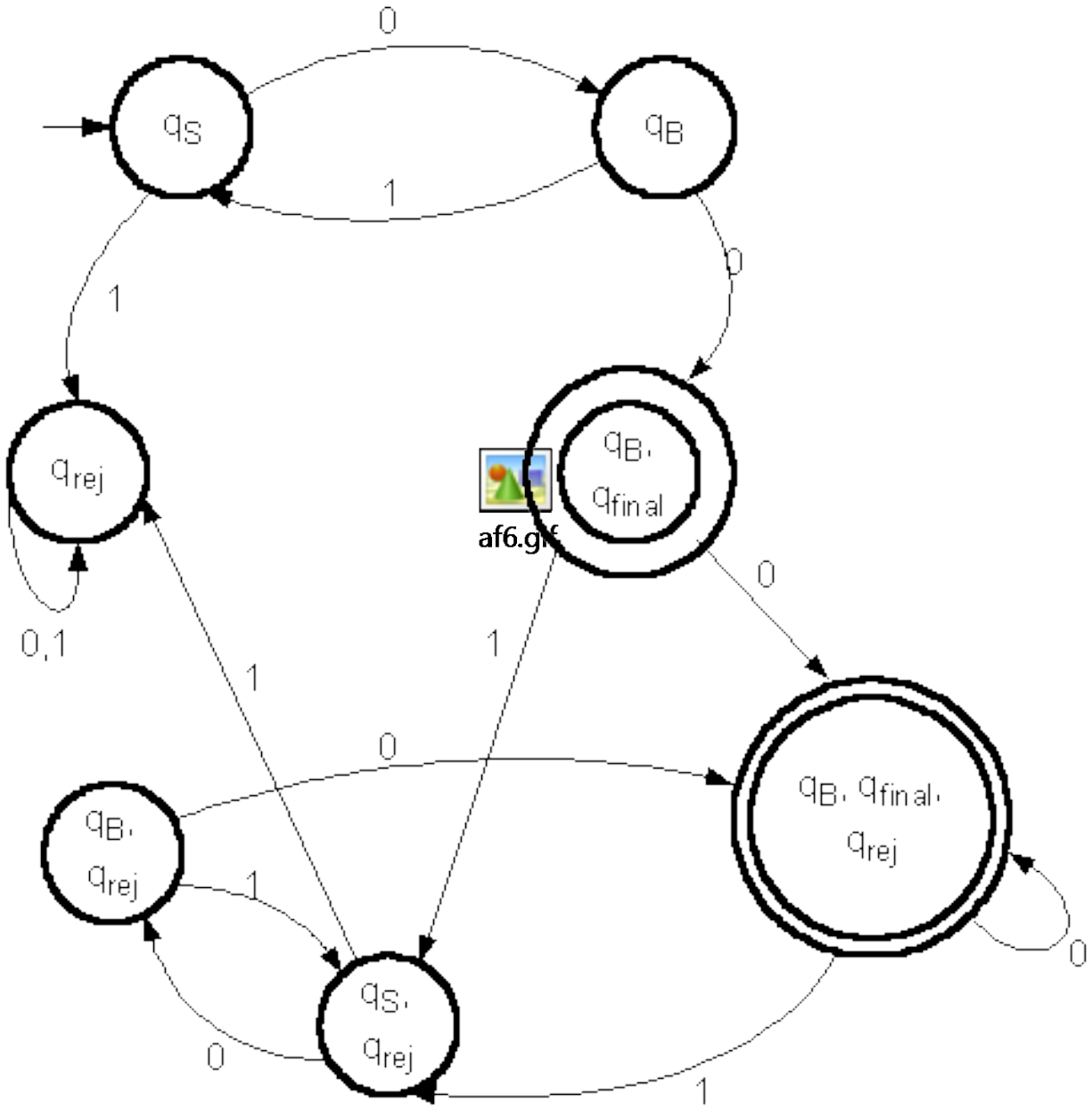
$L(G) = 00^+ \mid (0^+1)^+0^+$

af5.gif



- T2(a): GR ➔ AFND

- T1: AFND ➔ AFD


0
1
1
0
0,1
1
1
0
0
1
0
1
0
1
q_S
q_B
q_{rej}
q_B, q_{final}
q_B, q_{rej}
q_S, q_{rej}
q_B, q_{final}, q_{rej}
af6.gif

- T2(b): AFND ➔ GR´ (não se leva em conta o estado de rejeição)
- GR': S ➔ 0B
  B ➔ 0B | 0 | 1S
  B ➔ 0B | 0 (repetido)

$\therefore$ L(GR) = L(GR')

Um AF não é um programa.....

