



LFTC - Aula 01

Apresentação

Horário de aula - Laboratório 10 - discente 1

- terça-feira: 16h00-17h40
- quarta-feira: 16h00-17h40

Celso Olivete Júnior

celso.olivete@unesp.br



Professor Celso Olivete Júnior

- Bacharelado em Ciência da Computação (Unoeste-2002)
- Mestrado e Doutorado em Engenharia Elétrica - Área: Visão Computacional (USP-SC-2005/2009)
- Áreas de interesse e atuação:
 - Visão Computacional
 - Processamento de Imagens Médicas
 - Desenvolvimento para Web e Dispositivos Móveis
 - Compiladores



A disciplina LFTC

- Linguagens Formais e Teoria da Computação é uma disciplina fundamental dos cursos superiores da área de computação, especialmente daqueles que apresentam ênfase na formação científica do aluno, como é o caso dos cursos de **bacharelado em Ciência da Computação** e de vários cursos de Engenharia de Computação. Ela faz parte do núcleo denominado "Fundamentos da Computação" (conforme o currículo de referência da Sociedade Brasileira de Computação - www.sbc.org.br);



A disciplina LFTC

- Áreas de Aplicação relacionadas com o aprendizado adquirido nesta disciplina:
 - Inteligência Artificial
 - Gramáticas
 - Autômatos Finitos (Linguagens Regulares)
 - Compiladores e Interpretadores
 - Linguagens Livres de Contexto
 - Linguagens Sensíveis ao Contexto...



A disciplina LFTC

- utilização na disciplina de Compiladores

```
public class LFA {  
    public static void main(String[] args) {  
        int variavel = 20;  
        for (int i = 0; i < 10; i++) {  
            System.out.println("Alo mundo 10 vezes!!!!");  
        }  
        System.out.println( variavel );  
        System.out.println( variavel2 );  
        System.out.println( "Fim do alo mundo!!!!");  
    }  
}
```



Programa

1. Introdução e conceitos básicos

2. Expressões regulares e linguagens

3. Propriedades das linguagens regulares

4. Gramáticas e Linguagens

5. Autômatos Finitos

6. Autômatos Finitos com Pilhas

7. Máquinas Universais

8. Computabilidade



Objetivo da disciplina LFTC

- Dar ao aluno noção formal de algoritmo, computabilidade (decidibilidade) e dos formalismos para definição de linguagens (ou problemas), de modo a conscientizá-lo dos limites teóricos da ciência da computação.



Avaliação

- A cada bimestre

- Uma prova: NP
- Trabalhos e projetos: MT
- $MB = (7*NP + 3*MT)/10$ SE E SOMENTE SE ($NP \geq 5$ E $MT \geq 5$)

- Caso contrário ($MT < 5$ OU $NP < 5$)

- $MB = \text{Menor Nota (NP ou MT)}$

- Onde:

- NP = Nota da prova
- MT = Média dos trabalhos
- MB = Média do Bimestre

- A nota final (NF) do aluno no curso será a média das notas obtidas nos 2 bimestres
- Caso o aluno não obtenha a nota mínima para aprovação, será oferecida uma terceira avaliação (exame)



Avaliação

- A “**cola**” ou plágio em provas, exercícios ou atividades práticas implicará na atribuição de nota zero para todos os envolvidos. Dependendo da gravidade do incidente, o caso será levado ao conhecimento da Coordenação e do Conselho do Departamento, para as providências cabíveis. Na dúvida do que é considerado cópia ou plágio, o aluno deve consultar o professor antes de entregar um trabalho.



Projeto (em duplas)

❖ Parte 1: especificar e simular autômatos finitos através de diagramas de transições; especificar e simular expressões regulares e especificar e simular gramáticas regulares



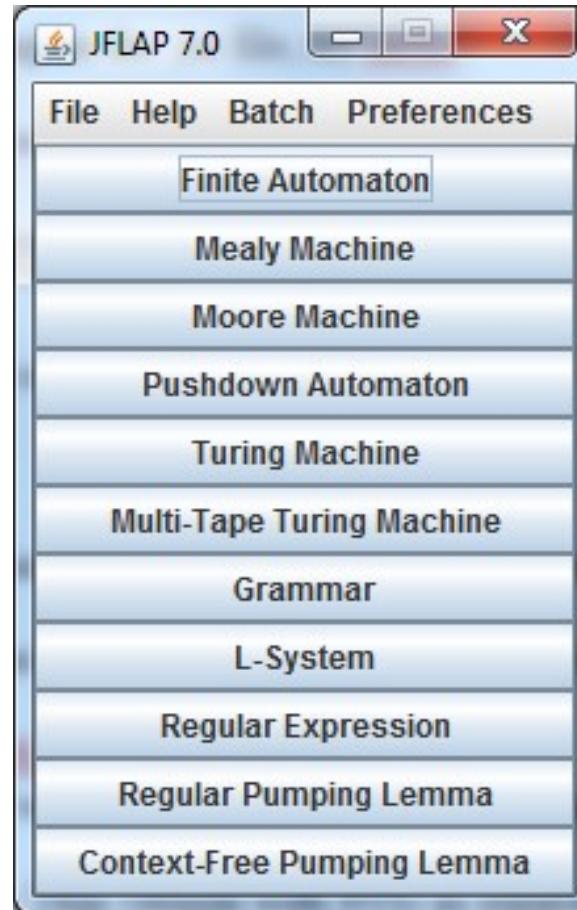
Projeto (em duplas)

❖ Parte 2:

❖ especificar e simular máquinas de Turing através de diagramas de transições



Ferramenta exemplo: JFlap



<http://www.jflap.org/>



Visão Geral da Disciplina



Linguagem Formal

- Uma Linguagem Formal possui:
 - **Sintaxe bem definida:** Dada uma sentença, é possível sempre saber se ela pertence ou não a uma linguagem;
 - **Semântica precisa:** De modo que não contenha sentenças sem significado ou ambíguas;

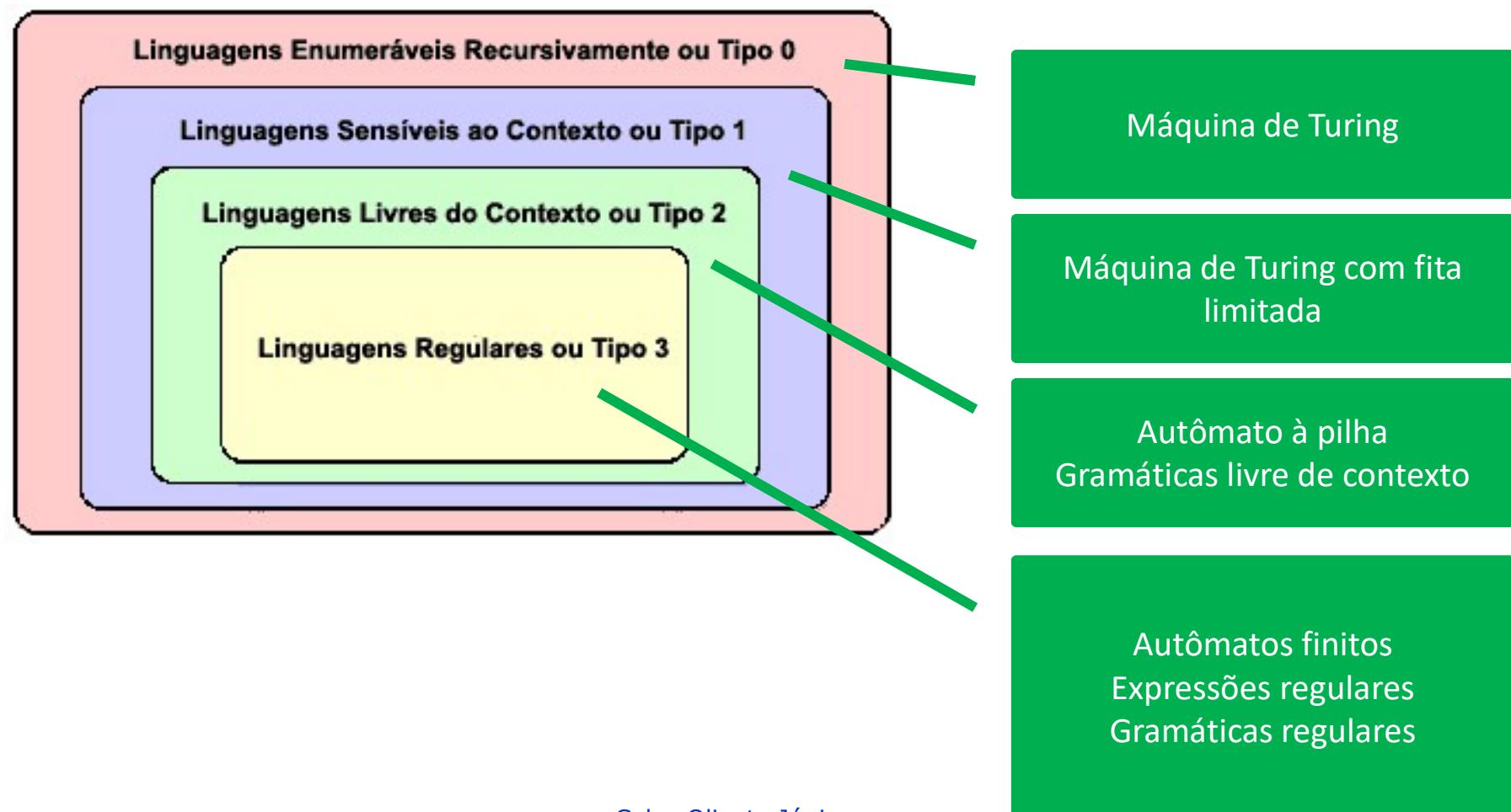


Linguagem Formal: exemplos na Computação

- Java, C, Pascal, HTML, Basic, C#, VB.net,...
- Ao projetar um sistema, o programador precisa estabelecer uma linguagem formal de comunicação com o usuário final - **linguagem de comunicação complexa = sistema mal projetado**

Fundamental em
COMPILEDORES

Classificação das Linguagens considerando a Hierarquia de Chomsky

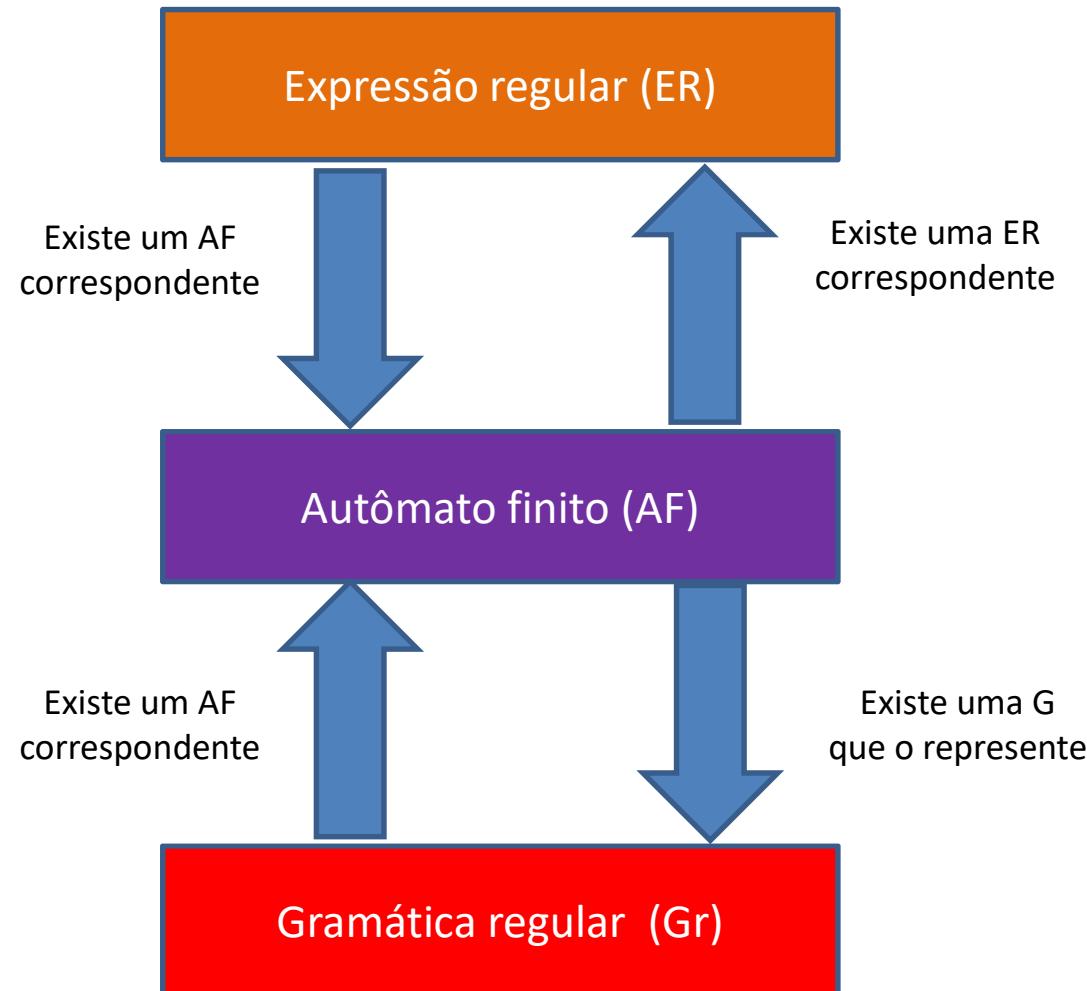




Linguagem

- Um linguagem é um conjunto de strings;
- Uma string é uma sequência de símbolos;
- Estes símbolos estão definidos em um alfabeto finito;
 - Ex: Linguagem C ou Pascal, etc.
- Linguagem regular - Formas de representação:
expressão regular (**ER**), autômato finito (**AF**) e
gramática regular (**GR**)
 - **ER:** Servem para verificar se uma string está ou não em uma linguagem

Equivalências entre linguagens regulares





Visão geral - Expressão regular (ER)

- Uma ER é definida a partir de conjuntos básicos e operações de **concatenação** e **união** e oferece um modo declarativo de expressar as cadeias que queremos aceitar.
- desenvolvidas a partir de :
 1. símbolos do alfabeto Σ
 2. operadores de:
 - **união** : representado por \cup ou $+$ ou $|$ \rightarrow conjunto de cadeias que está em um conjunto **ou** em outro
 - **concatenação**: representado por \cdot ou sem ponto \rightarrow junção dos elementos
 - **fecho-estrela (fechamento)**: representado por $*$ \rightarrow cadeias que podem ser formadas tomando qualquer número de repetição do elemento do conjunto (inclusive nenhuma vez)
 3. parênteses



Visão geral - Expressão regular

- Exemplos

ER	Linguagem reconhecida
aa	??
ba*	??
(a+b)	??
(a+b)*	??
(a+b)*aa(a+b)*	??
a*ba*ba*	??
(a+b)*(aa+bb)	??

Simulador de ER

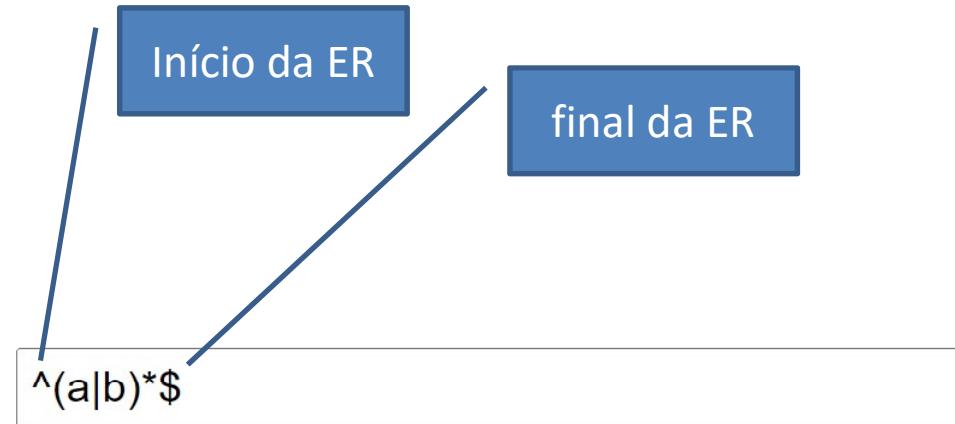
http://tools.lymas.com.br/regexp_br.php#



Visão geral - Expressão regular (ER)

• Exemplos

Expressão



Texto 1

aaaaaa

Texto 2

abac

Simulador de ER

http://tools.lymas.com.br/regexp_br.php#



Visão geral - Expressão regular

• Exemplos

ER	Linguagem reconhecida
aa	somente a palavra aa
ba*	iniciam com b seguido por 0 ou mais ocorrências de a
(a+b)	formada por a ou b
(a+b)*	formada por qualquer quantidade de a ou b , inclusive nenhuma vez
(a+b)*aa(a+b)*	??
a*ba*ba*	??
(a+b)*(aa+bb)	??

Simulador de ER

http://tools.lymas.com.br/regexp_br.php#



Visão geral - Expressão regular

• Exemplos

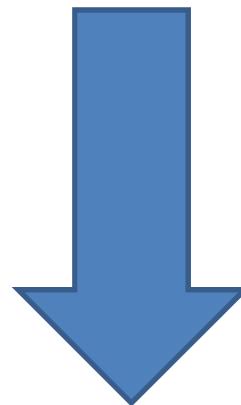
ER	Linguagem reconhecida
aa	somente a palavra aa
ba*	iniciam com b seguido por 0 ou mais a
(a+b)	formada por a ou por b
(a+b)*	formada por qualquer quantidade de a ou b, inclusive nenhuma vez
(a+b)*aa(a+b)*	contém aa como elemento
a*ba*ba*	todas as palavras contendo exatamente 2 b's
(a+b)*(aa+bb)	terminam com aa ou bb

Defina uma ER que reconheça:

- um identificador em Java
- um número inteiro



Visão geral - Expressão regular: atuam na especificação de uma linguagem



Autômatos finitos: é um formalismo que pode convertido em um programa de computador



Visão Geral - Autômatos finitos são formados por:

- ❖ Um conjunto finito de estados
- ❖ Arestas levando de um estado a outro, anotada com um símbolo
- ❖ Um estado inicial
- ❖ Um ou mais estados finais
- ❖ Normalmente os estados são numerados ou nomeados para facilitar a manipulação e discussão



Visão Geral - **Autômatos finitos**: constituem um modelo útil para muitos elementos de hardware e software. Ex:

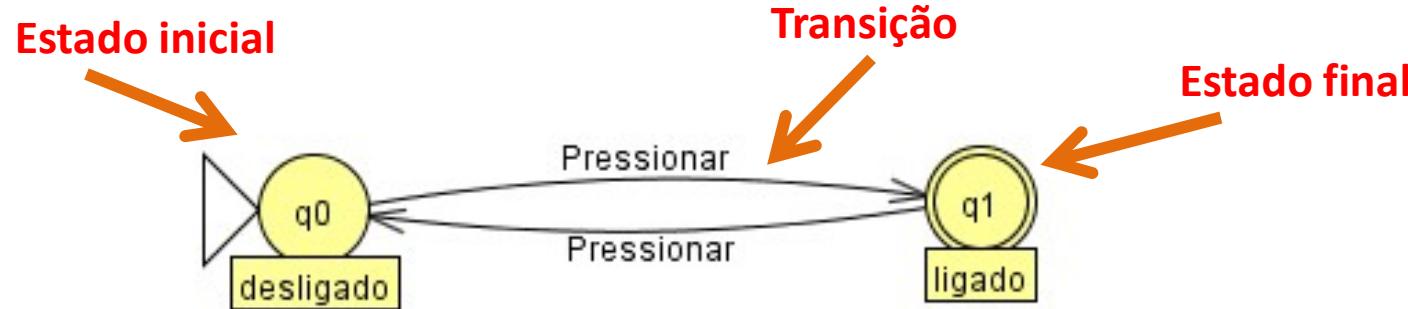
1. Controlar um interruptor: dispositivo para acender ou apagar uma lâmpada
2. software para projetar e verificar o comportamento de circuitos digitais;
3. analisador léxico de um compilador
4. descrever o processo de reconhecimento de padrões em cadeias de entrada, e assim podem ser utilizados para construir sistemas de busca;



Autômatos finitos: o que é e o que ele faz?

- Em cada um dos exemplos citados, podemos citar o autômato como *estando em um de um número finito de “estados”*
 - O estado permite “lembrar” o passado relevante do sistema (ele chegou lá após um certo número de transições/passos, que são o seu passado) e para onde pode seguir.
 - Se o número de estados de um sistema é finito, pode-se representá-lo com uma quantidade *limitada* de recursos.

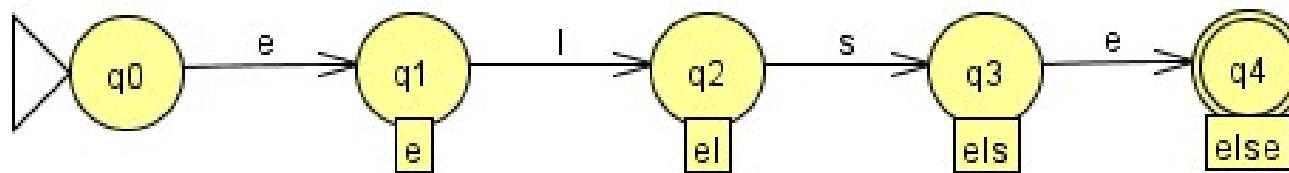
Autômato finito: exemplo clássico do interruptor



- o dispositivo memoriza se está no estado “ligado” ou no estado “desligado”, e permite ao usuário pressionar um botão diferente do estado atual



Autômato finito: parte de um analisador léxico -
reconhece a palavra reservada **else**

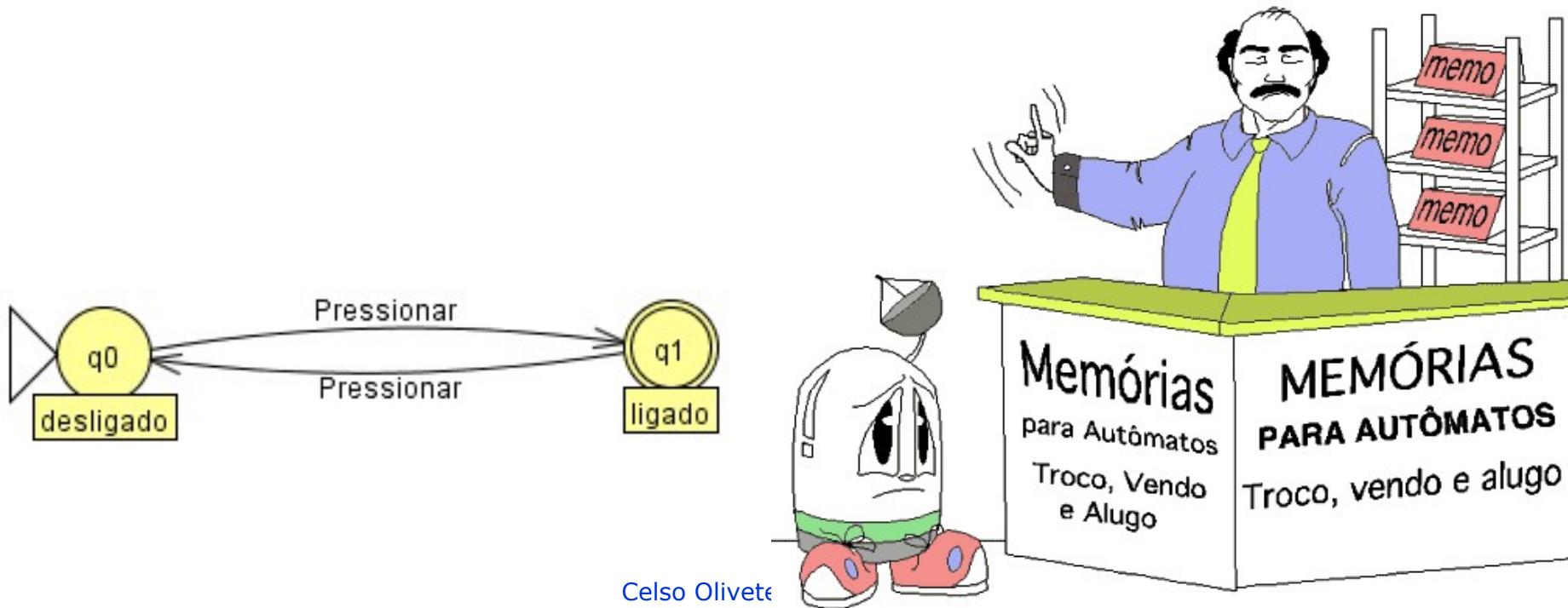


- cada estado memoriza o que já foi reconhecido - da palavra vazia até a palavra completa **else**
- parte do analisador léxico, representado pelo autômato, examina letra a letra do arquivo fonte que está sendo compilado até atingir o estado q4 (reconhece o **else**)

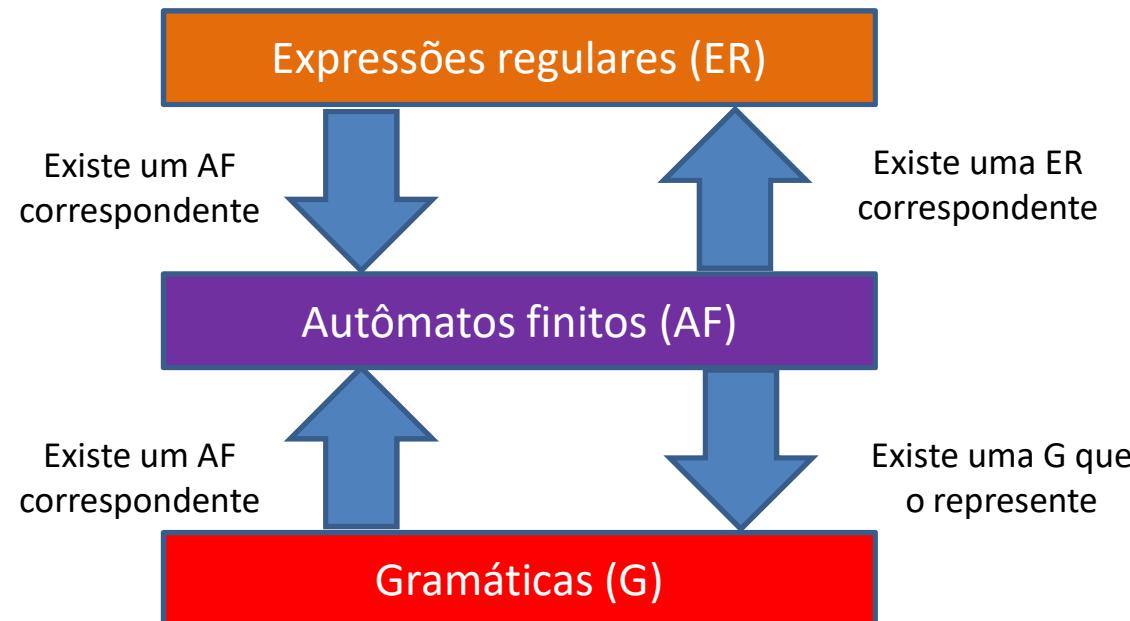


Autômatos finitos: principais características

- Memória Limitada.
- Trabalha apenas sobre o estado atual.
- Memória limitada pela quantidade de estados.



Autômatos finitos (AF) versus Expressões Regulares (ER) - como transformar?



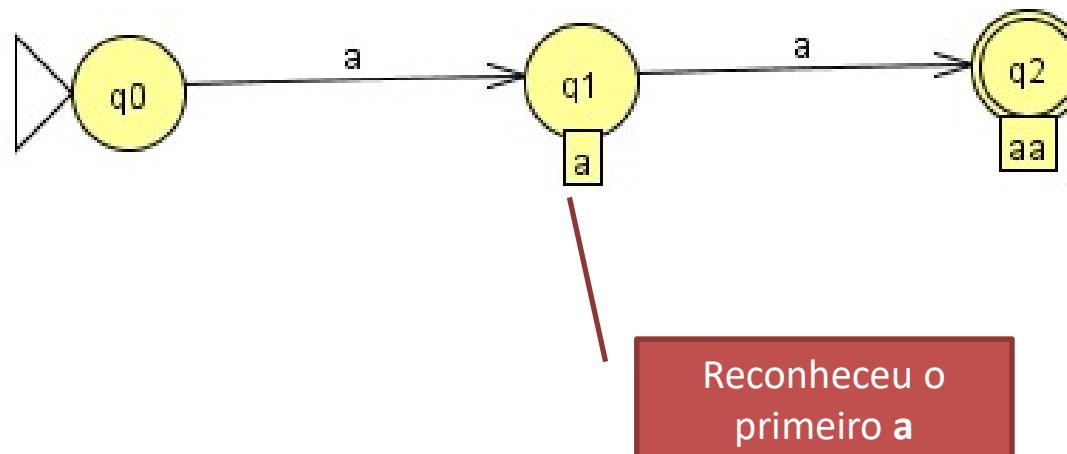


Autômatos finitos (AF) versus Expressões Regulares (ER) - como transformar?

ER	Linguagem gerada
aa	somente a palavra aa

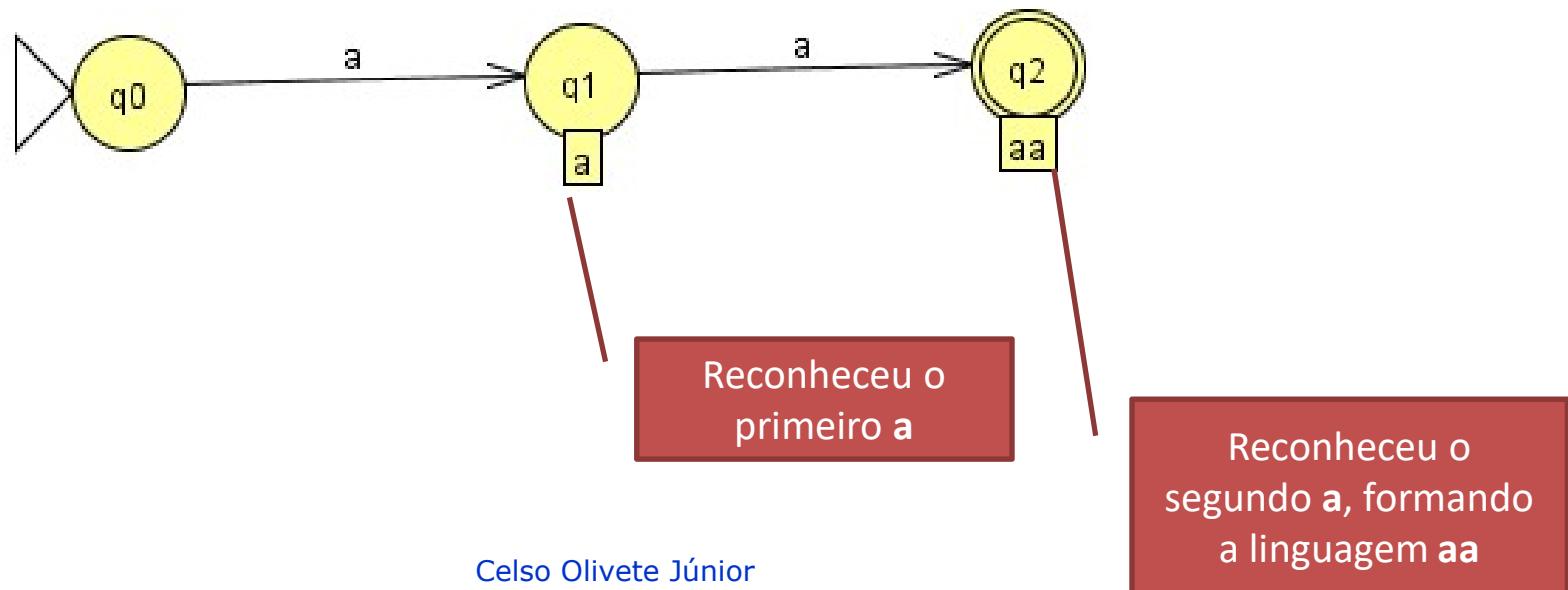
Autômatos finitos (AF) versus Expressões Regulares (ER) - como transformar?

ER	Linguagem gerada
aa	somente a palavra aa



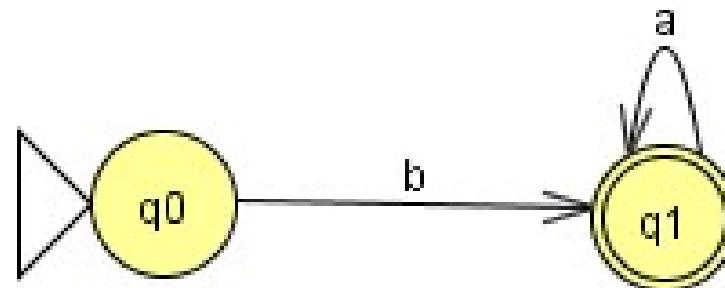
Autômatos finitos (AF) versus Expressões Regulares (ER) - como transformar?

ER	Linguagem gerada
aa	somente a palavra aa



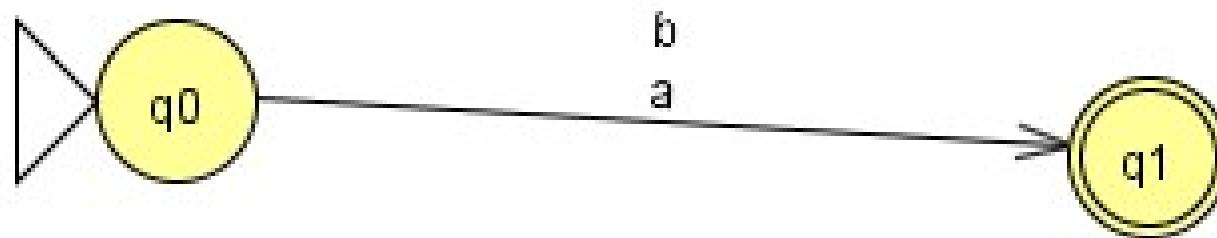
Autômatos finitos (AF) versus Expressões Regulares (ER) - como transformar?

ER	Linguagem gerada
ba*	iniciam com b seguido por 0 ou mais a



Autômatos finitos (AF) versus Expressões Regulares (ER) - como transformar?

ER	Linguagem gerada
$(a+b)$	formada por a ou por b





Visão Geral - Gramática

- A gramática é um formalismo projetado para a definição de linguagens
- Uma gramática mostra como gerar as palavras de uma linguagem
- Um elemento fundamental das gramáticas é denominado regra



Visão Geral - Gramática

- Exemplo de regra

$$\begin{array}{ll} S \rightarrow aB & S \rightarrow Ba \\ & \text{ou} \\ B \rightarrow b & B \rightarrow b \end{array}$$

- qual a linguagem gerada?
- ideia: “substituia” o símbolo que está em maiúsculo no lado direito da regra por seu significado correspondente encontrado no lado esquerdo:
 - B pode ser substituído por b
 - logo a linguagem gerada é representada pela **ER = ab ou ba**

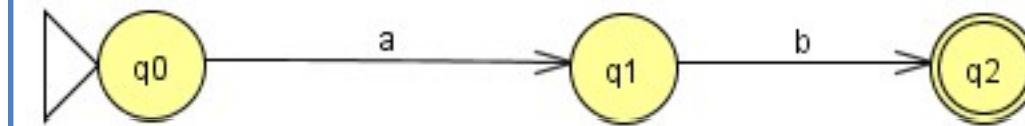
Visão Geral - Gramática

$S \rightarrow aB$

$B \rightarrow b$

- linguagem gerada é representada pela ER = ab

AF correspondente





Gramática:

- Exemplo 2

$$S \rightarrow aS \mid bA \mid \epsilon$$

$$A \rightarrow c$$

- qual a linguagem gerada?



Gramática:

- Exemplo 2

$S \rightarrow aS \mid bA \mid \epsilon$

$A \rightarrow c$

AF correspondente

- qual a linguagem gerada?
 - qualquer quantidade de a ou bc
- $ER = a^* + bc$



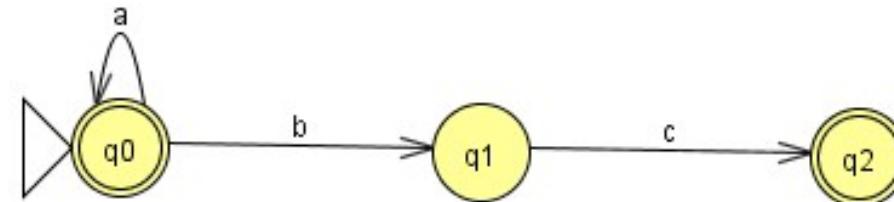
Gramática:

- Exemplo 2

$S \rightarrow aS \mid bA \mid \epsilon$

$A \rightarrow c$

AF correspondente



- qual a linguagem gerada?
 - qualquer quantidade de a ou bc
- $ER = a^* + bc$



Gramática: aplicação em compiladores:
reconhecimento de números inteiros com
sinal

$\text{Int} \rightarrow +\text{Dig} \mid -\text{Dig}$

$\text{Dig} \rightarrow 0\text{Dig} \mid 1\text{Dig} \mid \dots 9\text{Dig} \mid 0 \mid 1 \mid \dots 9$



Gramática: regras que definem o comando WHILE (Pascal)

```
comando      → comandoWhile
comandoWhile → WHILE expr_bool DO comando;
expr_bool    → expr_arit < expr_arit
              | expr_arit > expr_arit
              | ...
expr_arit   → expr_arit * termo
              | termo
              | ...
termo        → expr_arit
              | NÚMERO
              | IDENTIFICADOR
```



Próxima aula:

- Linguagens regulares:
 - Alfabeto
 - String
 - Concatenação
 - Comprimento de uma cadeia
 - Operações: união, concatenação, reverso, fechamento....
 - Expressões regulares
- Projeto:
 - Definição dos grupos





LFTC - Aula 02

Linguagens regulares - introdução

Celso Olivete Júnior

celso.olivete@unesp.br

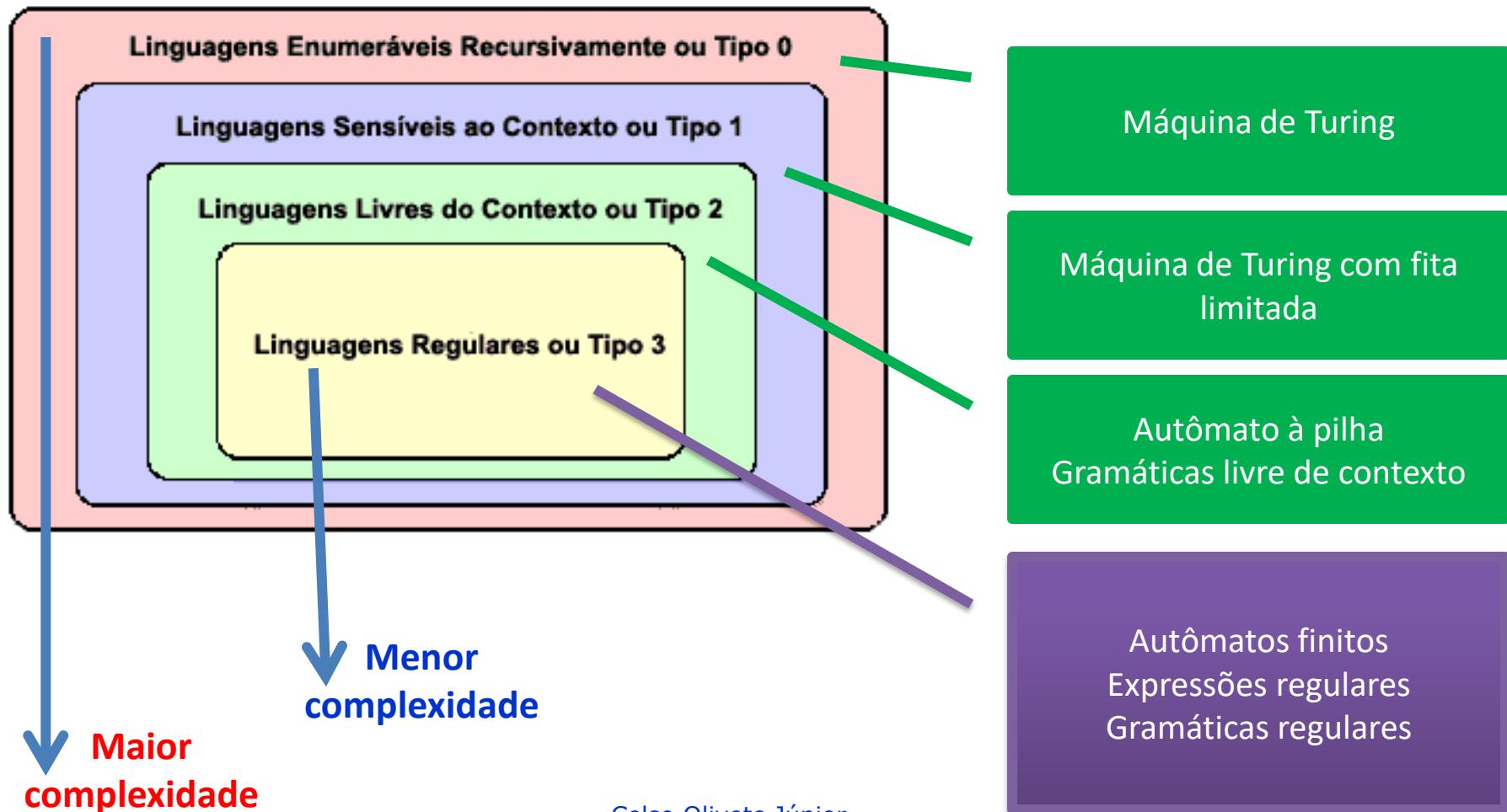


Na aula passada...

- Visão geral
- Linguagens regulares
 - expressões regulares
 - autômatos finitos
 - gramáticas regulares



Classificação das Linguagens segundo Hierarquia de Chomsky e seus reconhecedores





Na aula de hoje...

- Linguagens regulares: Expressões regulares
- Referência bibliográfica

HOPCROFT, J. E.; ULLMAN, J. D.; MOTWANI, R. *Introdução à Teoria de Autômatos, Linguagens e Computação*. Editora Campus, 2002 → Capítulos 1 e 3



Roteiro

- Definições prévias: alfabeto, strings e linguagem
- Expressões regulares



Roteiro

- Definições prévias: alfabeto, strings e linguagem
- Expressões regulares



Definições prévias

- **Alfabeto ou Vocabulário:** Conjunto finito não vazio de símbolos.

Símbolo é um elemento qualquer de um alfabeto. Representado por Σ

Ex: $\Sigma = \{a,b\} \rightarrow$ alfabeto formado pelas letras a e b

$\Sigma = \{0,1,2,3,4,5,6,7,8,9\} \rightarrow$ alfabeto formado pelos dígitos de 0 a 9

- **Cadeia, String ou Palavra:** Concatenação finita de símbolos de um alfabeto. Define-se como cadeia vazia ou nula uma cadeia que não contém nenhum símbolo.

Ex: $aab \rightarrow$ string sobre o alfabeto $\Sigma = \{a,b\}$

$123094 \rightarrow$ string sobre o alfabeto $\Sigma = \{0,1,2,\dots,9\}$

ϵ ou λ representam uma cadeia vazia



Definições prévias

- **Comprimento de uma string:** Número de símbolos de uma cadeia.

Ex: $\Sigma = |aab| = 3$

$$\Sigma = |123094|=6$$

$$\Sigma = |\varepsilon|=0$$

- **Concatenação de string:** Define-se a concatenação z de uma cadeia x com uma cadeia y , como sendo a concatenação dos símbolos de ambas as cadeias, formando a cadeia xy . $|z| = |x| + |y|$

Ex: $x = abaa; \quad y = ba \quad \rightarrow z = abaaba$

$x = ba; \quad y = \varepsilon \quad \rightarrow z = ba$



Definições prévias

• **Produto de alfabetos:** É o produto cartesiano de alfabetos.

Ex: $V_1 = \{a, b\}$ $V_2 = \{1, 2, 3\}$ $\rightarrow V_1.V_2 = V_1 \times V_2 = \{a1, a2, a3, b1, b2, b3\}$

- Observe que $V_1.V_2 \neq V_2.V_1$

• **Exponenciação de alfabetos:** São todas as cadeias de comprimento n sobre V (V^n). $V^0 = \{\epsilon\}$, $V^1 = V$, $V^n = V^{n-1}.V$

Ex: $V = \{0, 1\}$

- $V^3 = V^2.V = (V.V).V = \{00, 01, 10, 11\}.\{0, 1\} = \{000, 001, 010, 011, 100, 101, 110, 111\}$



Definições prévias

- **Fechamento (Clausura) de um Alfabeto:** Seja A um alfabeto, então o fechamento de A é definido como $A^* = A^0 \cup A^1 \cup A^2 \cup \dots \cup A^n \cup \dots$

Portanto A^* = conjunto das cadeias de qualquer comprimento sobre o alfabeto a (inclusive nenhum).

Ex: $A = \{1\}$

$$A^* = \{\varepsilon, 1, 11, 111, \dots\}$$

- **Fechamento Positivo de A :** $A^+ = A^* - \{\varepsilon\}$



Definições prévias

- **Prefixo** de uma palavra é qualquer sequência inicial de símbolos de uma palavra
- **Sufixo** é qualquer sequência final de símbolos de uma palavra
- Relativamente à palavra **abcb**, tem-se que:
 - ϵ , a, ab, abc, abcb são os prefixos
 - ϵ , b, cb, bcb, abcb são os respectivos sufixos



Definições prévias

- **Subpalavra** de uma palavra é qualquer sequência de símbolos contígua da palavra
- Qualquer prefixo ou sufixo de uma palavra é uma subpalavra



Definições prévias: relembrando...

- **Linguagem formal:** é uma coleção de cadeias de símbolos, de comprimento finito. Estas cadeias são denominadas sentenças da linguagem, e são formadas pela justaposição de elementos individuais, os símbolos ou átomos da linguagem.



Linguagem formal

- Ex: $\{ab, bc\}$

linguagem formada pelas cadeias ab ou bc)

$$\{ab^n, a^n b; n \geq 0\}$$

linguagem formada por todas as cadeias que começam com "a" seguido de um número qualquer de "b"'s (exemplo a, ab, abb, aab, aaab,...) ou começam com um número qualquer de "a"'s seguidos de um "b"



Problemas

- Na teoria de LFTC, um problema é a questão de decidir se determinado string é um elemento de uma linguagem
- Se Σ é um alfabeto e L é uma linguagem sobre Σ , então o problema é:
 - dado um string w em Σ^* , definir se w está ou não em L .



Problemas

- Exemplo:

dado um **string w** em Σ^* ,
definir se **w** está ou não
em **L**.

Linguagem L: consiste em um número igual de 0's e 1's

string **w** = 01110 → **w** **não** é elemento de L

string **w** = 0110 → **w** **é** elemento de L



Formas de definir uma linguagem

1. Por meio da descrição do conjunto finito ou infinito de cadeias (**Formalismo Descritivo**)

Ex: Linguagem com quantidade par de elementos;

$$L = \{ 0^n 1^n \mid n \geq 0 \}$$

2. por meio de uma **Gramática/Expressão Regular que a gere** (**Formalismo Gerativo**)



Definindo uma linguagem

- É definida usando um “**formador de conjuntos**” ou **Formalismo Descritivo**:

{ w | algo sobre w }

- Essa expressão é lida como “o conjunto de palavras w tais que (seja o que for dito sobre w à direita da barra vertical)”. Exemplos:

{w | w consiste em um número igual de 0's e 1's}

{w | w é um número inteiro binário primo}

{w | w é um programa em C sintaticamente correto}



Definindo uma linguagem

- Também é comum substituir w por alguma expressão com parâmetros e descrever os strings na linguagem declarando condições sobre os parâmetros. Exemplo:

$$\{0^n 1^n \mid n \geq 1\}.$$

“o conjunto de 0 elevado a n 1 elevado a n tal que n maior ou igual a 1”;

essa linguagem consiste nos strings $\{01, 0011, 000111, \dots\}$.



Definindo uma linguagem: outro exemplo

$$\{0^i 1^j \mid 0 \leq i \leq j\}.$$

Essa linguagem consiste em strings com alguns 0's (possivelmente nenhum) seguidos por um número igual ou superior de 1's.



Formas de definir uma linguagem

1. Por meio da descrição do conjunto finito ou infinito de cadeias (**Formalismo Descritivo**)
2. por meio de uma Gramática/Expressão Regular que a gere (**Formalismo Gerativo**)



Roteiro

- Definições prévias: alfabeto, strings e linguagem
- Expressões regulares e linguagens



Expressões regulares

- é uma das formas de definir uma linguagem regular
- podem ser consideradas uma “linguagem de programação”
 - aplicação de pesquisas em textos
 - descrever componentes de software
- estão relacionadas com as gramáticas regulares (definição/geração de sentenças) e com os autômatos finitos (aceitação de sentenças).



Expressões regulares

- servem como uma linguagem de entrada para muitos sistemas de processamento de strings. Exemplos:



Expressões regulares

- comando de pesquisa `grep` do unix
 - O comando `grep` vai procurar um arquivo, texto ou qualquer outra entrada e retornar quaisquer linhas que contenham a string especificada. Digamos que você precisa procurar dentro de um arquivo de log por todos os erros que tenham a ver com o `mysql`.

`grep 'mysql' arquivo-de-log.log`



Expressões regulares

- como parte de um componente do compilador

- especificar:
 - forma de um identificador

$[a-zA-Z_][a-zA-Z_0-9]^*$

- de um número inteiro com ou sem sinal
- ...



Os **operadores** de expressões regulares (ER's)

- Como já foi dito, as ER's denotam linguagens regulares. Exemplo:
- $01^* \mid 10^*$: linguagem (L) que consiste em todas as strings que começam com um 0 seguido por qualquer quantidade de número 1 (inclusive nenhum) **OU** começam com um 1 seguido por qualquer quantidade de 0



Os operadores de expressões regulares

- Os tipos de operadores sobre as ER's são:
 - União
 - Concatenação
 - Fechamento



Os operadores de expressões regulares

• Operador união

- Denotado por |
- $L \mid M$ (Linguagem L união com a Linguagem M).

Corresponde a:

- Conjunto de strings que estão em L ou M, ou em AMBAS.

Exemplo: $L=\{001, 10, 111\}$ e $M=\{\epsilon, 001\}$.

$$L \mid M = \{\epsilon, 001, 10, 111\}$$



Os operadores de expressões regulares

• Operador concatenação

- Denotado por . (ponto) ou sem nenhum operador
- $L.M$ ou LM (Linguagem L concatenada com Linguagem M). Corresponde a:
 - Conjunto de strings que podem ser formados tomando-se qualquer string em L e concatenando-se com qualquer string em M .



Os operadores de expressões regulares

- Operador concatenação
 - Conjunto de strings que podem ser formados tomando-se qualquer string em L e concatenando-se com qualquer string em M.

Exemplo: $L=\{001, 10, 111\}$ e $M=\{\epsilon, 001\}$.

$LM = \{001\epsilon, 10\epsilon, 111\epsilon, 001001, 10001, 111001\}$.

Os 3 primeiros símbolos de LM é o resultado da concatenação de cada sequência de L com ϵ .



Os operadores de expressões regulares

- Operador **fechamento** (ou estrela, ou fechamento de Kleene)
 - Denotado por * (asterisco)
 - L^* (Fechamento sobre a Linguagem L).

Corresponde a:

- Conjunto de todos strings que podem ser formados tomando-se qualquer número de strings de L (inclusive nenhum), possivelmente com repetições.



Os operadores de expressões regulares

- Operador fechamento (ou estrela, ou fechamento de Kleene)

Exemplo: $L = \{0, 11\}$

$$L^* = \{011, 11110, \dots \epsilon\}$$

Observe que L é infinito



Construindo expressões regulares

- É necessário algum método para **agrupar** os **operadores** com seus **operandos**, tais como **parênteses**.
- Por exemplo, a álgebra aritmética começa com constantes como inteiros e números reais, além de variáveis, e elabora expressões mais complexas com operadores aritméticos como **+** e *****



Construindo expressões regulares

- A álgebra de ER's segue esse padrão → usa constantes e variáveis que denotam linguagens e operadores para as 3 operações ($|$ união, $.$ concatenação e * fechamento)
- A base para esta definição consiste em:
 1. as constantes ε e \emptyset (vazio) são ER's, denotando as linguagens $L(\varepsilon) = \{\varepsilon\}$ e $L(\emptyset) = \{\emptyset\}$
 2. se a é qualquer símbolo, então a é uma ER, denotando a linguagem $L(a) = \{a\}$
 3. L (letra maiúscula e itálico) representa qualquer linguagem



Construindo expressões regulares

1. Se E e F são ER's, então $E \mid F$ é uma ER denotando a **união** de $L(E)$ e $L(F)$

$$L(E|F) = L(E) \mid L(F)$$

2. Se E e F são ER's, então EF é uma ER denotando a **concatenação** de $L(E)$ e $L(F)$

$$L(EF) = L(E)L(F)$$



Construindo expressões regulares

3. Se E é uma ER, então E^* é uma ER denotando o **fechamento** de $L(E)$

$$L(E^*) = (L(E))^*$$

4. Se E é uma ER, então (E) é uma ER denotando a mesma linguagem que E

$$L((E)) = L(E)$$



Exemplos de expressões regulares

Exemplos

ER formada por 01

ER = 01

ER para strings que são formadas por zero ou mais ocorrências de 01

ER = $(01)^*$
é diferente de 01^*

ER formada por 0's e 1's alternados

$(01)^* \mid (10)^* \mid 0(10)^* \mid 1(01)^*$
resultando, por exemplo, na sequência
010101 | 101010 | 01010 | 10101

Obs: cuidado com o uso dos parênteses!!