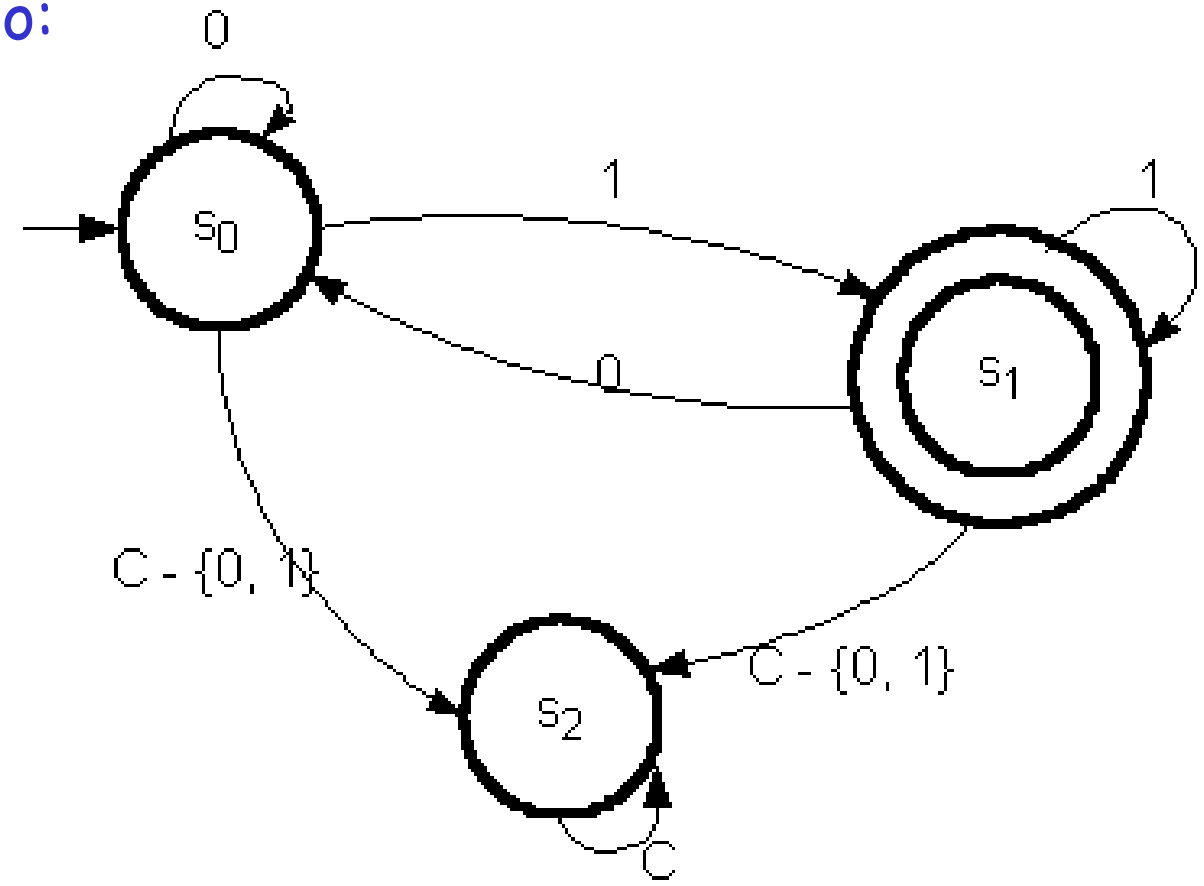
....mas serve como um bom modelo. Vejamos como podemos implementar um AFD:

- de forma direta: escrevendo um código que simule as transições.
- controlado por tabela: simulador universal de AFD

•(veja "Teoria dos Autômatos", "Autômatos Finitos", Implementação de Autômatos Finitos" em http://www.icmc.usp.br/~gracan/teoria/Teoria.html)

Implementando Autômatos Finitos por Código Direto

Exemplo:



C: conjunto dos caracteres

```
program AF1;
    type estado = (s0, s1, s2);
    var s: estado;
        c: char;
        F: set of estado;
function g(s:estado;c:char):estado;
    begin
      case s of
        s0: case c of
          '0': g:=s0;
          '1': g:=s1;
          else g:=s2
        end;
        s1: case c of
          '0': g:=s0;
          '1': g:=s1;
          else g:=s2
        end;
      end
    end; {g}
```

```
begin {progr. pc.}
   F:= [s1];
   s:= s0;
   while not eof do
    begin
      read(c);
      s:= g(s,c)
    end;
    if s in F then writeln('Cadeia aceita')
    else writeln('Cadeia rejeitada')
    end.
```

Método direto: específico, eficiente; cresce com o número de estados. É necessário modificar o programa para cada alteração no autômato.

Controle por Tabela de Transição

Apl_Trans:

Transição	Caracter lido	Próximo Estado
1	{0}	s0
2	{1}	s1
3	C - {0, 1}	s2
4	{0}	s0
5	{1}	s1
6	C - {0, 1}	s2
7	C	s2

Prim_Trans:

Estado	Transição
s0	1
s1	4
s2	7

Interpretador de tabelas "universal"

```
function g(s:estado, c:char): estado;
var t: 1 .. Ntrans;

        begin
            t := Prim_trans[s];
            while not(c in Apl_trans[t]. Conjcar)
                do t := t + 1;
            g := Apl_trans[t].prox_estado
        end;
```

Resumo Parcial

AFD: conjunto finito de estados e conjunto finito de símbolos de entrada. Uma função determina como o estado se altera toda vez que um símbolo é processado.

Linguagem de um AF: Um AF aceita cadeias. Uma cadeia é reconhecida se, começando no estado inicial, as transições levam a um estado de aceitação.

AFND: difere do AFD pelo fato de que pode ter qualquer número de transições (inclusive zero) para os estados seguintes, a partir de um dado estado e de um dado símbolo de entrada.

Construção de subconjuntos: tratando conjuntos de estados de um AFND como estados de um AFD, é possível converter qualquer AFND em um AFD que aceite a mesma linguagem.