Exemplos de expressões regulares

Exemplos

ER para strings que são formadas por zero seguido por qualquer ocorrência de 1 (inclusive nenhuma)

$$ER = 01^*$$

ER formada por todas as palavras sobre (a,b) contendo **aa** como subpalavra

$$ER = (a|b)^*aa(a|b)^*$$



Expressões regulares: precedência de operadores → essencial na omissão de parênteses

Ordem de precedência em ER's

- | | |
|---|--|
| 1 | O operador fechamento (*) é o de precedência mais alta |
| 2 | Em seguida, o operador de concatenação (.) |
| 3 | Por último, todas as operações de união () |





Expressões regulares: precedência de operadores

• Exemplo: $ER = 01^* \mid 1$

• é agrupada como $(0 (1^*)) \mid 1$.

Agrupando...

- | | |
|---|---|
| 1 | O operador fechamento é realizado primeiro |
| 2 | Em seguida, o operador de concatenação 0 e (1^*)
$\rightarrow 0(1^*)$ |
| 3 | Por último, o operador de união (\mid) |

• Linguagem gerada \rightarrow todos os strings formados por 0 seguido por qualquer número de 1's (inclusive nenhum) OU 1

$$L = \{1, 0, 01, 011, \dots\}$$



Expressões regulares: exemplo de aplicação

- como componente léxico
 - Identificadores da linguagem Pascal que são compostos por letras (a.-zA-Z) ou underline(_) seguido por qualquer combinação de letras, underline ou dígitos (0...9)
- Supondo que:
 - letras são representadas por L
 - underline por _
 - e os dígitos por D
- ER = $L|_ (L|_|D)^*$



REGEXP - simulador de expressões regulares

http://tools.lymas.com.br/regexp_br.php#

Expressão	<input type="text" value="^(a b)*c\$"/>
Texto 1	<input type="text" value="aaaaaaabac"/>
Texto 2	<input type="text" value="c"/>
Ignorar Maiusc./minusc.	<input type="checkbox"/>

Considerações

- ❖ ER deve ser escrita **iniciando** com ^
- ❖ ER deve ser **finalizada** com \$

Manual

<http://www.gnu.org/s/libtool/manual/emacs/Regexp.html>



REGEXP - simulador de expressões regulares

ER que reconhece identificadores de uma linguagem de programação deve ser escrita da seguinte forma

$$^{\text{[a-zA-Z]}}[0-9a-zA-Z]*\$$$

TESTE DE REGEXP (EXPRESSÕES REGULARES)

Sync, share and backup, online or in LAN - encrypted, direct and independent



Expressão

Texto 1

Texto 2



Exercícios (em dupla)

1. Descreva as linguagens denotadas pelas ER's abaixo sobre o alfabeto $\Sigma = \{0,1\}$.

a- $0 \mid 10^*$

b- $(0 \mid 1)0^*$

c- $(0011)^*$

d- $(0 \mid 1)^* 1(0 \mid 1)^*$

e- 0^*11^*0

f- $0(0 \mid 1)^*0$

g- $(\varepsilon + 0)(\varepsilon \mid 1)$

h- $(000^* \mid 1)^*$

i- $(0^* \mid 0^*11(1 \mid 00^*11)^*) (\varepsilon \mid 00^*)$



Exercícios

2. Sobre o $\Sigma=\{a,b\}$, defina expressões regulares que representam as linguagens cujas sentenças estão descritas a seguir:

- Possuem comprimento maior ou igual a 3;
- Possuem comprimento menor ou igual a 3;
- Possuem comprimento diferente de 3;
- Possuem comprimento par;
- Possuem comprimento ímpar;
- Possuem comprimento múltiplo de 4.

3. Fazer o conjunto de exercícios da seção 3.1 do livro do HOPCROFT, páginas 96 e 97.

4. Faça um simulador para Expressões Regulares (qualquer linguagem de programação)



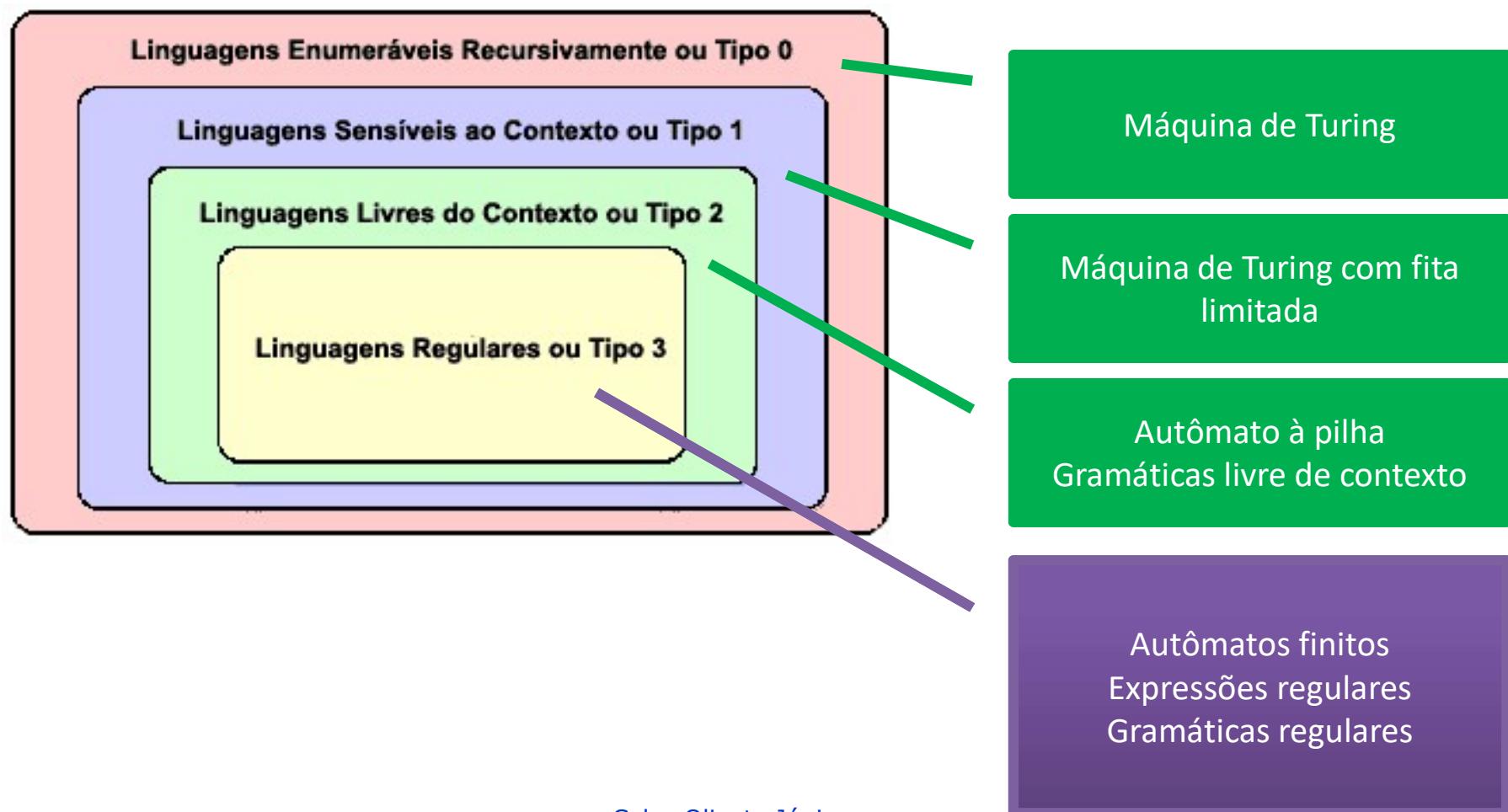
LFA - Aula 03

Linguagens regulares -
Gramáticas regulares

Celso Olivete Júnior
celso.olivete@unesp.br



Classificação das Linguagens segundo Hierarquia de Chomsky





Na aula passada...

- Expressões regulares



Na aula de hoje...

- Linguagens regulares: **Gramáticas regulares**
- Referência bibliográfica

RAMOS, M.V.M.; NETO, J.J.; VEGA, I.S. *Linguagens Formais: Teoria, Modelagem e Implementação*. Editora Bookman 2009. → Capítulo 3



Relembrando...

- Uma linguagem regular é o conjunto de linguagens reconhecida/gerada pelos seguintes formalismos:
 - Expressões regulares
 - Gramáticas regulares
 - Autômatos finitos



Os operadores de expressões regulares

- Os tipos de operadores sobre as ER's são:
 - União (|)
 - Concatenação (.)
 - Fechamento (*)



Exemplos de expressões regulares

Exemplos

ER para strings que são formadas por zero seguido por qualquer ocorrência de 1 (inclusive nenhuma)

$$\text{ER} = 01^*$$

ER formada por todas as palavras sobre (a,b) contendo **aa** como subpalavra

$$\text{ER} = (a|b)^*aa(a|b)^*$$



Relembrando...

- Uma linguagem regular é o conjunto de linguagens reconhecido/gerado pelos seguintes formalismos:
 - Expressões regulares
 - **Gramáticas regulares**
 - Autômatos finitos



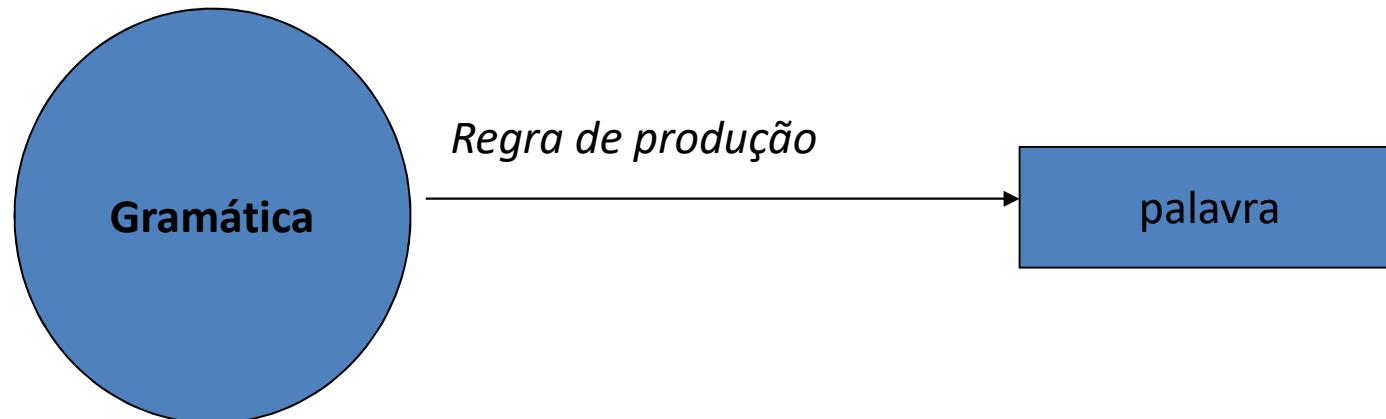
Gramática

- Uma **gramática** consiste em **uma ou mais variáveis que representam linguagens.**
- Exemplo: linguagem dos palíndromos (permite a mesma leitura da esquerda para direita quanto da direita para esquerda). Ex: **Anita latina**
 - Se w é um palíndromo, então $0w0$ e $1w1$ são palíndromos.
 - Neste caso a linguagem é formada apenas por uma variável (**w**).



Gramáticas

- Uma **gramática** consiste em uma ou mais variáveis que representam linguagens.





Gramáticas

- Formalmente as gramáticas são caracterizadas como quádruplas ordenadas

$$G = (\{V\}, \{T\}, P, S)$$

• onde:

- V representa o vocabulário não terminal da gramática - **variáveis**.
- T é o vocabulário **terminal**, contendo os símbolos que constituem as sentenças da linguagem.
- P representa o conjunto de todas as leis de formação (**regras de produção**) utilizadas pela gramática para definir a linguagem.
- S representa o símbolo de **íncio**



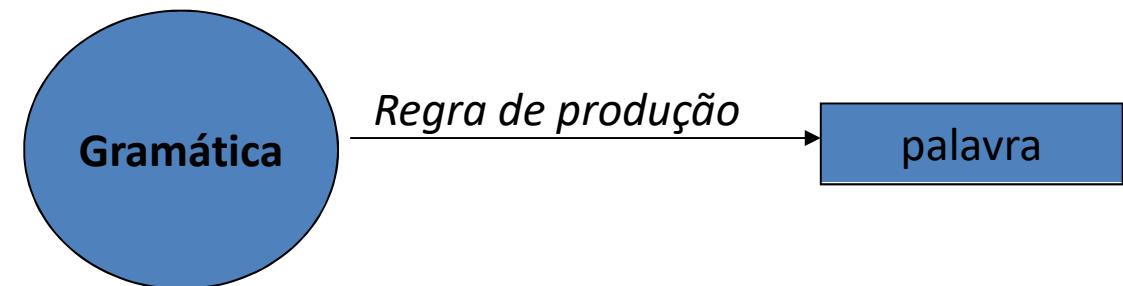
Gramáticas (não é gramática Regular!)

- $G = (\{S,A,B\}, \{a,b\}, P, S)$

- $P = \{ \quad S \rightarrow AB$

$$A \rightarrow a \mid AB$$

$$B \rightarrow b \}$$



- Passos para gerar a palavra: **abb**

- $S \rightarrow AB \rightarrow ABB \rightarrow aBB \rightarrow abB \rightarrow abb.$



Gramática no JFlap

```
Input:abb
String accepted! 13 nodes generated.
```

LHS	RHS
S	$\rightarrow AB$
A	$\rightarrow a$
A	$\rightarrow AB$
B	$\rightarrow b$

```

graph TD
    S((S)) --> A1((A))
    S((S)) --> B1((B))
    A1 --> a1((a))
    B1 --> A2((A))
    B1 --> B2((B))
    A2 --> b1((b))
    B2 --> b2((b))
    
```



Gramáticas

■ Notação / Convenções

- Variáveis: letras do alfabeto maiúsculas $\{A, B, \dots, Z\}$
- Terminais: letras do início do alfabeto minúsculas $\{a, b, c, \dots\}$, dígitos $\{0..9\}$ e outros caracteres como $+, -, *, /$
- Não-Terminais: letras do fim do alfabeto maiúsculas, como X ou Y , são não-terminais ou variáveis



Gramáticas

- podem ser classificadas em gramáticas lineares (regulares) à direita (GLD) ou à esquerda (GLE), cujas regras $\alpha \rightarrow \beta$ são da forma:
 - $\alpha \in V$ - α é um não terminal (Lado esquerdo deve ter apenas não terminais - apenas 1)
 - GLD: $\beta \in (T \cup \{\epsilon\})^*$ ($V \cup \{\epsilon\}$) - $A \rightarrow wB$ ou $A \rightarrow w$
 - GLE: $\beta \in (V \cup \{\epsilon\})^*$ ($T \cup \{\epsilon\}$) - $A \rightarrow Bw$ ou $A \rightarrow w$.



Gramáticas regulares (lineares) obedecem a regra $\alpha \rightarrow \beta$

- $\alpha \in V$

- o lado esquerdo da regra é formado por um símbolo não terminal

- GLD: $\beta \in (T \cup \{\epsilon\}) (V \cup \{\epsilon\})$ - $A \rightarrow wB$ ou $A \rightarrow w$ com $|w| \geq 0$

- o lado direito da regra é formado por N símbolos terminais seguido de UM símbolo não terminal OU formado apenas por N símbolos terminais

- GLE: $\beta \in (V \cup \{\epsilon\}) (T \cup \{\epsilon\})$ - $A \rightarrow Bw$ ou $A \rightarrow w$ com $|w| \geq 0$

- o lado direito da regra é formado por UM símbolo não terminal seguido de N símbolos terminais OU formado apenas por N símbolos terminais



Gramáticas regulares (lineares)

- GLD e GLE geram exatamente a mesma classe de linguagens. Portanto, é indiferente o emprego de uma ou outra dessas duas variantes de gramática, já que ambas possuem a mesma capacidade de representação de linguagens.
- As linguagens geradas por GLD e GLE são as linguagens regulares
 - Logo, GLD e GLE são equivalentes



Gramáticas regulares

- Ex: dada a linguagem representada pela ER

$(a|b)^*(aa|bb)$

qual a gramática que a reconhece?



Gramáticas regulares

- Ex: dada a linguagem representada pela ER

$(a|b)^*(aa|bb)$

qual a gramática que a reconhece?

$$G = (\{S,A\}, \{a,b\}, P, S)$$

$$\begin{aligned} P = \{ & S \rightarrow aS \mid bS \mid A, \\ & A \rightarrow aa \mid bb \} \end{aligned}$$



Gramáticas regulares

- Ex: dada a linguagem representada pela ER
 $(a|b)^*(aa|bb)$
qual a gramática que a reconhece?

$$G = (\{S,A\}, \{a,b\}, P, S)$$

$$\begin{aligned} P = & \{ S \rightarrow aS \mid bS \mid A, \\ & A \rightarrow aa \mid bb \} \end{aligned}$$

- mostre os passos para reconhecer a palavra **babb**



Gramáticas regulares

- Ex: dada a linguagem representada pela ER
 $(a|b)^*(aa|bb)$
qual a gramática que a reconhece?

$$G = (\{S,A\}, \{a,b\}, P, S)$$

$$\begin{aligned} P = \{ & S \rightarrow aS \mid bS \mid A, \\ & A \rightarrow aa \mid bb \} \end{aligned}$$

- mostre os passos para reconhecer a palavra **babb**
 $S \rightarrow bS \rightarrow baS \rightarrow baA \rightarrow babb$



Extensões para GLD e GLE

- Gramática Linear Unitária à Direita (**GLUD**)
 - como na gramática linear à direita.
Adicionalmente $|w| \leq 1$ no máximo um terminal do lado direito da regra
- Gramática Linear Unitária à Esquerda (**GLUE**)
 - como na gramática linear à esquerda.
Adicionalmente $|w| \leq 1$ no máximo um terminal do lado direito da regra



Gramáticas regulares

- Ex: dada a linguagem representada pela ER $a(ba)^*$
dê as GLD, GLE, GLUD e GLUE que as reconhece.
- mostre os passos para reconhecer a palavra **ababa**



Gramáticas regulares

- Ex: dada a linguagem representada pela ER $a(ba)^*$
dê as **GLD**, **GLE**, **GLUD** e **GLUE** que as reconhece

$$G = (\{S,A\}, \{a,b\}, P, S)$$

$$P = \{ S \rightarrow aA$$

$$A \rightarrow baA \mid \epsilon \}$$

- mostre os passos para reconhecer a palavra **ababa**
 $S \rightarrow aA \rightarrow abaA \rightarrow ababaA \rightarrow ababa\epsilon$



Gramáticas regulares

- Ex: dada a linguagem representada pela ER $a(ba)^*$
dê as GLD, **GLE**, GLUD e GLUE que as reconhece

$$G = (\{S,A\}, \{a,b\}, P, S)$$

$$P = \{ S \rightarrow Sba \mid a \}$$

- mostre os passos para reconhecer a palavra **ababa**
S → Sba → Sbaba → ababa



Gramáticas regulares

- Ex: dada a linguagem representada pela ER $a(ba)^*$
dê as GLD, GLE, **GLUD** e GLUE que as reconhece

$$G = (\{S,A\}, \{a,b\}, P, S)$$

$$P = \{ S \rightarrow aA$$

$$A \rightarrow bB \mid \epsilon$$

$$B \rightarrow aA$$

- mostre os passos para reconhecer a palavra **ababa**
 $S \rightarrow aA \rightarrow abB \rightarrow abaA \rightarrow ababB \rightarrow ababaA \rightarrow ababa\epsilon$



Gramáticas regulares

- Ex: dada a linguagem representada pela ER $a(ba)^*$ dê as GLD, GLE, GLUD e **GLUE** que as reconhece

$$G = (\{S,A\}, \{a,b\}, P, S)$$

$$P = \{ S \rightarrow Aa \mid a$$

$$A \rightarrow Sb \}$$

- mostre os passos para reconhecer a palavra **ababa**
S → Aa → Sba → Aaba → Sbaba → ababa



Exercícios

- Para cada uma das linguagens (arquivo pdf disponível no ClassRoom), construir a Gramática (indicar se é GLE, GLD, GLUE ou GLUD) e a Expressão Regular. As gramáticas podem ser resolvidas com o JFlap.

- ❖ Comprimento:
 - Igual a ...
 - Maior (ou igual) a ...
 - Menor (ou igual) a ...
 - Par
 - Ímpar
 - Múltiplo de ...
- ❖ Símbolos e subcadeias:
 - Começa com ...
 - Termina com ...
 - Contém ...
 - Contém exatamente tantas ocorrências ...
 - Contém no mínimo tantas ocorrências ...
 - Contém no máximo tantas ocorrências ...
 - Justaposição
- ❖ Combinações:
 - Negação
 - E
 - Ou
 - Ou exclusivo

$$\Sigma = \{a, b, c\}$$

1

Cadeias de comprimento
qualquer, incluindo zero.

$\{\varepsilon, a, b, c, aa, ab, ac, ba, bb, bc,$
 $ca, cb, cc, aaa, aab, \dots\}$

$$\Sigma = \{a, b, c\}$$

2

Cadeias de comprimento
qualquer, maior que zero.

{a, b, c, aa, ab, ac, ba, bb, bc,
ca, cb, cc, aaa, aab, ... }

$$\Sigma = \{a, b, c\}$$

3

Cadeias de comprimento 3.

{bca, aab, aca, bab, cab, acc, abb,
abc, acb, aaa, cbb, baa, ... }

$$\Sigma = \{a, b, c\}$$

Cadeias de comprimento
diferente de 3.

4

{a, bc, bbcc, bcabaab, bcaa, c,
 ϵ , acababab, acaacabbab,
cabacacb, aabc, babac, ba,
abaaa, bbcb, ... }

$$\Sigma = \{a, b, c\}$$

5

Cadeias de comprimento
maior que 3.

{bbcc, bcabaab, bcaa,
cababab, acaacabbab,
cabacacb, aabc, babac, abaaa,
bbcb, ... }

6

$$\Sigma = \{a, b, c\}$$

Cadeias de comprimento
maior ou igual a 3.

{bbc, bcabaab, bcaa,
cababab, acaacabbab,
cabacacb, aabc, babac, abaaa,
bcb, aaa, ... }

$$\Sigma = \{a, b, c\}$$

Cadeias de comprimento
menor que 3.

$$\{\varepsilon, a, b, c, aa, ab, ac, ba, bb,
bc, ca, cb, cc\}$$

7

$$\Sigma = \{a, b, c\}$$

8

Cadeias de comprimento
múltiplo de 3.

{bca, acabababb, ϵ , acabab,
cabacacbb, aabcbabaccba,
baaaba, aaa, bbbbbbb,
aabaaacbabb, aac, ... }

9

$$\Sigma = \{a, b, c\}$$

Cadeias de comprimento
múltiplo de 4.

{bcaa, acababab, ϵ ,
aabcbabaccba, aaac, ... }

$$\Sigma = \{a, b, c\}$$

10

Cadeias com uma quantidade
par de símbolos.

{ ε , bb, ac, aabc, abac, abbc,
abcc, acac, acbc, aaaacb,
bababc, ... }

$$\Sigma = \{a, b, c\}$$

11

Cadeias com uma quantidade
ímpar de símbolos.

{bcb, acbbb, a, c, aabcbbb,
bbbacbbba, abc, cbabc, aaa, ... }

$$\Sigma = \{a, b, c\}$$

Cadeias iniciando com “abb”.

{abb, abba, abbab, abbabb,
abbcabbc, abbccbbb, ... }

12

$$\Sigma = \{a, b, c\}$$

Cadeias que não iniciam com
“aa”.

{abb, aba, abbb, bbabb,
bcabbc, babbccc, ... }

13

$$\Sigma = \{a, b, c\}$$

Cadeias terminando com 3
símbolos “b” consecutivos.

{bbb, acbbb, aabcbbb,
abacbbb, abbb, bbbb,
acacbbb, bbbacbbb,
abababbb, ... }

14

$$\Sigma = \{a, b, c\}$$

Cadeias terminando com 3 símbolos “b” consecutivos, e não mais que isso.

{bbb, acbbb, aabcbbb,
abacbbb, abbb, acacb,bbb,
bbbacbbb, abababbb, ... }

15

$$\Sigma = \{a, b, c\}$$

16

Cadeias que não terminam
com 2 símbolos “b”
consecutivos.

{a, b, c, acbba, aab, abacbbc,
abcc, acacbc, bbbacaa,
ababa, ccc, ... }

$$\Sigma = \{a, b, c\}$$

Cadeias iniciando com “a” e terminando com “c”.

{ac, abc, acc, aac, aabc, abac,
abbc, abcc, acac, acbc, aaaac,
aaabc, ... }

17

$$\Sigma = \{a, b, c\}$$

Cadeias que iniciam com “a” e
não terminam com “c”.

{a, ab, acb, aaca, aabcb, aba,
abb, abcca, acaca, acb, aaaa,
aaab, ... }

18

$$\Sigma = \{a, b, c\}$$

Cadeias que não iniciam com
“a” e que terminam com “c”.

{c, bc, bac, bbc, , ccc, babcb,
babac, babbc, bbcc, cacacac,
cbc, ccccc, bbbbb, ... }

19

$$\Sigma = \{a, b, c\}$$

Cadeias que não iniciam com
“a” e não terminam com “c”.

20

{baca, cb, cacb, caaa, caabcb,
babacb, babbcb, cabccb, ca,
bb, bcab, bbbcb, ... }

$$\Sigma = \{a, b, c\}$$

Cadeias com exatamente 3
símbolos “b”.

{bcbb, acbbb, bbab, cbbb,
aabccb, bacabccba, bbbc,
cbabcba, ababab, ... }

21

$$\Sigma = \{a, b, c\}$$

Cadeias com pelo menos 2 símbolos “a”.

22

{bcbaab, acbabab, aaabbab,
cababab, aabcbabaa,
bacabcaacba, aaabaaabb, bc,
cbabacaba, abbab, ... }

$$\Sigma = \{a, b, c\}$$

Cadeias com no máximo 4
símbolos “c”.

{bcbaab, acbabab, ccabbab,
cabacacb, aabcbabac,
baabaaba, aaabbc, ccabcac,
abcbab, ccc, ... }

23

$$\Sigma = \{a, b, c\}$$

Cadeias que contenham no mínimo 2 símbolos “a” ou no máximo 3 símbolos “c”, de forma não exclusiva.

{abccabc, abaccbcb,
aaabcc, acccbc, abcabcabc,
cababc, aa, ababbabca, ccc, ... }

24

$$\Sigma = \{a, b, c\}$$

Cadeias com no mínimo 3 e no máximo 5 símbolos “a”.

{bcabaab, acababab,
acaacabbab, cabacacb,
aabcbabac, baaba, aaabbc,
acacabcac, aabaacbabb, aaaa, ... }

25

$$\Sigma = \{a, b, c\}$$

Cadeias que iniciam e terminam com símbolos diferentes.

{abccabc, abaccbcb,
caabca, acccb, abc, bababc, ba,
bacacc, bcbabca, cca, ... }

26

$$\Sigma = \{a, b, c\}$$

Cadeias que não possuem símbolos “a” à direita de símbolos “b”, nem símbolos “c” à direita de símbolos “b”.

{abcc, abbbbb, cccc, aabbcc,
abc, bbbc, b, aaa, aacccc, bc,
 ϵ , abc, ... }

27

28

$$\Sigma = \{a, b, c\}$$

Cadeias que possuem uma seqüência de um ou mais símbolos “b” imediatamente à direita de cada símbolo “a”.

{abccabc, abbabbccbc, caabca, abccb, abc, bababc, b, bacacc, bcbabca, ccabb, ... }

$$\Sigma = \{a, b, c\}$$

Cadeias que não contenham símbolos “b” justapostos.

{abccabc, abaccbcb,
aaabcc, acccbc, abcabcabc,
cababc, aa, bacacc, ababcbabca,
ccc, ... }

29

$$\Sigma = \{a, b, c\}$$

Cadeias com uma quantidade
par de símbolos “b”.

{bb, bcb, bbcc, bcababab, caa, c,
 ϵ , acbababab, acaacabba,
cabacacb, ababc, bbabbac,
babbb, aaaa, cbb, ... }

30

$$\Sigma = \{a, b, c\}$$

Cadeias com uma quantidade
ímpar de símbolos “c”.

{bbc, bcb, cbbcc, bcababab, caa,
c, acbababab, acaacabcba,
cacbaccacb, ababc, bbabbac,
cbccabbb, acacaca, cbb, ... }

31

$$\Sigma = \{a, b, c\}$$

Cadeias com quantidade
par de símbolos “a” e
ímpar de símbolos “c”.

{cabccabcc, aaacacacb,
bccc, cb, aabcabacaabc,
cabccabcc, accca, bacacc,
aca, ccc, ... }

32

$$\Sigma = \{a, b, c\}$$

Cadeias que contenham a
subcadeia “abc”.

{abcb, babcb, bbabcc, abc,
abcaabcb, cbabc, ababbabca, ... }

33

$$\Sigma = \{a, b, c\}$$

Cadeias que contenham pelo menos três símbolos iguais consecutivos.

{abbb, cacccbab,
bbbbbbcccc, bbaaa, aaaaa,
cccccbabc, abaaabbabca, ... }

34

$$\Sigma = \{a, b, c\}$$

Cadeias que não contenham
dois símbolos consecutivos
iguais.

{abcb, cacbcbab,
bababababcacbcac, babababa,
acabacaca, cbabc, a, b ... }

35

$$\Sigma = \{a, b, c\}$$

Cadeias que não contêm o símbolo “a”.

{cbbbc, ccccb, bb, cb,
bbabaabb, babb, aaa, ϵ ,
aaabbb, aababa, baa, ... }

36

$$\Sigma = \{a, b, c\}$$

Cadeias que não contenham a
subcadeia “ab”.

{cbacbc, acacbc, acacbb, caaaa, aacbbbacacbbc, cbbb, aaa, ϵ , aaacbbb, bbac, cccbaa, ... }

37

$$\Sigma = \{a, b, c\}$$

Cadeias que não contêm a
subcadeia “abc”.

{cababbc, acacbc, b,
aabb, caaaba, aabbabacabbc,
cbabb, aaa, ε, aaabbb, aababac,
cccbaa, ... }

38

39

- Identificadores utilizados em linguagens de programação de alto nível qualquer
- Conjunto dos símbolos utilizados por uma linguagem de programação qualquer



LFA - Aula 04

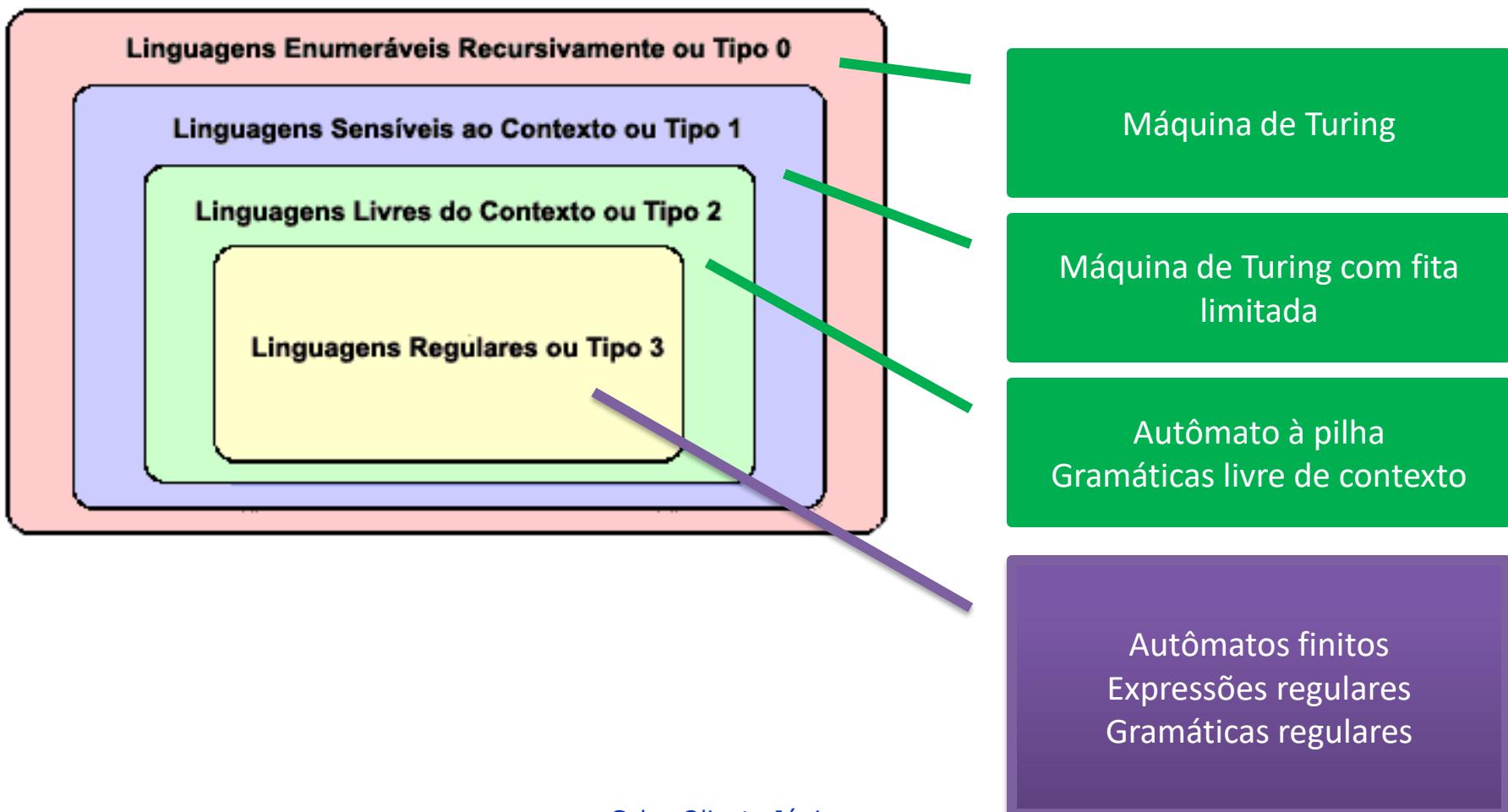
Autômatos Finitos

Celso Olivete Júnior

celso.olivete@unesp.br



Classificação das Linguagens segundo Hierarquia de Chomsky





Na aula passada...

- Gramáticas regulares
 - linear à direita
 - $A \rightarrow wB$ ou $A \rightarrow w$ com $|w| \geq 0$
 - linear unitária à direita
 - $A \rightarrow wB$ ou $A \rightarrow w$ com $|w| \leq 1$
 - linear à esquerda
 - $A \rightarrow Bw$ ou $A \rightarrow w$ com $|w| \geq 0$
 - linear unitária à esquerda
 - $A \rightarrow wB$ ou $A \rightarrow w$ com $|w| \leq 1$



Na aula de hoje:

- Linguagens regulares: Autômatos finitos (AFs)
 - Da mesma forma como ocorre com as expressões regulares (ER's) e com as gramáticas lineares (GL's), os AFs também possibilitam a formalização das linguagens regulares.
 - ER's e GL's são dispositivos de geração de sentenças
 - AF's são dispositivos de aceitação de sentenças - reconhecedores

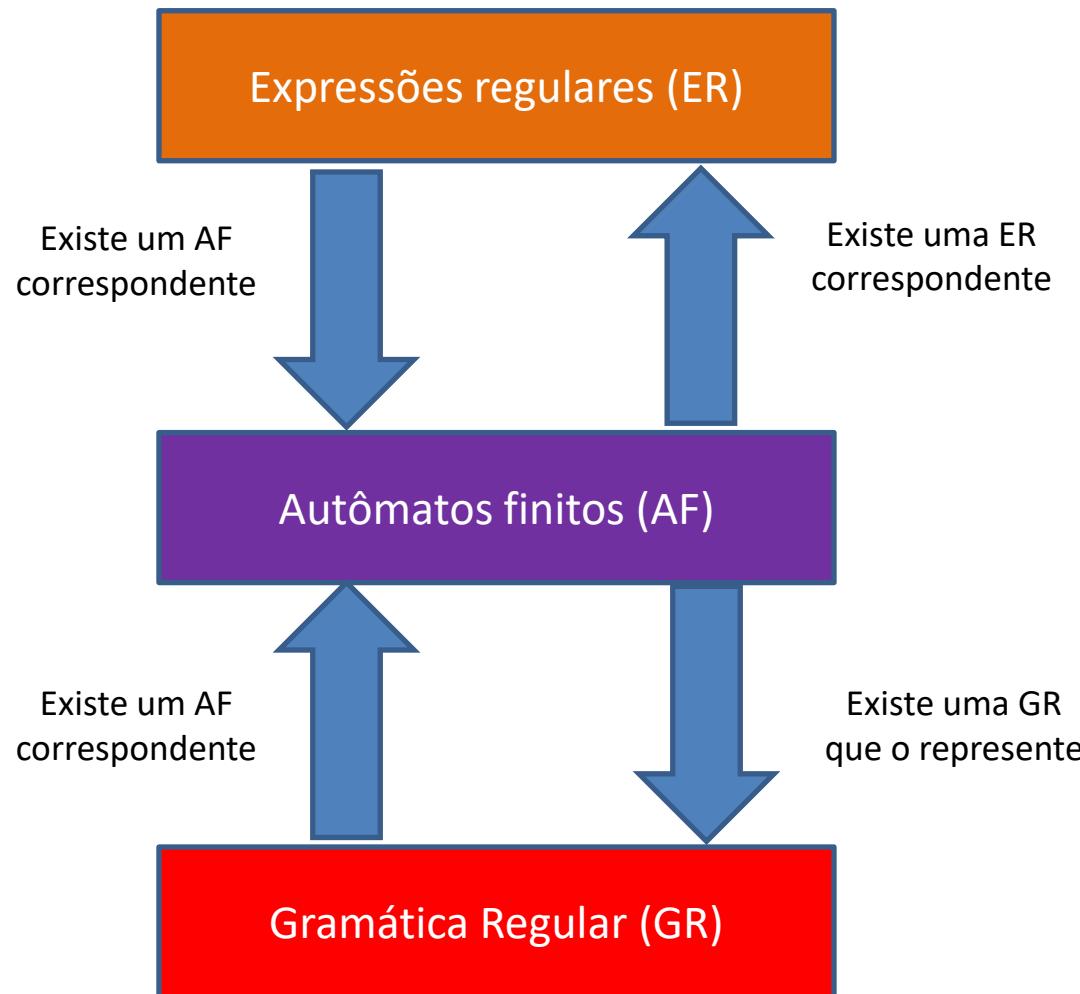


Linguagens regulares: Autômatos finitos(AF's)

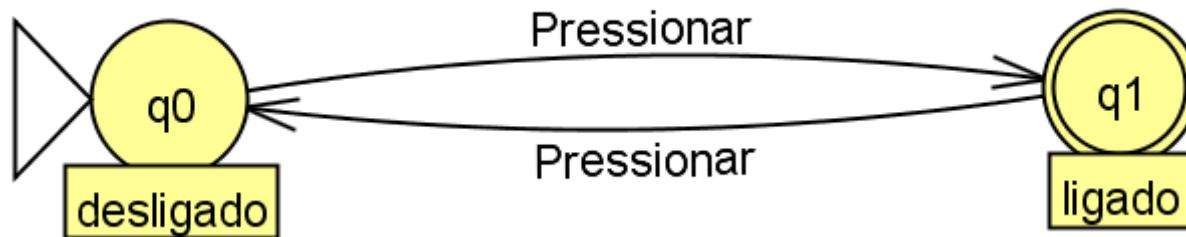
- Referências bibliográficas

- HOPCROFT, J. E.; ULLMAN, J. D.; MOTWANI, R. *Introdução à Teoria de Autômatos, Linguagens e Computação*. Editora Campus, 2002 → Capítulo 2
- RAMOS, M.V.M.; NETO, J.J.; VEGA, I.S. *Linguagens Formais: Teoria, Modelagem e Implementação*. Editora Bookman 2009. → Capítulo 3

Equivalências

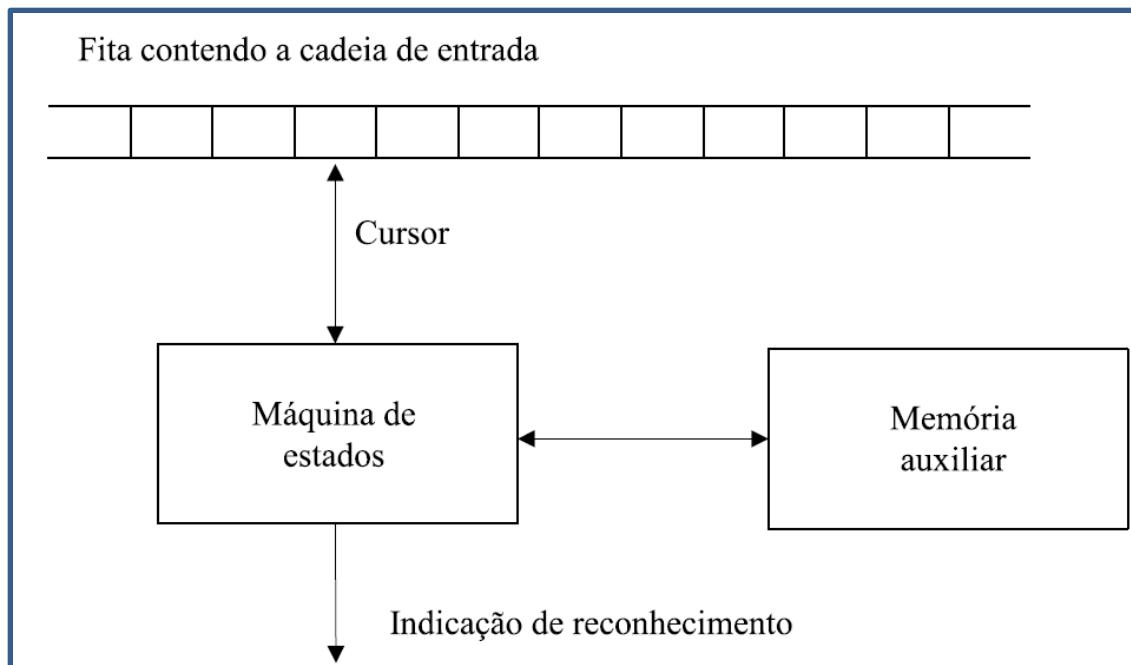


Autômatos finitos: exemplo clássico do interruptor



A estrutura de um **reconhecedor genérico**

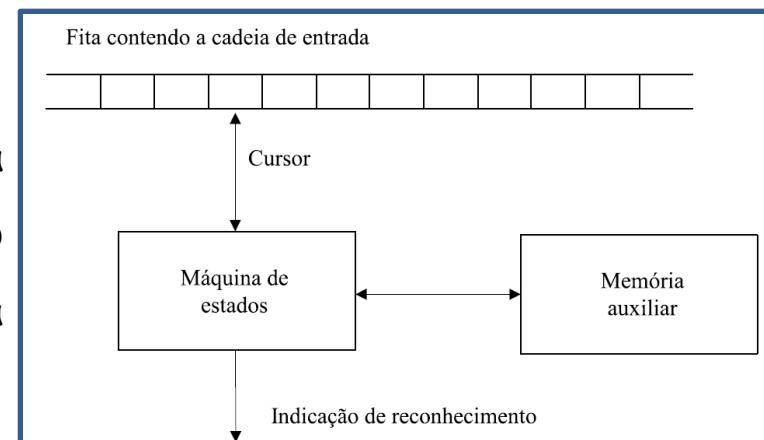
- um **reconhecedor genérico** apresenta: uma **memória (fita)** contendo o texto de entrada do reconhecedor, um **cursor**, que indica o próximo elemento da fita a ser processado, uma **máquina de estados finitos**, sem memória, e uma **memória auxiliar opcional**.





Fita de entrada

- Contém a cadeia a ser analisada pelo reconhecedor. Ela é dividida em células, e cada célula pode conter um único símbolo da cadeia de entrada, pertencente ao alfabeto Σ de entrada escolhido para o reconhecedor.
- A cadeia de entrada é **disposta da esquerda para a direita**, sendo o seu **primeiro símbolo** colocado na posição **mais à esquerda da fita**.
- Dependendo do tipo de reconhecedor considerado, a **fita** (ou o conjunto de fitas) de entrada **pode apresentar comprimento finito ou infinito**. Neste último caso, a fita pode ter ou não limitação à esquerda e/ou à direita.
- A **cadeia de entrada** registrada na fita de entrada **pode estar delimitada por símbolos especiais**, não pertencentes ao alfabeto de entrada, à sua esquerda e/ou à sua direita, porém isso não é obrigatório

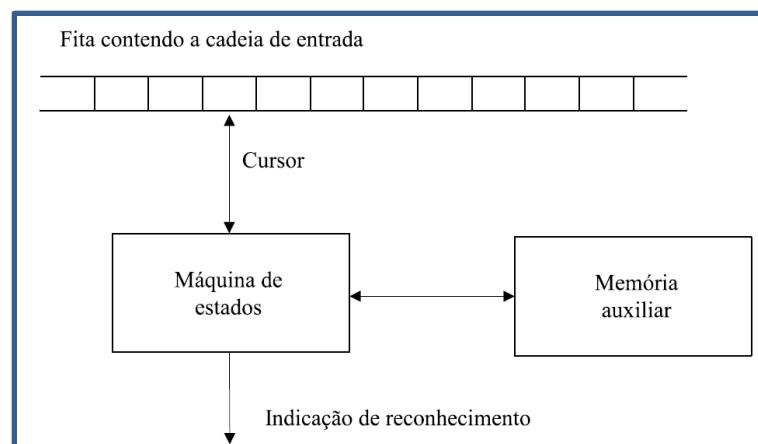




Cursor

• A leitura dos símbolos gravados na fita de entrada é feita através de um cabeçote de acesso, normalmente denominado **cursor**, o qual sempre aponta o próximo símbolo da cadeia a ser processado. Os **movimentos do cursor são controlados pela máquina de estados**, e podem, dependendo do tipo de reconhecedor, ser unidirecionais (podendo deslocar-se para um lado apenas, tipicamente para a direita) ou bidirecionais (podendo deslocar-se para a esquerda e para a direita).

• Determinados tipos de reconhecedores utilizam o cursor não apenas para lerem os símbolos da fita de entrada, mas também para escreverem sobre a fita, substituindo símbolos nela presentes por outros, de acordo com comandos determinados pela máquina de estados.

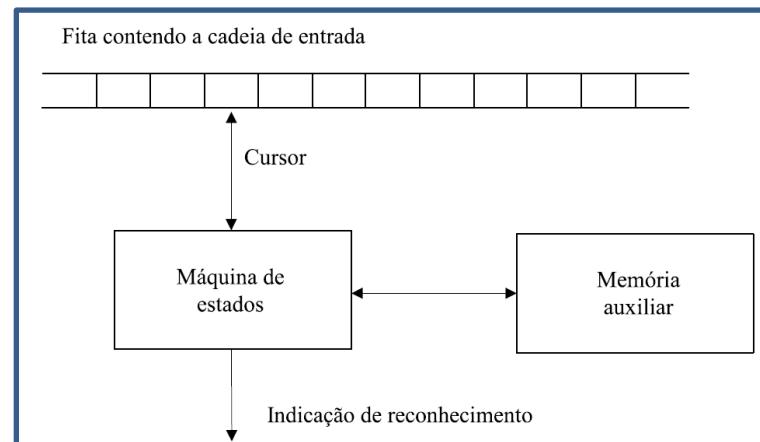




Máquina de estados

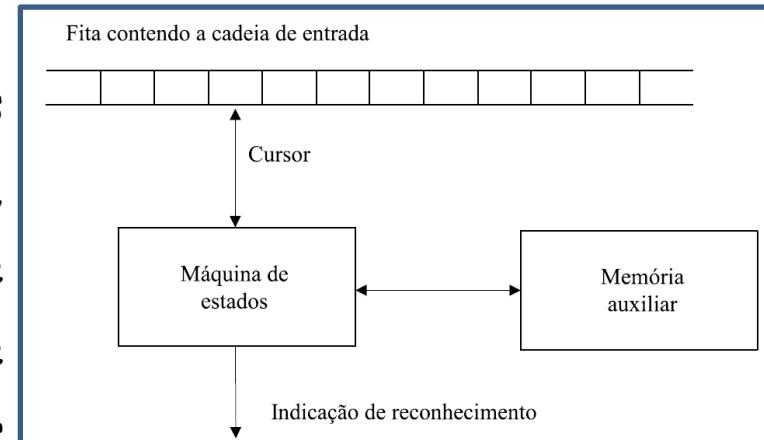
- A máquina de estados funciona como um **controlador** central do reconhecedor, e contém uma coleção finita de **estados**, responsáveis pelo registro de informações colhidas no passado, mas consideradas relevantes para decisões futuras, e transições, que **promovem as mudanças de estado** da máquina em sincronismo com as operações efetuadas **através do cursor** sobre a fita de entrada.

- Além disso, a máquina de estados finitos **pode utilizar uma memória auxiliar para armazenar e consultar outras informações**, também coletadas ao longo do processamento, que sejam eventualmente necessárias ao completo reconhecimento da cadeia de entrada.



Memória auxiliar

- A memória auxiliar é **opcional**, e torna-se necessária apenas em reconhecedores de linguagens que apresentam uma certa complexidade. Normalmente, ela **assume a forma de uma estrutura de dados** de baixa complexidade, como, por exemplo, uma **pilha** (no caso do reconhecimento de linguagens livres de contexto ou **TIPO 2**).
- As informações registradas na memória auxiliar são codificadas com base em um alfabeto de memória, e todas as operações de manipulação da memória auxiliar (leitura e escrita) fazem referência apenas aos símbolos que compõem esse alfabeto.
- Os elementos dessa memória são referenciados através de um cursor auxiliar que, eventualmente, poderá coincidir com o próprio cursor da fita de entrada. Seu tamanho não é obrigatoriamente limitado e, por definição, seu conteúdo pode ser consultado e modificado.



Funcionamento do reconhecedor

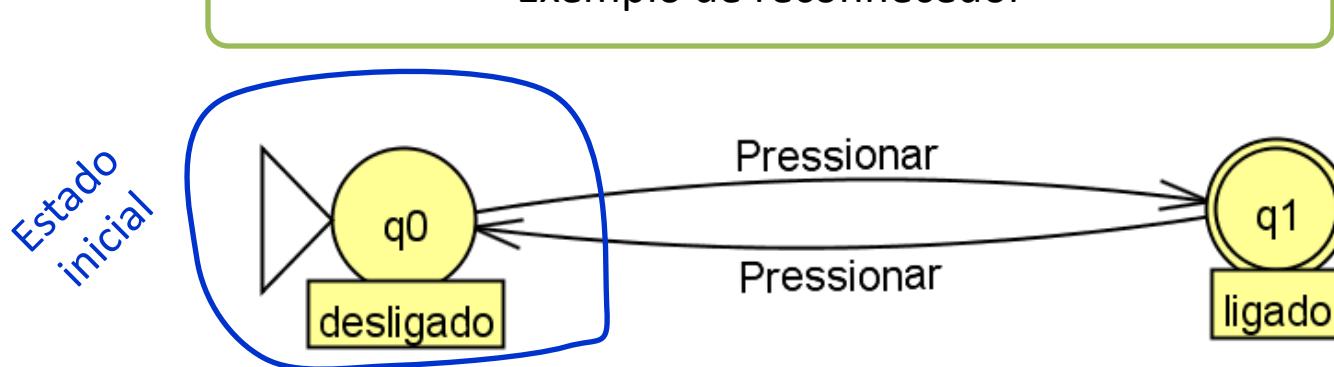
- A operação de um reconhecedor baseia-se em uma sequência de movimentos que o **conduzem**, de uma configuração inicial única, para alguma **configuração de parada**, indicativa do sucesso ou do fracasso da **tentativa de reconhecimento** da cadeia de entrada.
- A configuração de um reconhecedor genérico é caracterizada pela quádrupla:
 1. Estado;
 2. Conteúdo da fita de entrada;
 3. Posição do cursor;
 4. Conteúdo da memória auxiliar.

Configuração inicial

- A **configuração inicial** de um **reconhecedor** é definida como sendo aquela em que as seguintes condições são verificadas:

1. Estado: inicial, único para cada reconhecedor;
2. Conteúdo da fita de entrada: com a cadeia completa a ser analisada;
3. Posição do cursor: apontando para o símbolo mais à esquerda da cadeia;
4. Conteúdo da memória auxiliar: inicial, predefinido e único.

Exemplo de reconhecedor

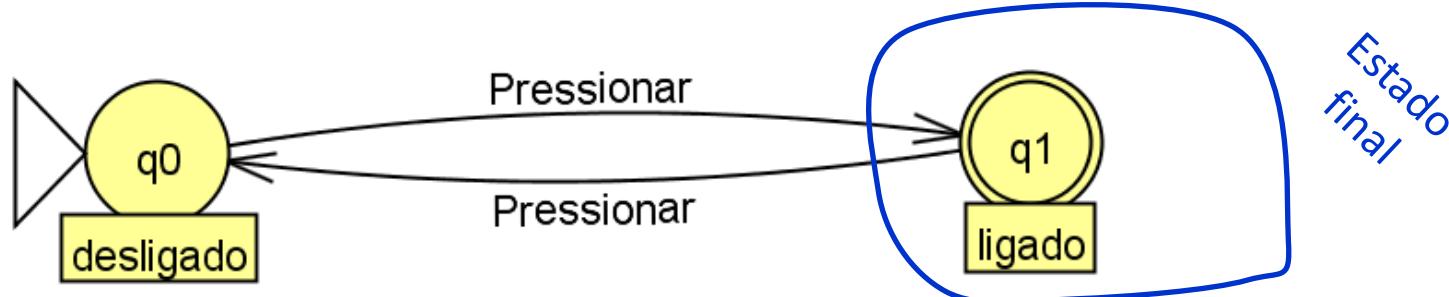




Configuração final

1. Estado: algum dos estados finais, que não são necessariamente únicos no reconhecedor;
2. Conteúdo da fita de entrada: inalterado ou alterado, em relação à configuração inicial, conforme o tipo de reconhecedor;
3. Posição do cursor: apontando para a direita do último símbolo da cadeia de entrada ou apontando para qualquer posição da fita, conforme o tipo de reconhecedor;
4. Conteúdo da memória auxiliar: final e predefinido, não necessariamente único ou idêntico ao da configuração inicial, ou apenas indefinido.

Exemplo de reconhecedor



Movimentação

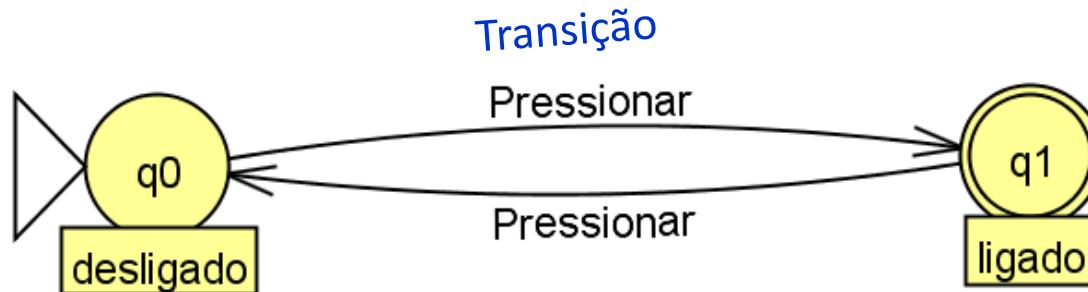
- A especificação de uma possibilidade de movimentação entre uma configuração (estado) e outra é denominada **transição**. A movimentação do reconhecedor da configuração corrente para uma configuração seguinte é feita, portanto, levando-se em conta todas as transições passíveis de serem aplicadas pelo reconhecedor à configuração corrente.

Exemplo de reconhecedor



Movimentação

- Uma transição mapeia operações formadas por:
 - Estado corrente;
 - Símbolo correntemente apontado pelo cursor da fita de entrada;
 - Símbolo correntemente apontado pelo cursor da memória auxiliar;
- em operações formadas por:
 - Próximo estado;
 - Símbolo que substituirá o símbolo correntemente apontado pelo cursor da fita de entrada e o sentido do deslocamento do cursor;
 - Símbolo que substituirá o símbolo correntemente apontado pelo cursor da memória auxiliar.



Linguagem

- Diz-se que um reconhecedor **aceita (ou reconhece) uma cadeia** se lhe **for** possível atingir alguma configuração final a partir de sua configuração inicial única, através de movimentos executados sobre tal cadeia.
- **Caso contrário**, diz-se que o reconhecedor **rejeita a cadeia**.
- A **maneira como tais configurações sucedem umas às outras** durante o reconhecimento (ou aceitação) da cadeia de entrada **define uma característica fundamental dos reconhecedores**, conforme explicado a seguir.
 - Seja o reconhecedor determinístico ou não-determinístico, a linguagem por ele aceita (ou definida) corresponde ao conjunto de todas as cadeias que ele aceita.



Autômatos finitos e linguagens regulares

- Da mesma forma como ocorre com as ER's e GR's, os **autômatos finitos** também possibilitam a formalização das linguagens regulares. No entanto, diferentemente daquelas notações, que constituem dispositivos de geração de sentenças, os **autômatos finitos são dispositivos de aceitação de sentenças** e constituem um caso particular do modelo geral de **reconhecedores**



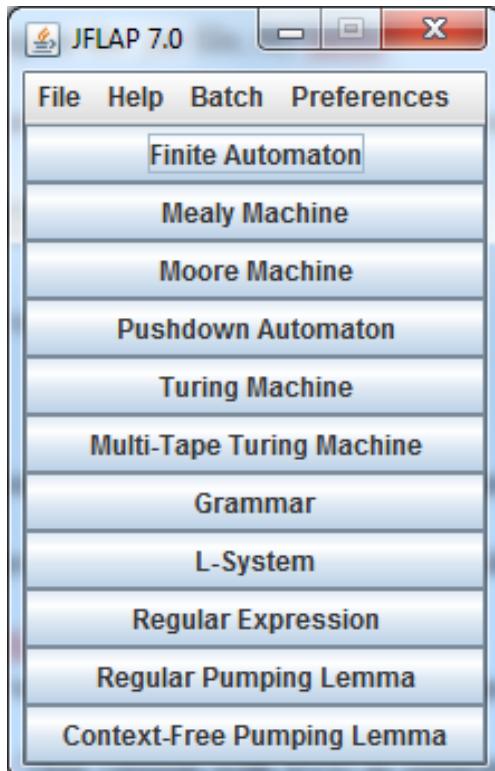
Autômatos finitos - particularidades

• Correspondem à **instância mais simples do modelo geral de reconhecedores** apresentado. As suas principais particularidades em relação ao modelo geral são:

1. **Inexistência de memória auxiliar;**
2. Utilização do **cursor** da fita de entrada **apenas para leitura de símbolos**, não havendo operações de escrita sobre a fita;
3. **Movimentação do cursor** de leitura em apenas um sentido, da esquerda para a direita;
4. A **fita** de entrada possui **comprimento limitado**, suficiente apenas para acomodar a cadeia a ser analisada.



Para construir e simular os AF's utilizaremos a ferramenta JFLAP

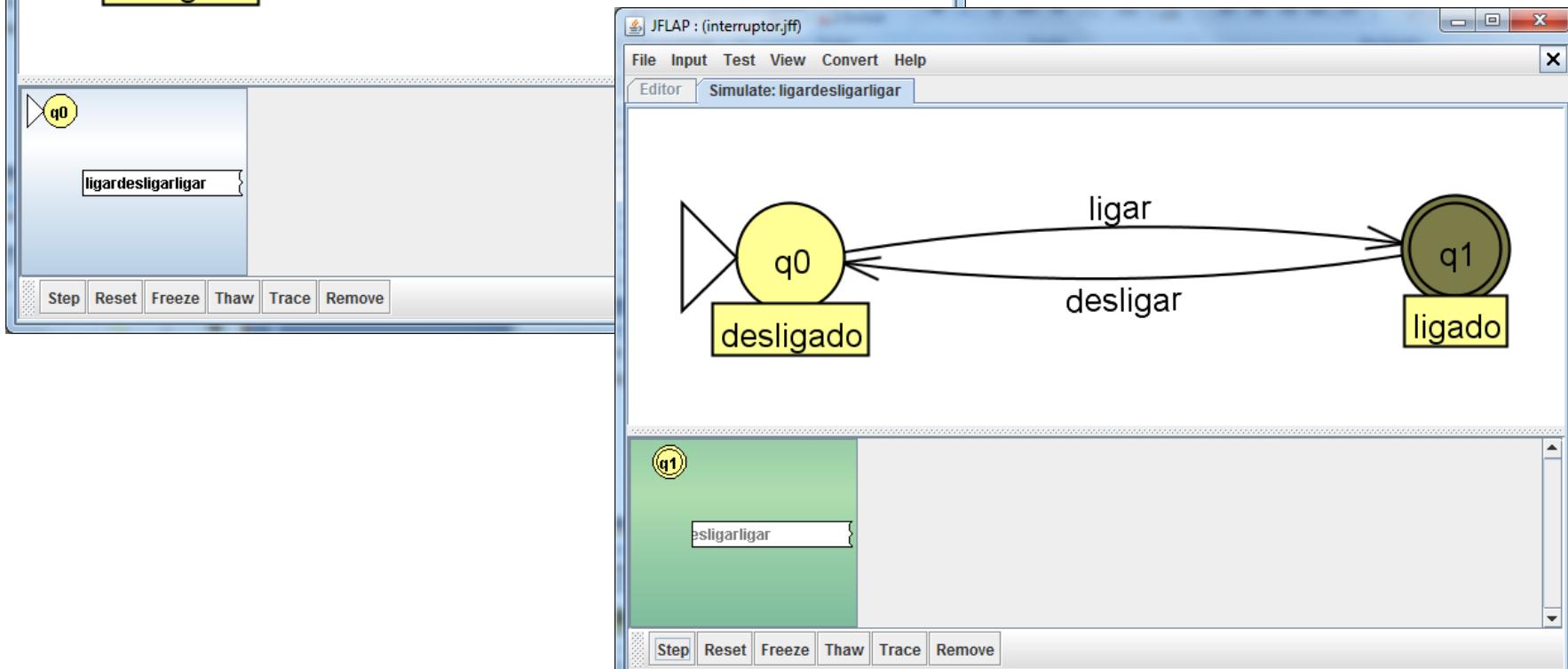
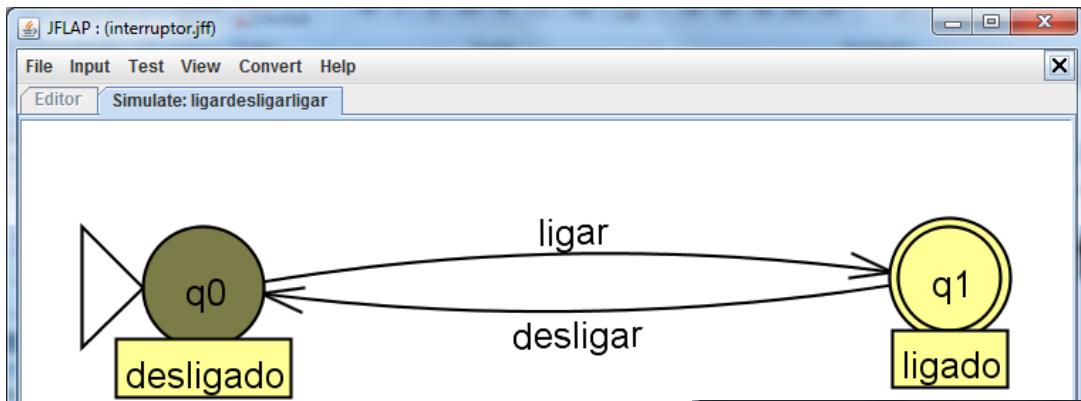


Tutorial JFLAP

<http://pt.scribd.com/doc/75454773/JFLAP>

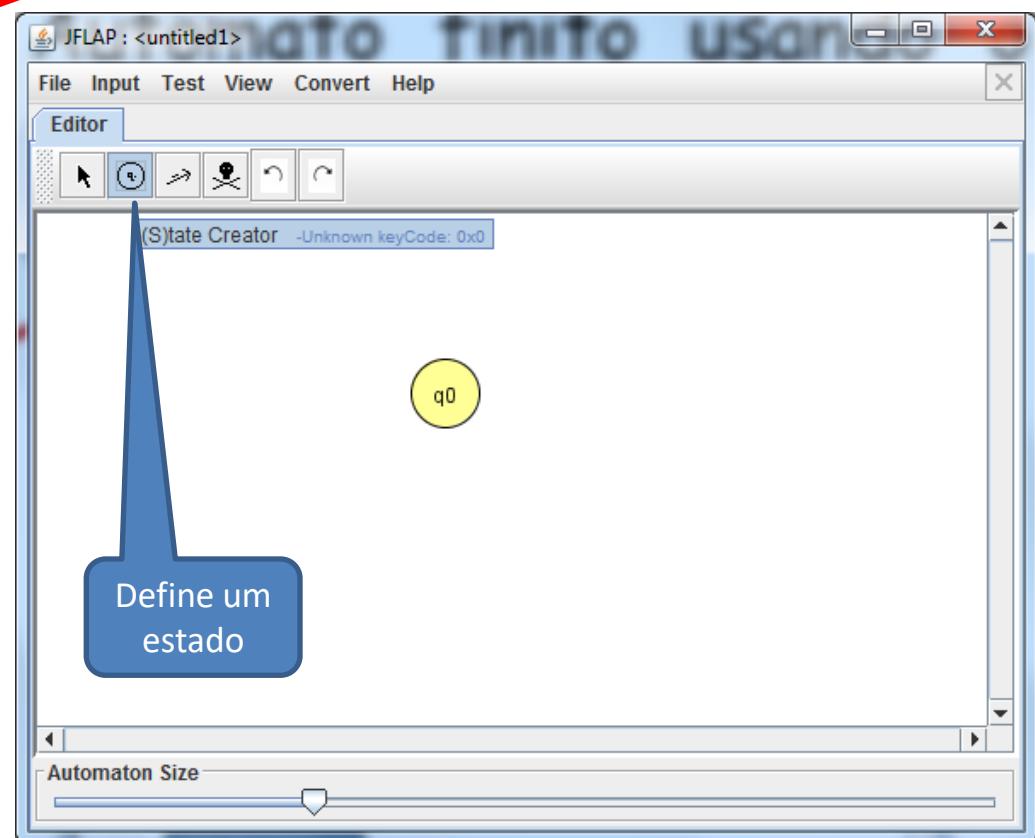
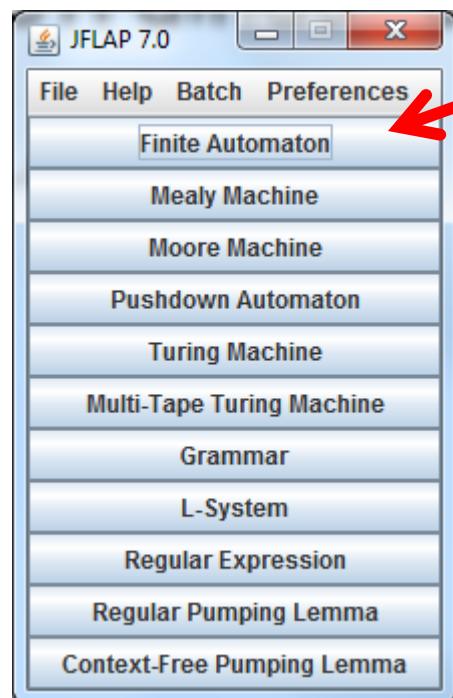
<http://www.jflap.org/>

Exemplo de Autômato finito usando o JFLAP



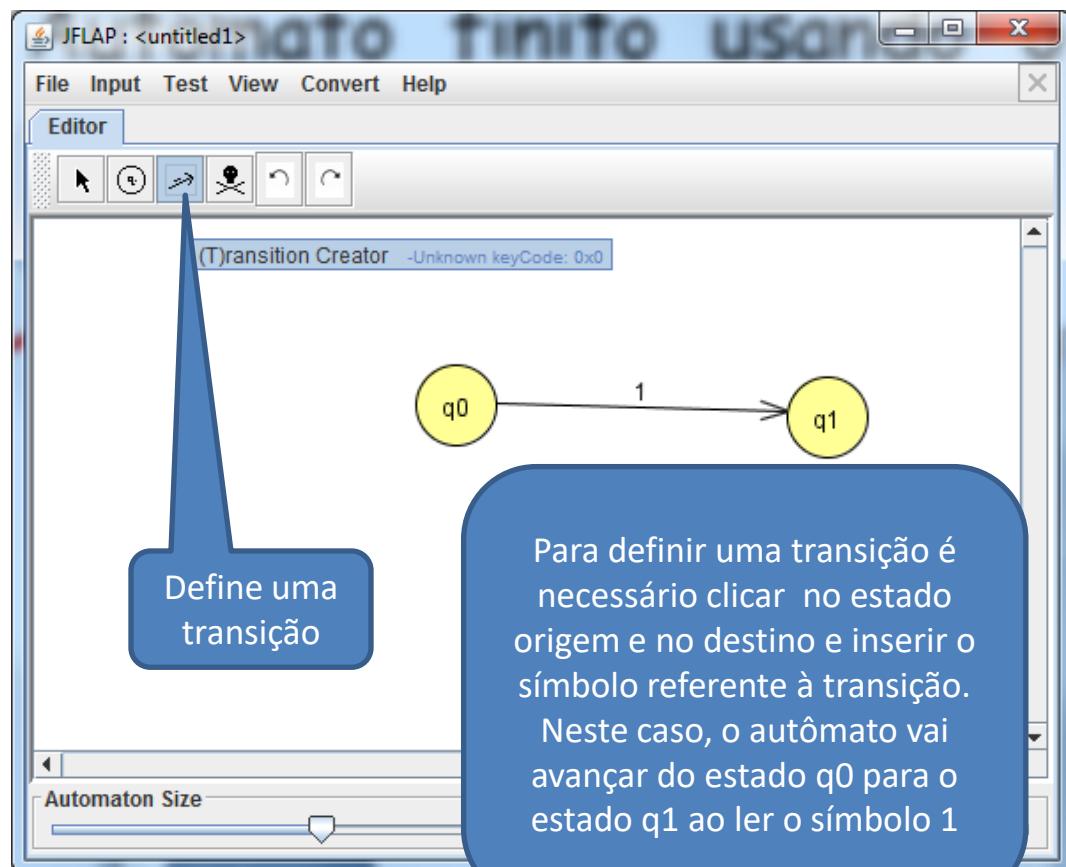
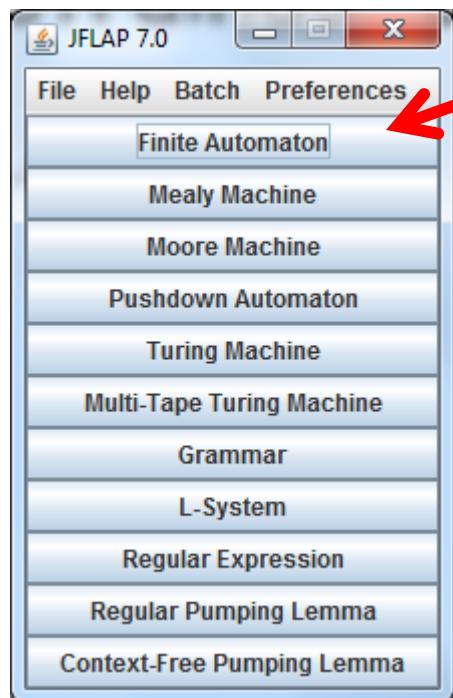
Construindo um Autômato finito usando o JFLAP

Opção para construir AF



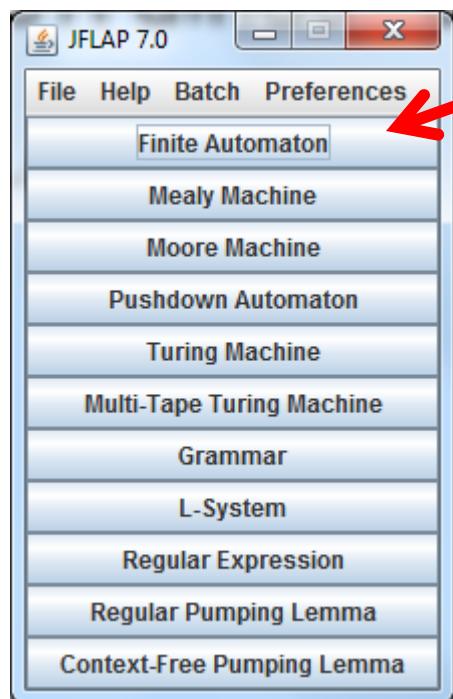
Construindo um Autômato finito usando o JFLAP

Opção para construir AF



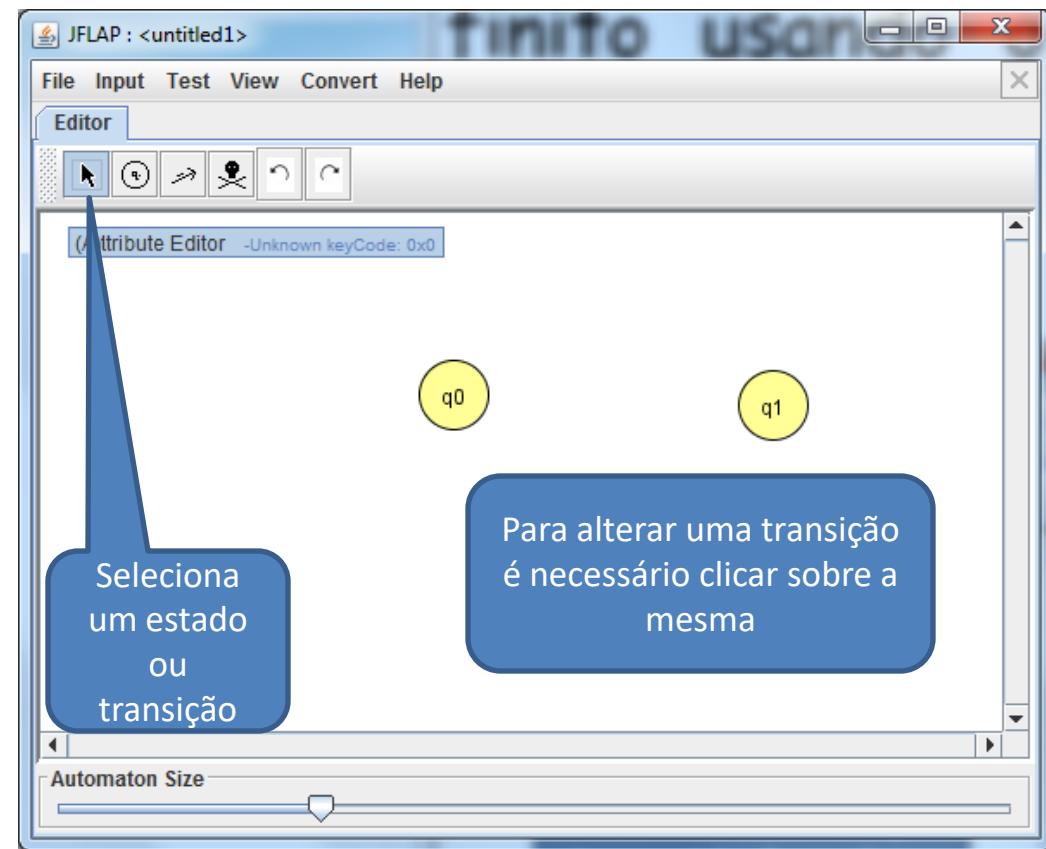
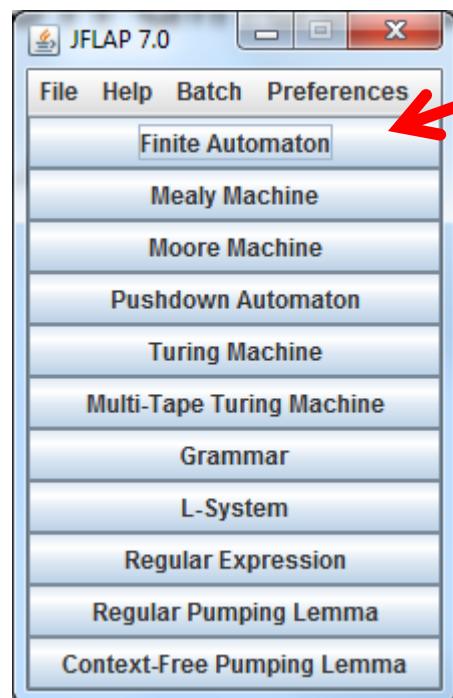
Construindo um Autômato finito usando o JFLAP

Opção para construir AF

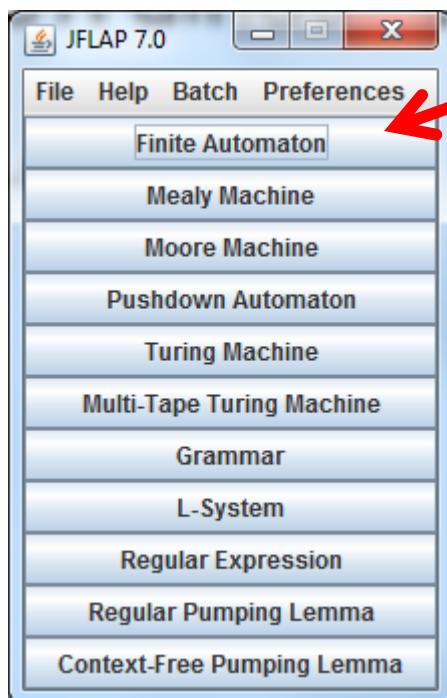


Construindo um Autômato finito usando o JFLAP

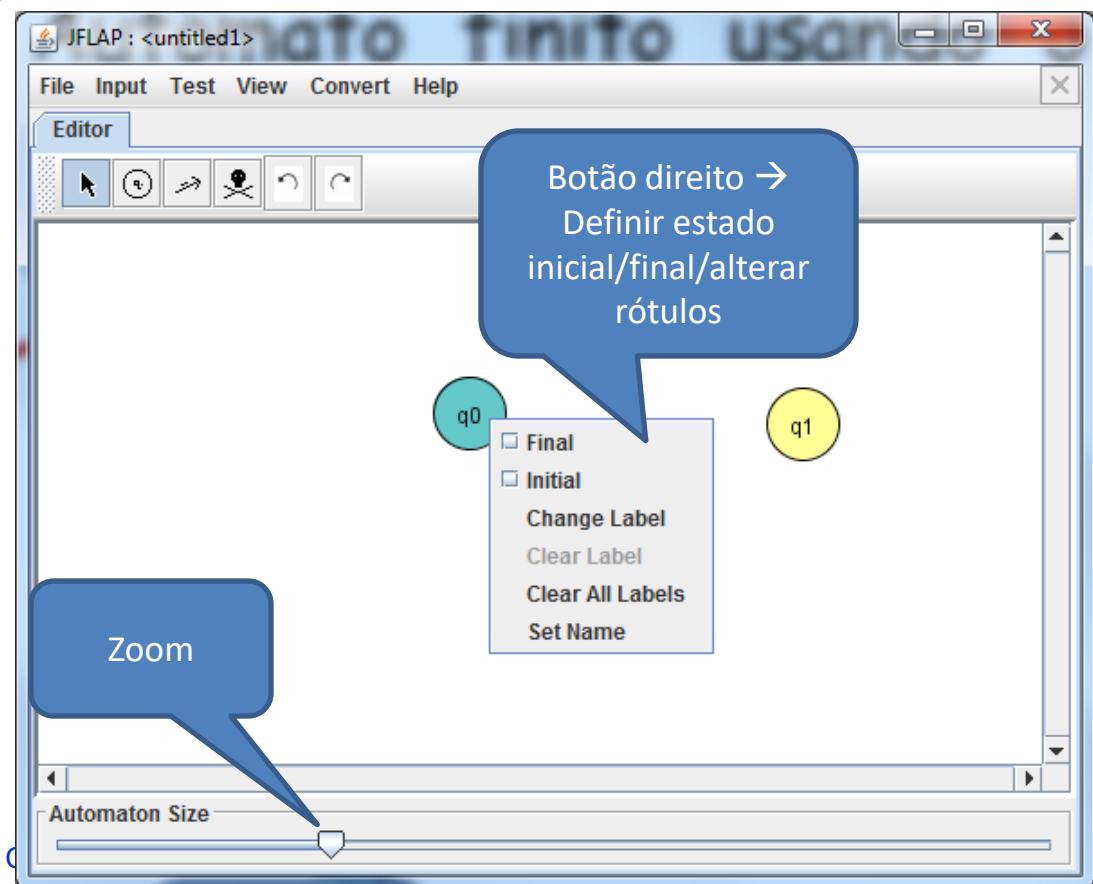
Opção para construir AF



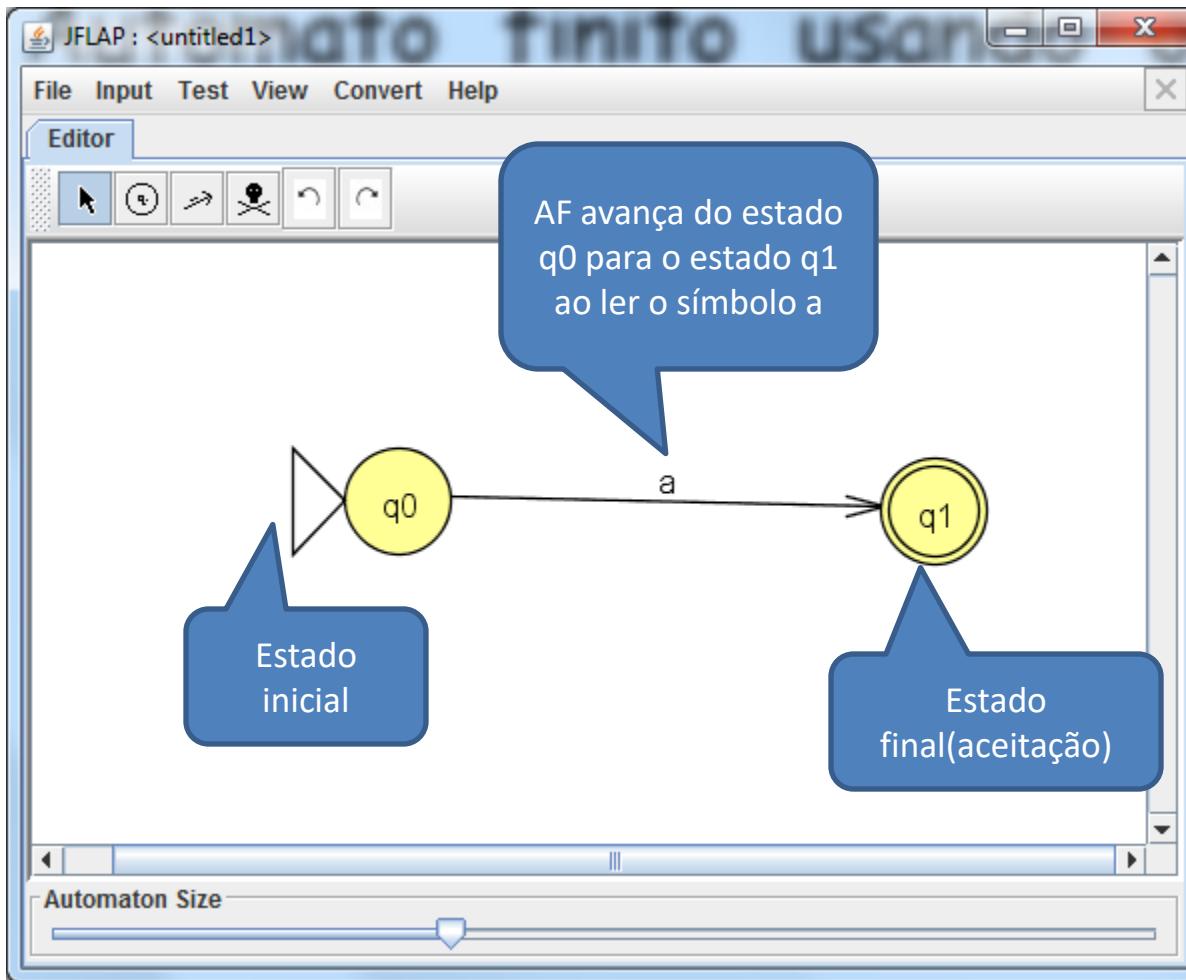
Construindo um Autômato finito usando o JFLAP



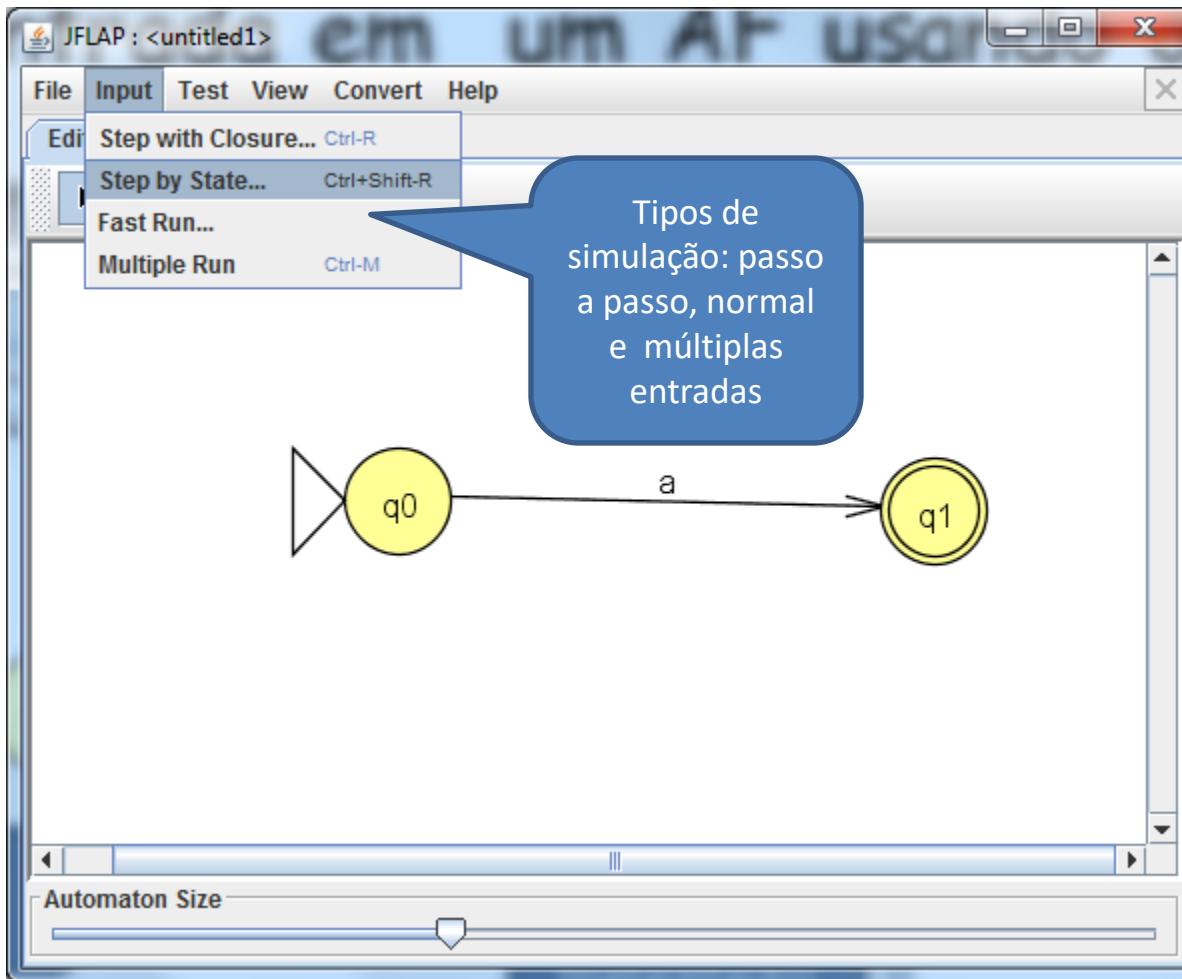
Opção
para
construir
AF



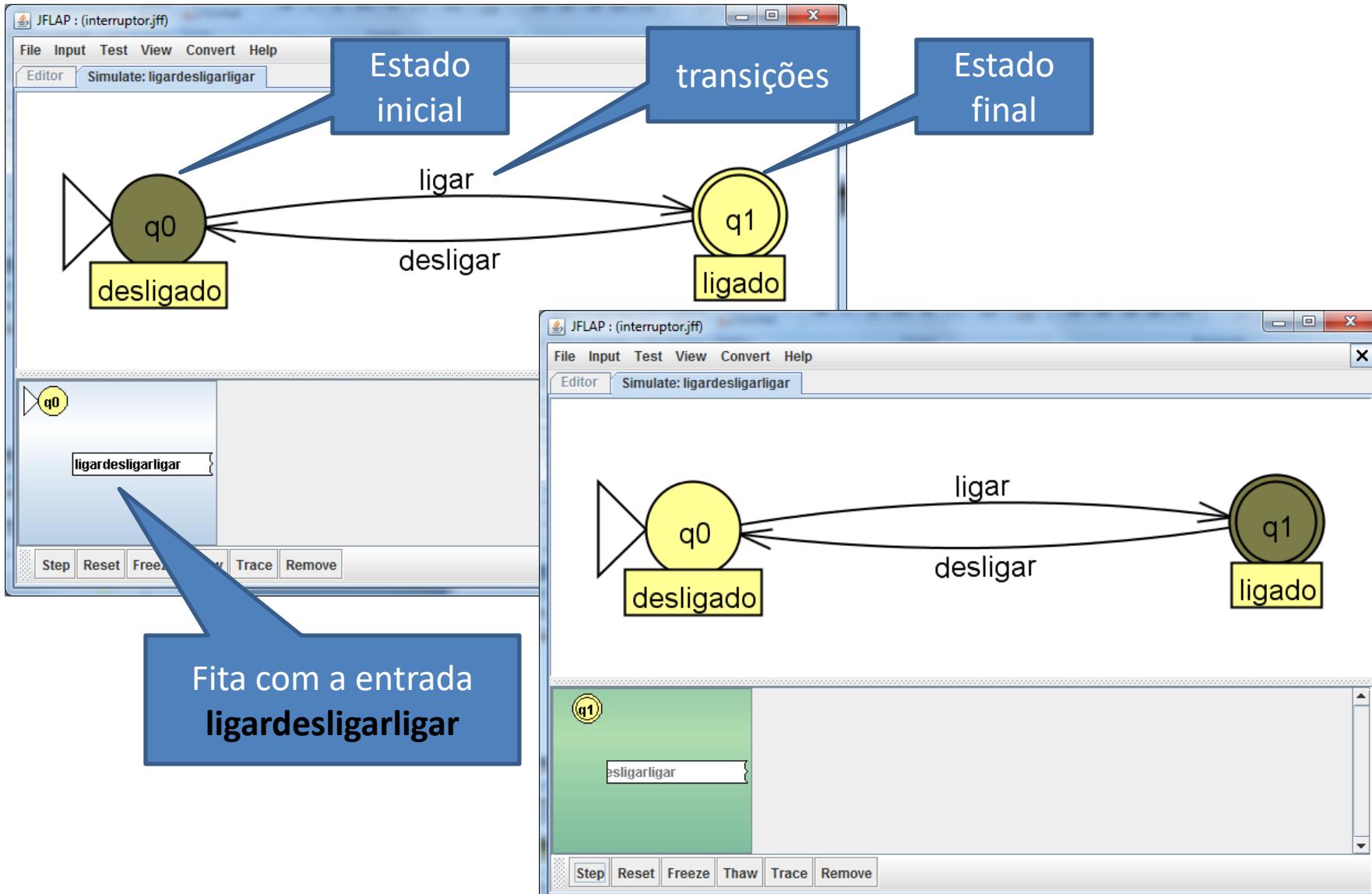
Construindo um Autômato finito usando o JFLAP



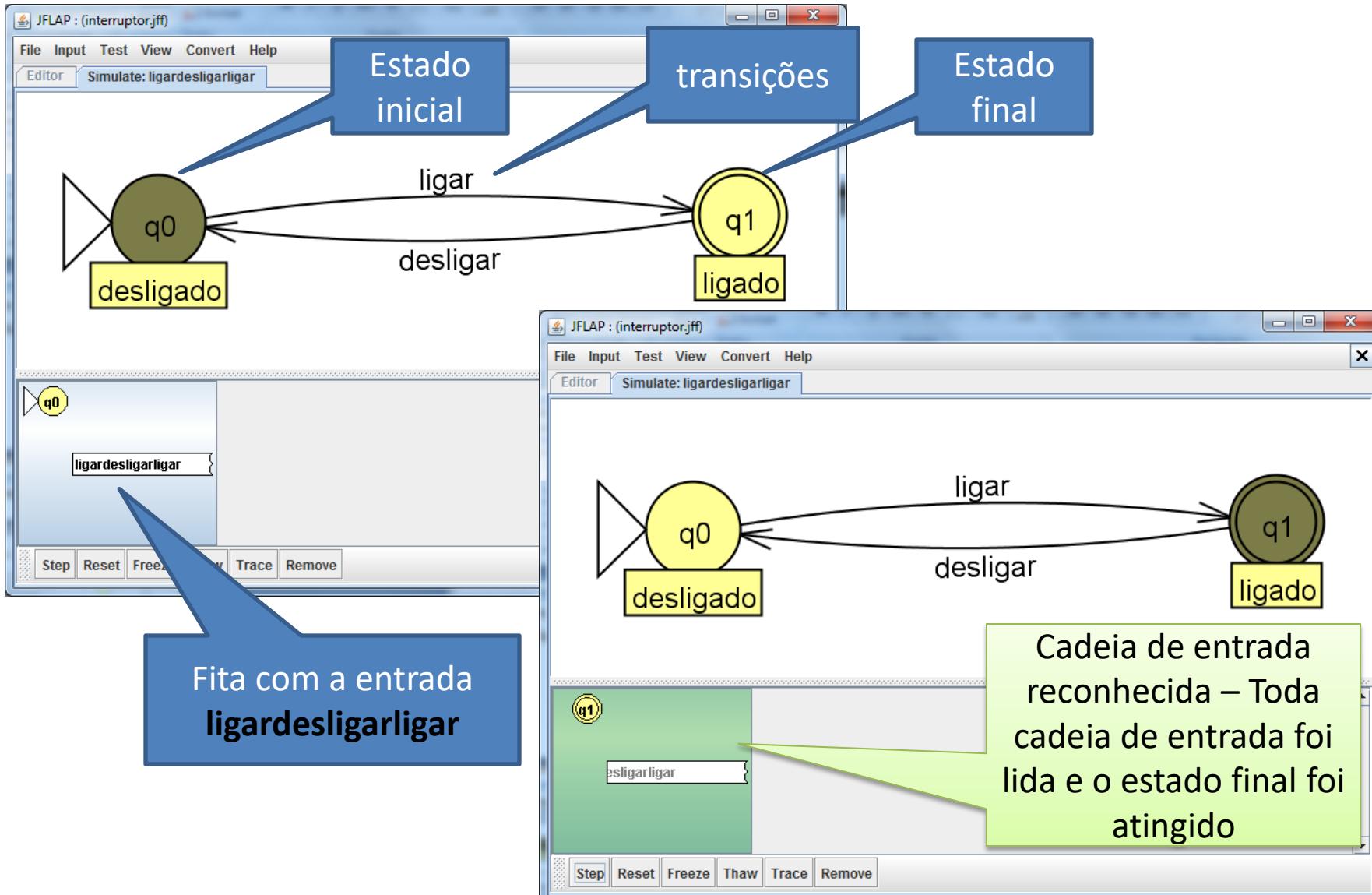
Simulando uma entrada em um AF usando o JFLAP



Exemplo de Autômato finito usando o JFLAP



Exemplo de Autômato finito usando o JFLAP



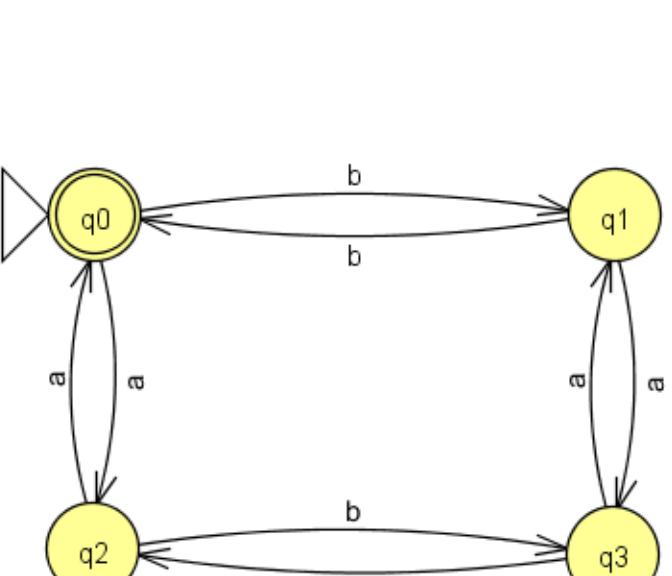
Exemplo 2 - $L = \{w | w \text{ tem o número par de } a \text{ e } b\}$ Ex: aabb aaaabbbb

JFLAP : (anbn.jff)

File Input Test Convert Help

Editor Multiple Run

Input	Result
ab	Reject
aabb	Accept
aaabbb	Reject
aaaabbbb	Accept



Run Inputs Clear Enter Lambda View Trace

Exemplo 2 - $L = \{w | w \text{ tem o número par de } a \text{ e } b\}$ Ex: aabb aaaabbbb

JFLAP : (anbn.jff)

File Input Test Convert Help

Editor Multiple Run

Input	Result
ab	Reject
aabb	Accept
aaabbb	Reject
aaaabbbb	Accept

```

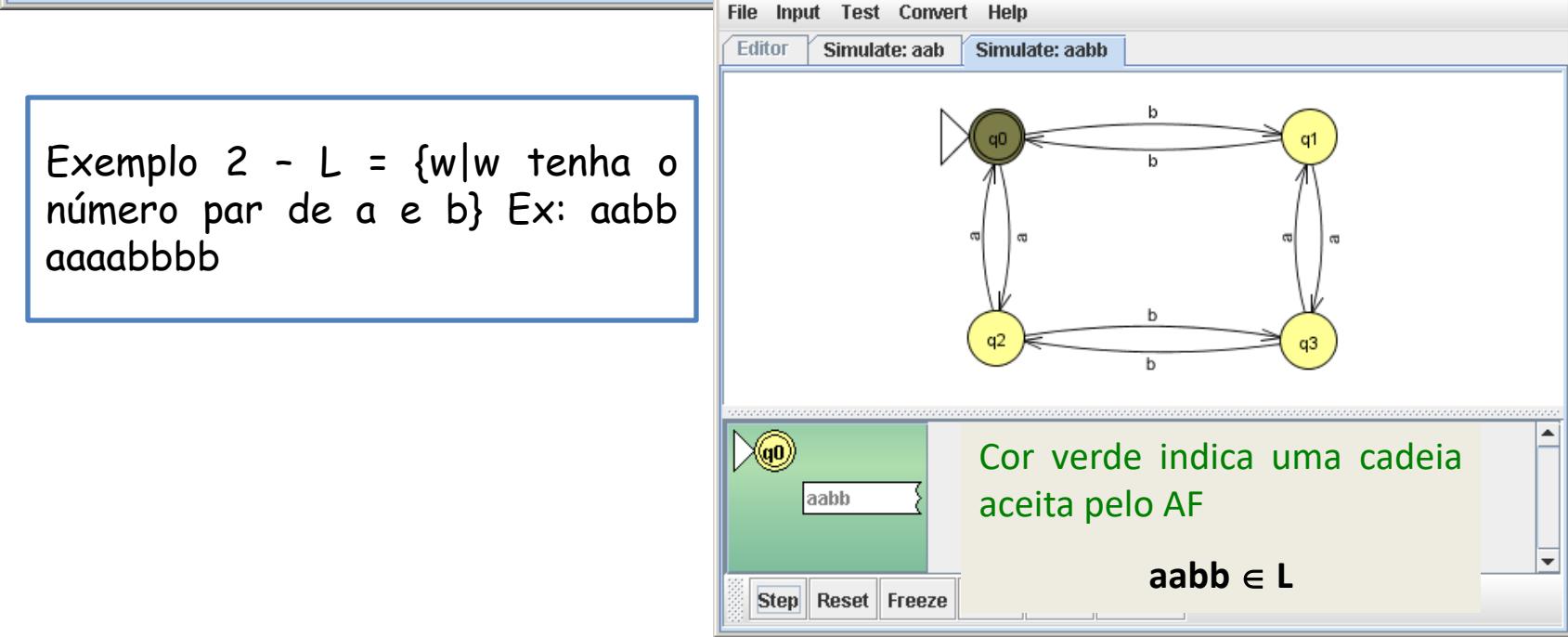
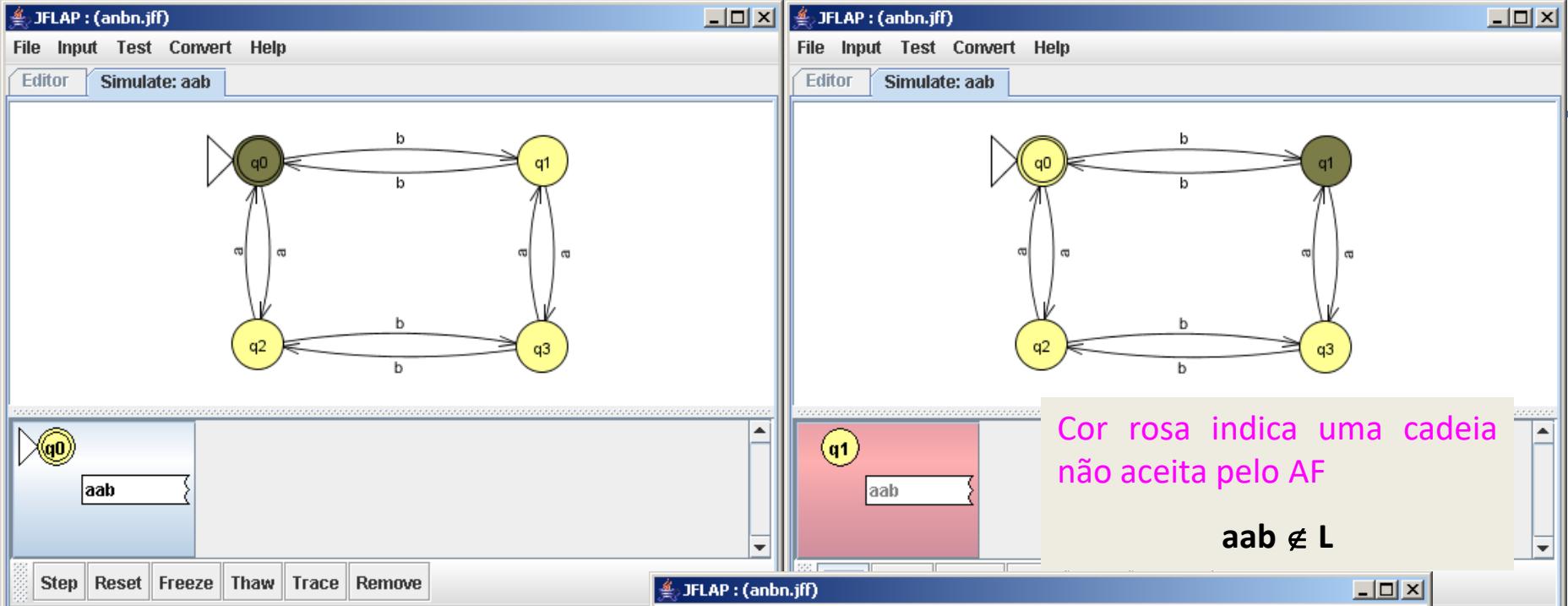
graph LR
    q0((q0)) -- a --> q2((q2))
    q0 -- b --> q1((q1))
    q1 -- a --> q3((q3))
    q1 -- b --> q2
    q2 -- a --> q0
    q2 -- b --> q3
    q3 -- a --> q1
    q3 -- b --> q0
  
```

The diagram shows a state transition graph with four states: q0, q1, q2, and q3. Transitions are labeled with 'a' or 'b'. q0 transitions to q2 on 'a' and to q1 on 'b'. q1 transitions to q3 on 'a' and to q2 on 'b'. q2 transitions to q0 on 'a' and to q3 on 'b'. q3 transitions to q1 on 'a' and to q0 on 'b'. This structure ensures that every path from start to end has an even number of both 'a' and 'b' characters.

Run Inputs Clear Enter Lambda View Trace

Simulando com
múltiplas entradas.
JFLAP: Input →
Multiple Run

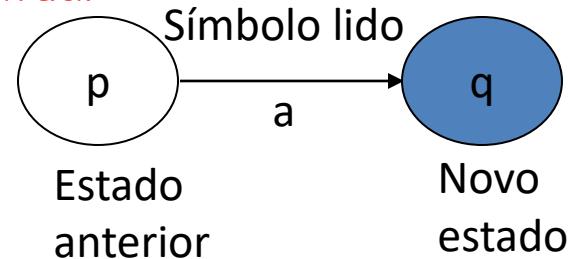
33



Classificação dos AF's

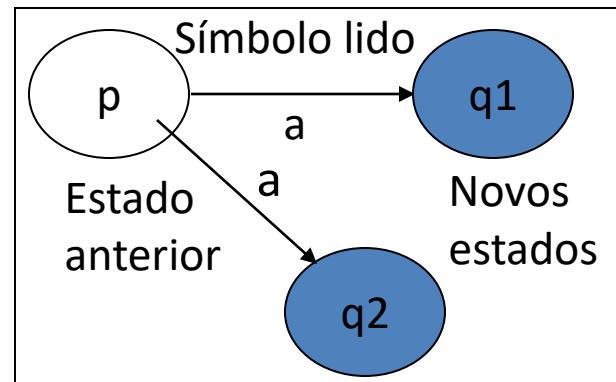
- **AF Determinístico (AFD)**

- Para cada entrada (símbolo) existe **um e somente um estado** ao qual o autômatos pode transitar **a partir de seu estado atual e do símbolo lido**



- **AF Não - Determinístico (AFND)**

- O autômatos tem o poder de estar em vários estados ao mesmo tempo





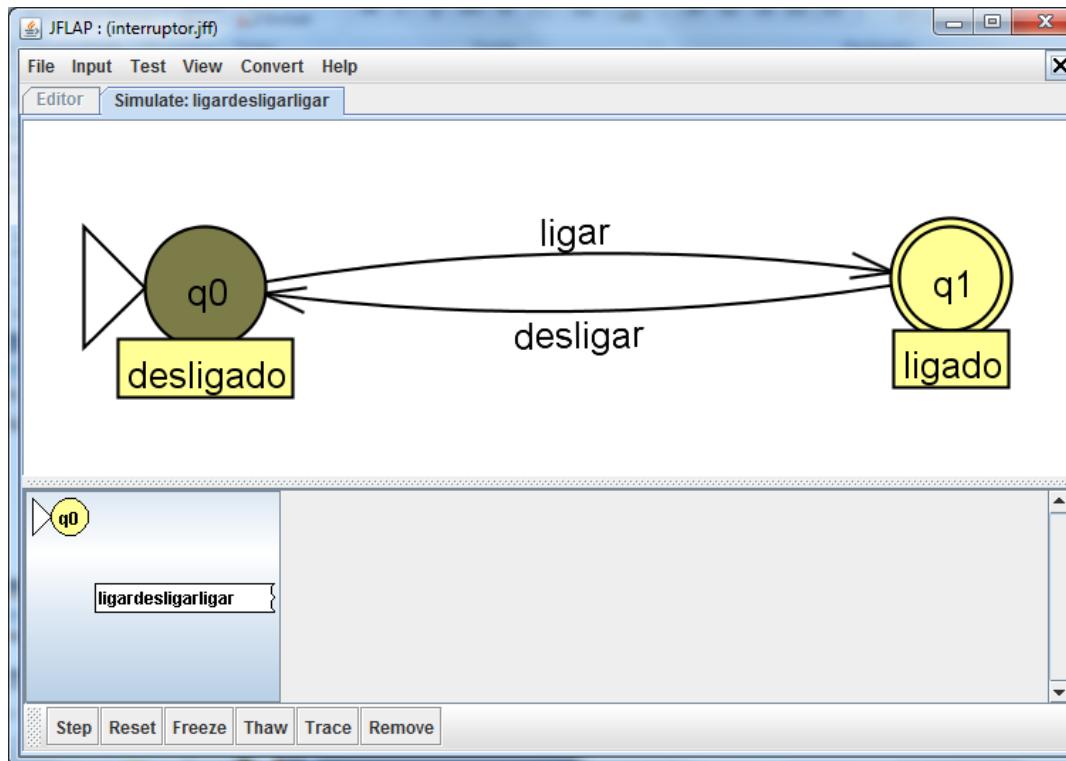
AFD

- Denotado por $A = (\{Q\}, \Sigma, \delta, q_0, \{F\})$
 - Um conjunto finito de estados, frequentemente denotado por Q
 - Um alfabeto de entrada, denotado pelo Σ
 - Uma função transição δ que toma como argumentos um estado e um símbolo de entrada e retorna um novo estado.
 - q_0 - Um estado inicial (pertencente a Q)
 - Um conjunto de estados finais ou de aceitação F (subconjunto de Q)

$$A = (Q, \Sigma, \delta, q_0, F)$$

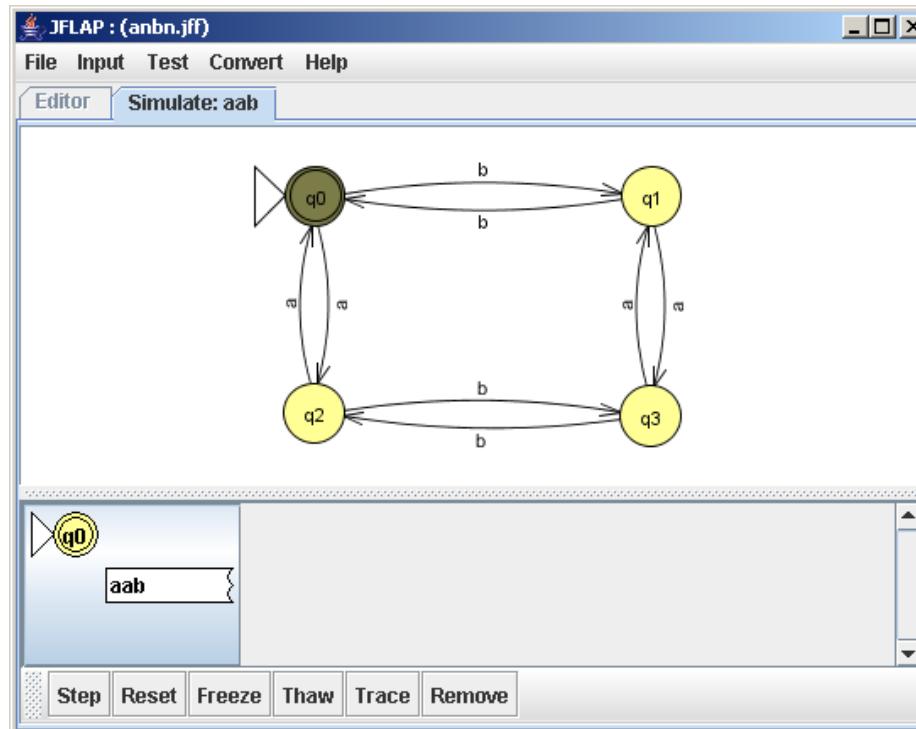
Baseado no exemplo do interruptor

$$A = (\{q_0, q_1\}, \{\text{ligar}, \text{desligar}\}, \delta, q_0, \{q_1\})$$



$$A = (Q, \Sigma, \delta, q_0, F)$$

$$A = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_0\})$$

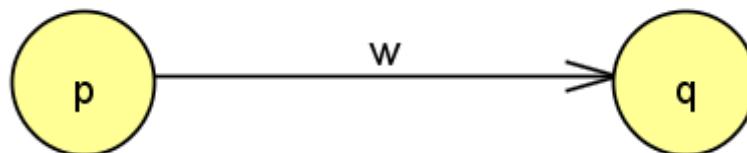


AFD - Função de Transição de Estados

$$\delta: Q \times \Sigma^* \rightarrow Q$$

$\delta(p, w) = q \rightarrow$ no estado **p** ao ler o símbolo **w**, o AFD atingirá o estado **q**.

- Isto é, existe um caminho no diagrama de transições de **p** para **q** denominado **w**



AFD - Linguagem Aceita

- Uma cadeia x é dita ser aceita pelo AFD $A = (Q, \Sigma, \delta, q_0, F)$ se $\delta(p_i, x) = q$ para algum $q \in F$.

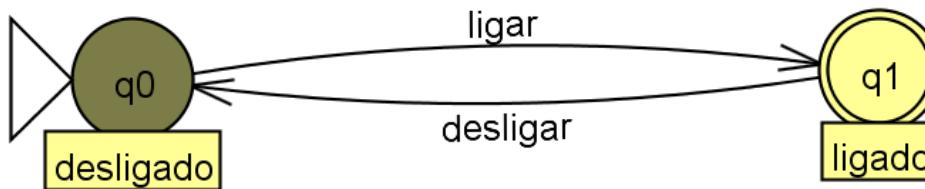
Ou $L(M) = \{x \mid \delta(p_i, x) \in F\}$

- **Definição 1:** Uma linguagem aceita por um AFD é uma linguagem regular (ou do tipo 3)
- **Definição 2:** Dois AFD A_1 e A_2 são equivalentes se $L(AFD1) = L(AFD2)$

AFD - Formas de representação

1. Diagramas de Transições

Grafo



AFD - Formas de representação

2. Tabelas de Transições

Estado Inicial,
representado por
→

Estado de
aceitação,
representado por
*

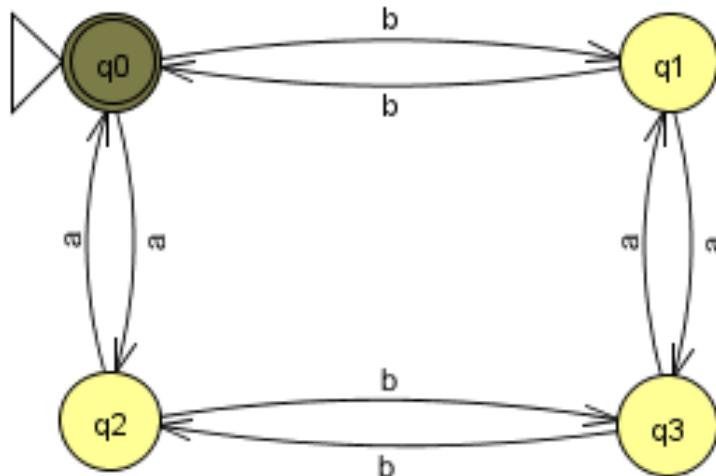
Símbolos de entrada

	0	1
→q0	q2	q1
*q1	q1	q1
q2	q2	q1

Conjunto de Estados

Qual o diagrama
de transições
correspondente?

Reconhecimento de bbaa



$$A = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_0\})$$

Tabela de Transição de Estados

	a	b
$\rightarrow^* q_0$	q2	q1
q1	q3	q0
q2	q0	q3
q3	q1	q2

Função de Transição de Estados

$$\delta(q_0, a) = q_2 \quad \delta(q_0, b) = q_1$$

$$\delta(q_1, a) = q_3 \quad \delta(q_1, b) = q_0$$

$$\delta(q_2, a) = q_0 \quad \delta(q_2, b) = q_3$$

$$\delta(q_3, a) = q_1 \quad \delta(q_3, b) = q_2$$



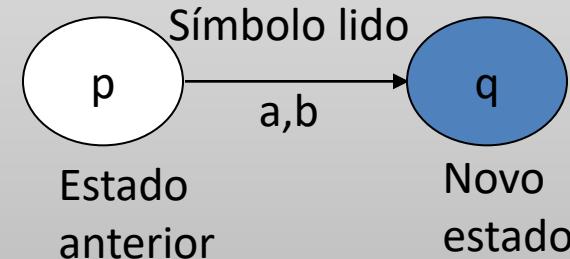
Exercícios

1. Construir AFD's (**determinísticos**) que reconhecem as linguagens sobre o alfabeto $\{a,b\}$ e cujas sentenças estão descritas a seguir:

- Começam com aa;
- Não começam com aa;
- Terminam com bbb;
- Não terminam com bbb;
- Contém a subcadeia aabbb;
- Possuem comprimento maior ou igual a 3;
- Possuem comprimento menor ou igual a 3;
- Possuem comprimento diferente de 3;
- Possuem comprimento par;
- Possuem comprimento ímpar;
- Possuem comprimento múltiplo de 4;
- Possuem quantidade par de símbolos a;
- Possuem quantidade ímpar de símbolos b.

Relembrando AFD:

UM E SOMENTE um estado a seguir a partir de seu estado atual e do símbolo lido





Exercícios

2. Construa ER's, GR's e AFD's, na forma de diagrama de estados e tabela de transições, que reconhecem as seguintes linguagens

$L_1 = \{x \in \{0,1\}^* \mid \text{número de } 1's \text{ em } x \text{ é múltiplo de } 3\}$

$L_2 = \{x \in \{0,1\}^* \mid x \text{ contém a subcadeia } 001\}$

L_3 = um AFD que reconhece a formação de identificadores em Pascal (LD), onde: _ representa underline; L [a-z] e D = [0-9]

L_4 = um AFD que reconhece a formação de números inteiros em Pascal (0...9)

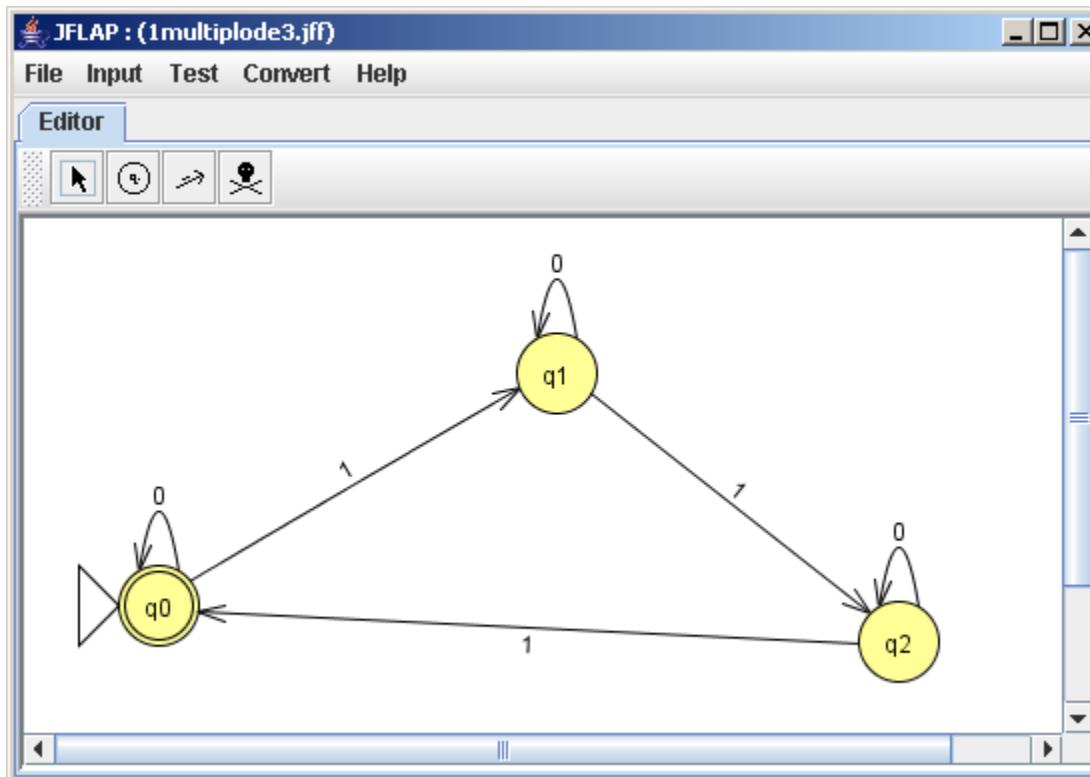
L_5 = um AFD que reconhece a formação de Operadores relacionais em Pascal (<, >, <=, =, >, >=)



Resolução dos AFD's referentes ao exercício 2

$L1 = \{x \in \{0,1\}^* \mid \text{número de } 1's \text{ em } x \text{ é múltiplo de } 3\}$

$$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_0\})$$

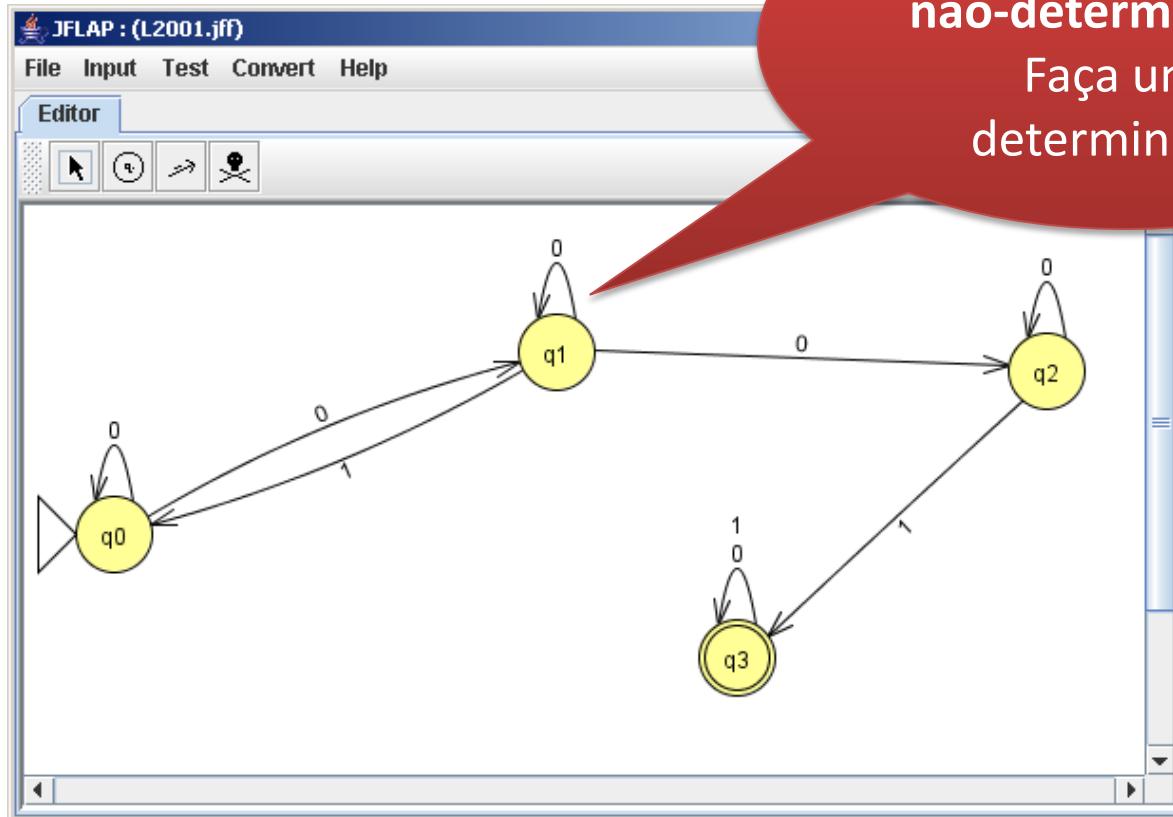


Aceita: 000111 0111111 1111111111

Rejeita: 011 00001111 00001

$$L2 = \{x \in \{0,1\}^* \mid x \text{ contém a subcadeia } 001\}$$

$$A = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0, \{q_3\})$$



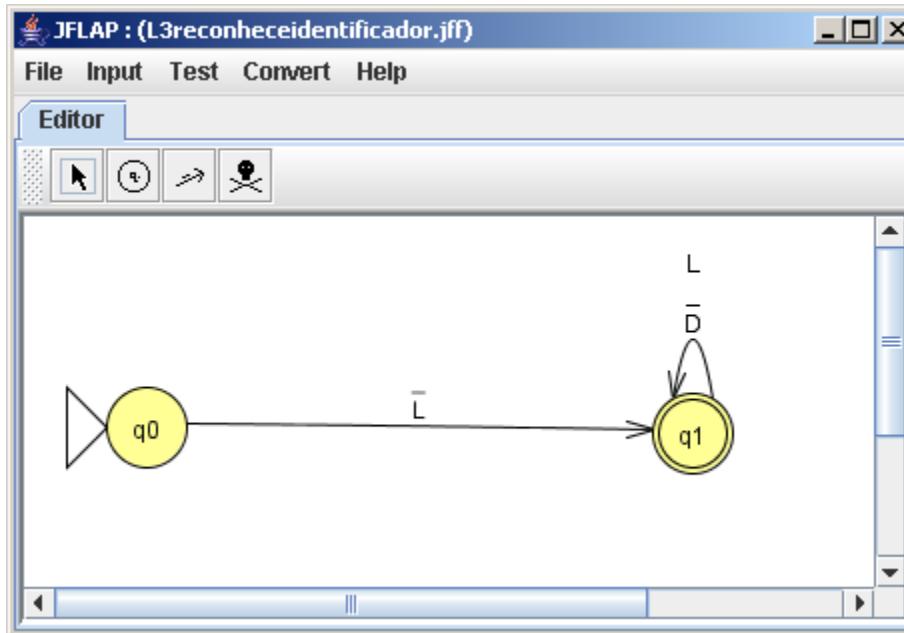
Essa resolução está
não-determinística.
Faça uma
determinística

Aceita: 00111 1001111 11100111111

Rejeita: 011 01111 000

L3 = um AFD que reconhece a formação de identificadores em Pascal

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



L é o conjunto das letras

D é o conjunto dos dígitos

_ é o sublinhado

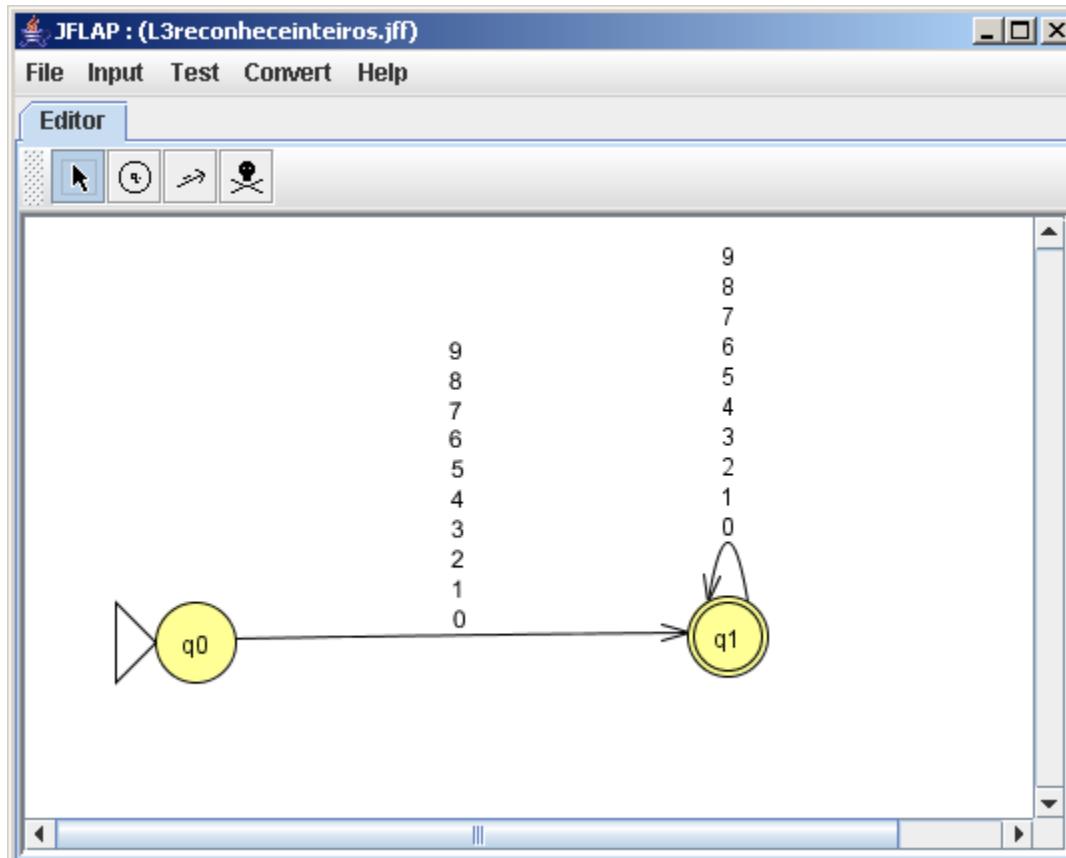
$_LLL \rightarrow _xyz$

$LD \rightarrow A123$

$L_D \rightarrow x_2$

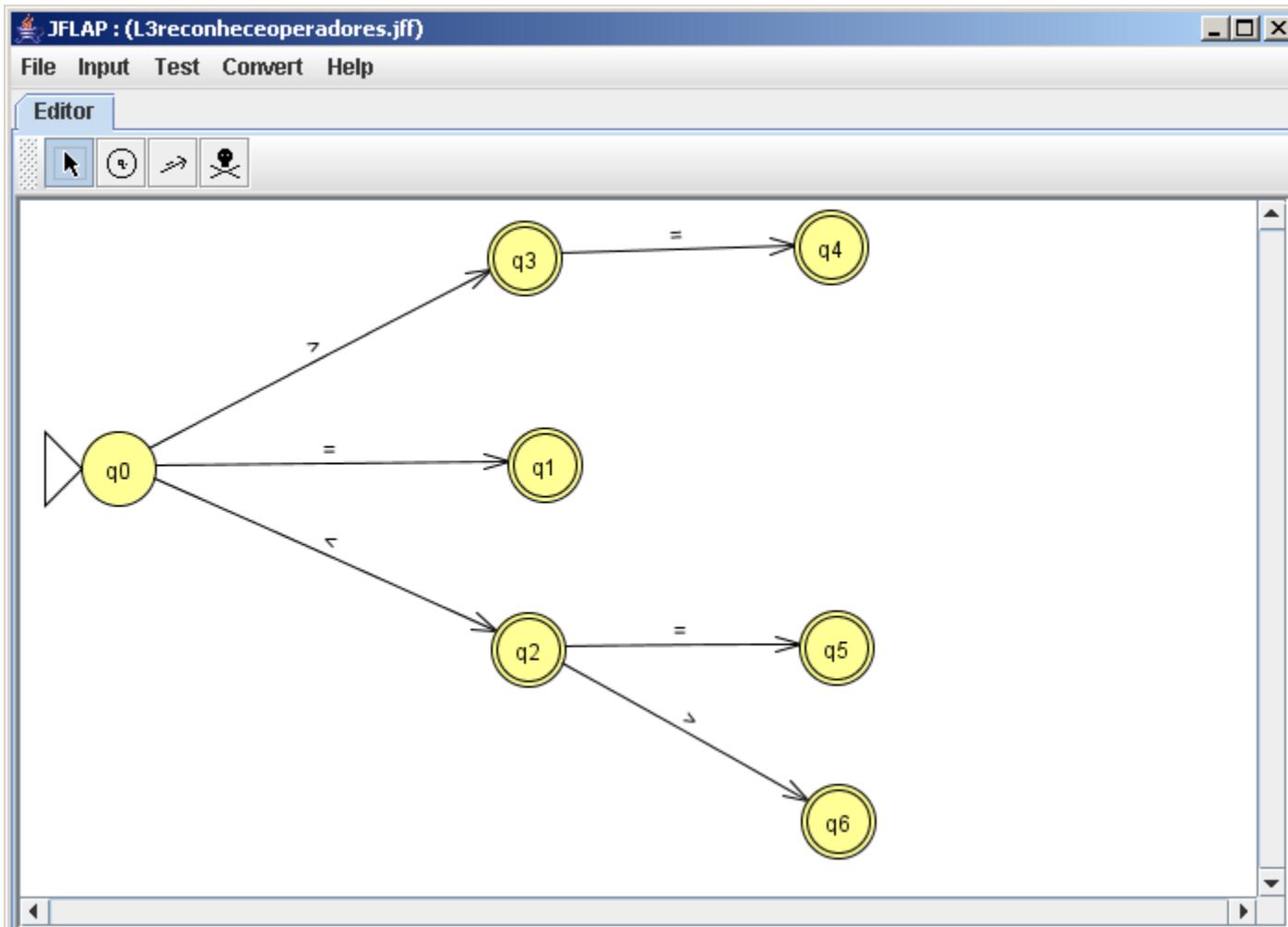
L4 = um AFD que reconhece a formação de números inteiros em Pascal

$$A = (\{q_0, q_1\}, \{0, \dots, 9\}, \delta, q_0, \{q_1\})$$



L5 = um AFD que reconhece a formação de operadores relacionais em Pascal

$$A = (\{q_0, q_1\}, \{\langle, \rangle, \leq, =, \geq\}, \delta, q_0, \{q_1, q_2, q_3, q_4, q_5, q_6\})$$





Na próxima aula...

- AFND

LFTC - Aula 05

AFND: com e sem movimentos vazios

Celso Olivete Júnior

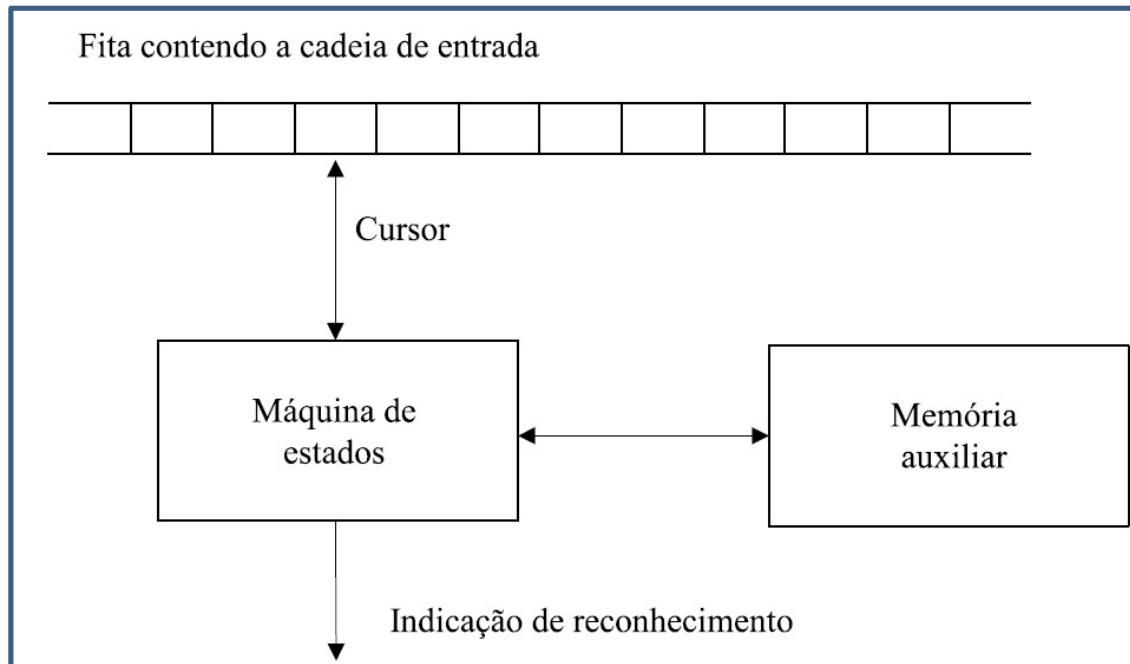
celso.olivete@unesp.br

Na aula passada...

- Reconhecedores genéricos
- Autômatos finitos determinísticos

Na aula passada...

- Estrutura de um reconhecedor genérico



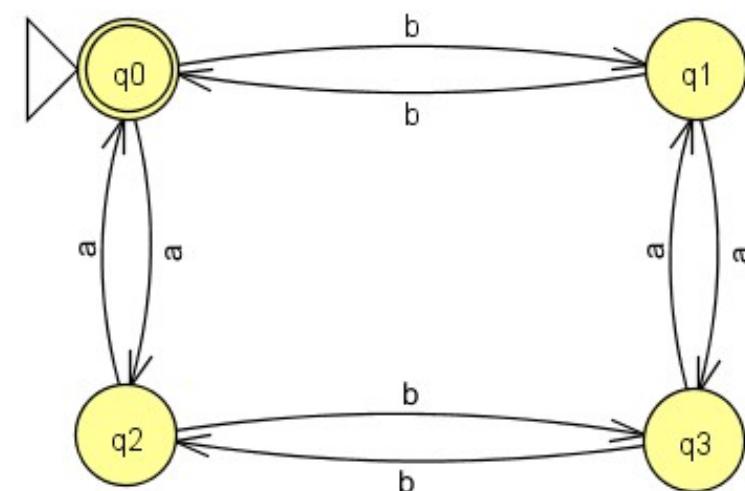
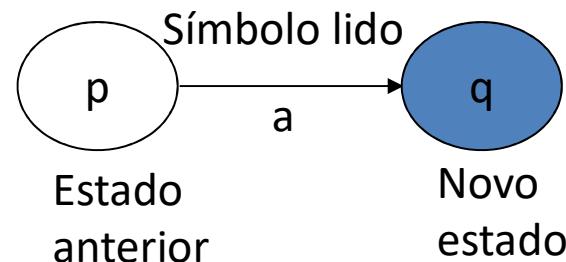
Na aula passada...

- Reconhecedor autômato finito- particularidades:
 1. Inexistência de memória auxiliar;
 2. Utilização do **cursor** da fita de entrada apenas para leitura de **símbolos**, não havendo operações de escrita sobre a fita;
 3. Movimentação do **cursor** de leitura em apenas um sentido, **da esquerda para a direita**;
 4. A **fita** de entrada possui **comprimento limitado**, suficiente apenas para acomodar a cadeia a ser analisada

Na aula passada...

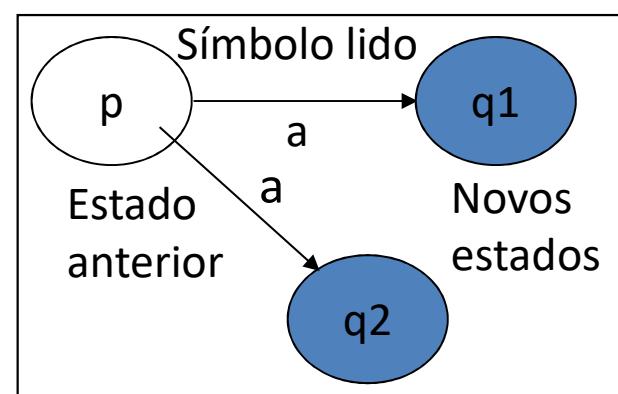
- Autômato finito determinístico

- Para cada entrada (símbolo) existe **um e somente um estado** ao qual o autômato pode transitar a partir de seu estado atual e do símbolo lido



Na aula de hoje...

- Autômato finito não-determinístico com e sem movimentos vazios
 - O autômato tem o poder de estar em **vários estados ao mesmo tempo**



No estado **p** ao ler o símbolo **a** assume **q1** e **q2** como novos estados atuais



AFND: aceitação e rejeição

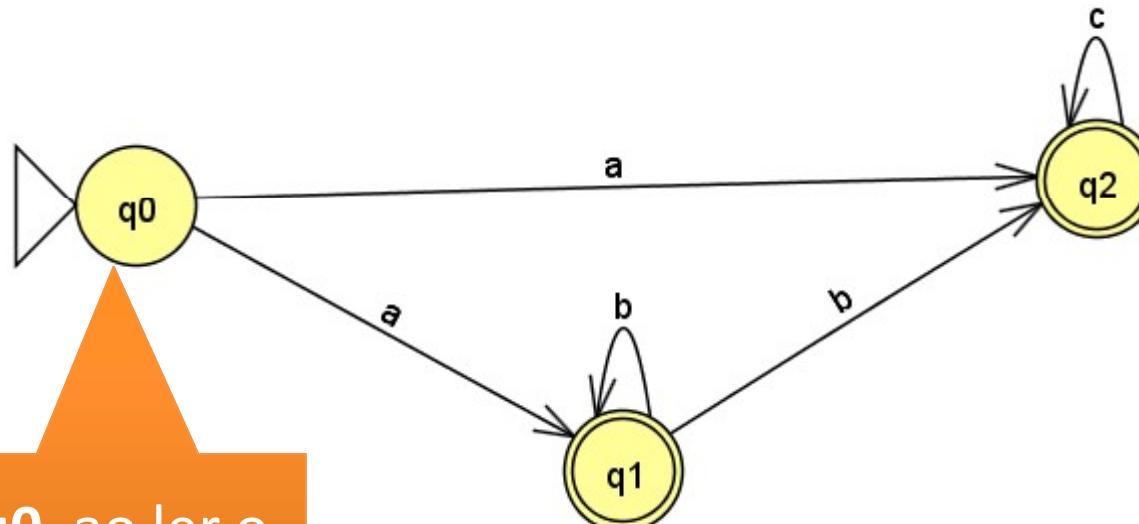
- Diz-se que um autômato finito não-determinístico **aceita** uma cadeia de entrada quando houver alguma sequência de movimentos que o leve da configuração inicial para uma configuração final.
- Diferentemente do autômato finito determinístico, em que essa sequência, se existir, é única para cada cadeia de entrada, no caso do **autômato finito não-determinístico** é possível que exista mais de uma sequência que satisfaça a essa condição para uma dada cadeia de entrada.
- Sempre que o **AFND** se deparar com mais de uma possibilidade de movimentação, **é feita a escolha (arbitrária)** de uma das alternativas; em caso de insucesso no reconhecimento, deve-se **considerar sucessivamente cada uma das demais alternativas ainda não consideradas**, até o seu esgotamento; persistindo o insucesso, e esgotadas as alternativas, diz-se que o autômato **rejeita** a cadeia.

Aceitação e rejeição de cadeias em AF's

	Dada uma cadeia de entrada, ele:		Aceita a cadeia de entrada se:	Rejeita a cadeia de entrada se:
AFD	Executa uma única sequência de movimentos.		Parar em uma configuração final.	Parar em uma configuração não-final.
AFND	Pode executar várias sequências distintas de movimentos.		Parar em uma configuração final.	Parar sem conseguir atingir nenhuma configuração final.

Exemplo de AFND

- Reconhece a linguagem ab^*c^*



No estado **q0**, ao ler o símbolo **a**, o autômato vai para os estados **q1** e **q2**

AFD versus AFND

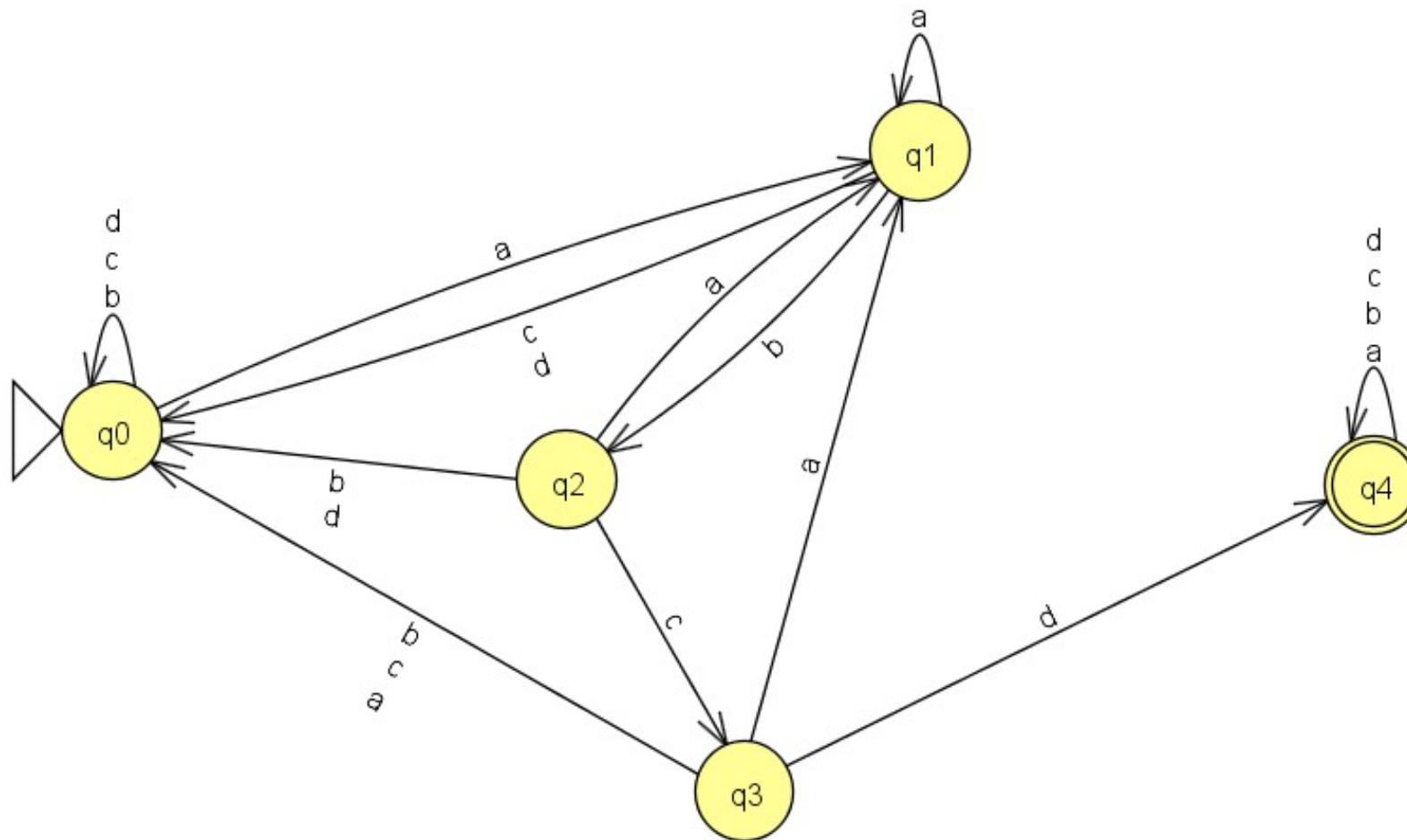
- os autômatos finitos não-determinísticos, em certos casos, podem mostrar-se mais simples de serem analisados do que as correspondentes versões determinísticas



EXEMPLO

AFD versus AFND

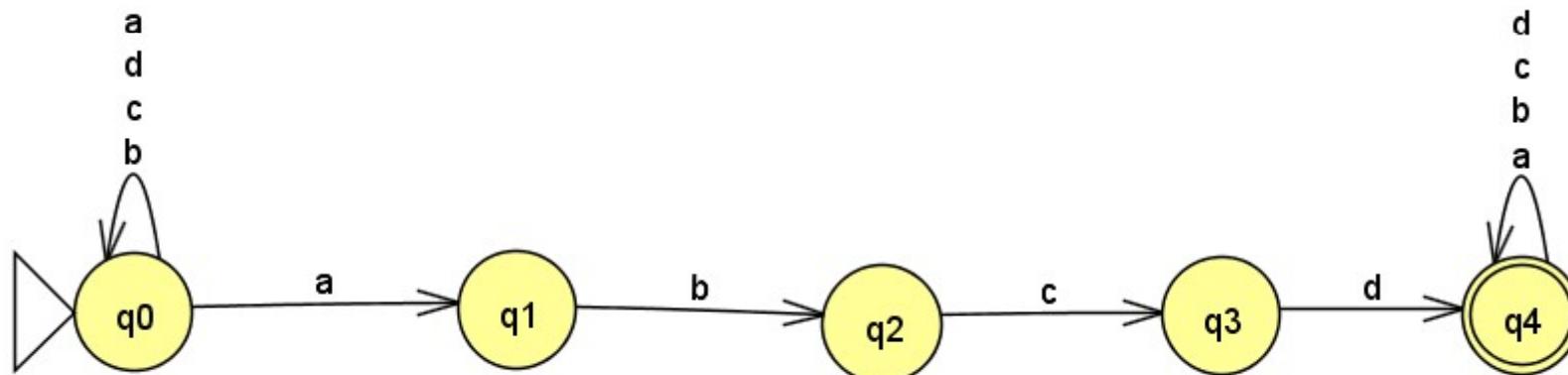
- AFD: Reconhece a linguagem $(a|b|c|d)^*abcd\ (a|b|c|d)^*$





AFD versus AFND

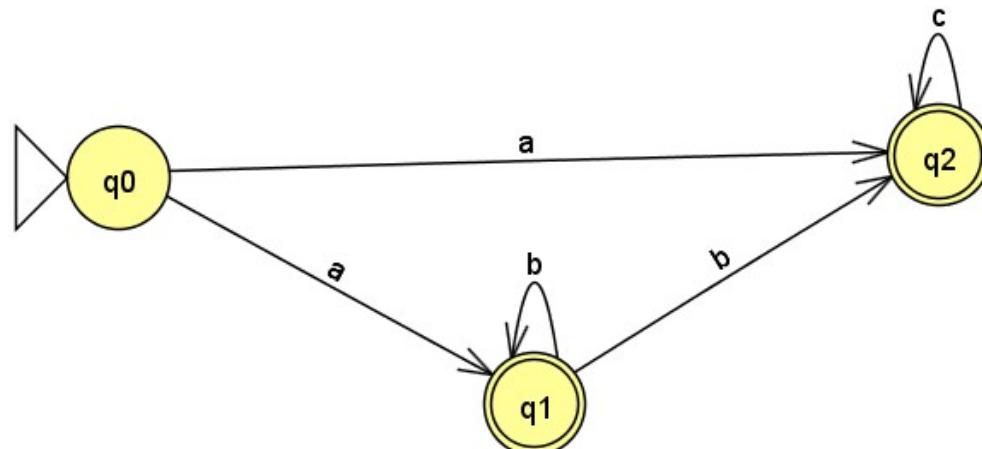
- AFND: Reconhece a mesma linguagem
 $(a|b|c|d)^*abcd \quad (a|b|c|d)^*$



Mais simples de ser analisado

AFND: notação tabular (tabela de transição)

- Cada **linha** da tabela representa um **estado** distinto q do autômato, e cada **coluna** é associada a um **elemento** distinto de seu **alfabeto** Σ de entrada. As células correspondentes à intersecção de cada linha com cada coluna são preenchidas com o elemento (conjunto) determinado pela função de transição



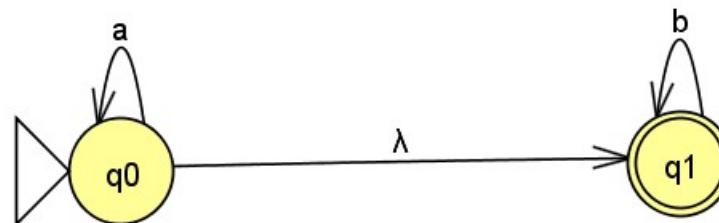
notação tabular

δ	a	b	c
$\rightarrow q_0$	$\{q_1, q_2\}$	\emptyset	\emptyset
$*q_1$	\emptyset	$\{q_1, q_2\}$	\emptyset
$*q_2$	\emptyset	\emptyset	$\{q_2\}$

**AFND com movimentos vazios (transições
em vazio) → AFNDE**

AFND com movimentos vazios (transições em vazio) → AFNDE

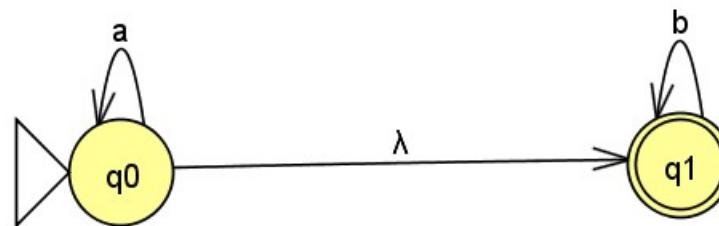
- AFND que apresentam transições em vazio são aqueles que admitem transições de um estado para outro com ϵ ou λ , além das transições normais, que utilizam os símbolos do alfabeto de entrada.



- Transições em vazio são executadas sem que seja necessário consultar o símbolo corrente da fita de entrada, e sua execução nem sequer causa o deslocamento do cursor de leitura.

AFNDE

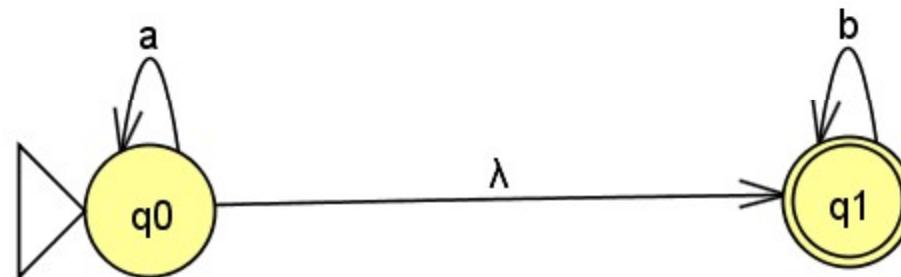
- Quando um autômato **transita em vazio**, isso significa que ele muda de estado sem consultar a cadeia de entrada.



- Sempre que ocorrer a simultaneidade entre alguma transição em vazio e outras transições (vazias ou não) com origem em um mesmo estado, isso acarreta a necessidade de uma escolha arbitrária da transição a ser aplicada na respectiva configuração, e isso, por sua vez, **caracteriza a manifestação de um não-determinismo**.

AFNDE

- Reconhece a cadeia a^*b^*



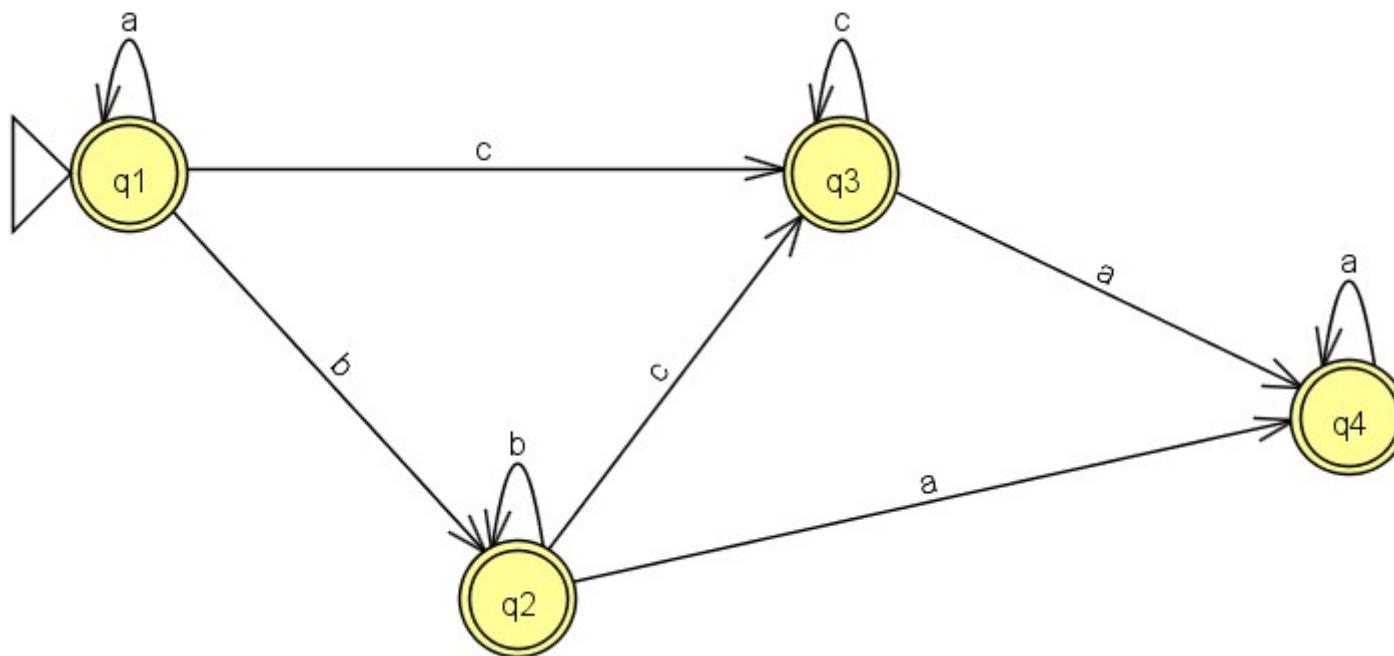
- Simule para a entrada ab

Uso de transições em vazio

- Assim como ocorre no caso dos AFD's e AFND's, alguns AFNDE se mostram mais simples de serem analisados do que as correspondentes versões isentas de transições em vazio.

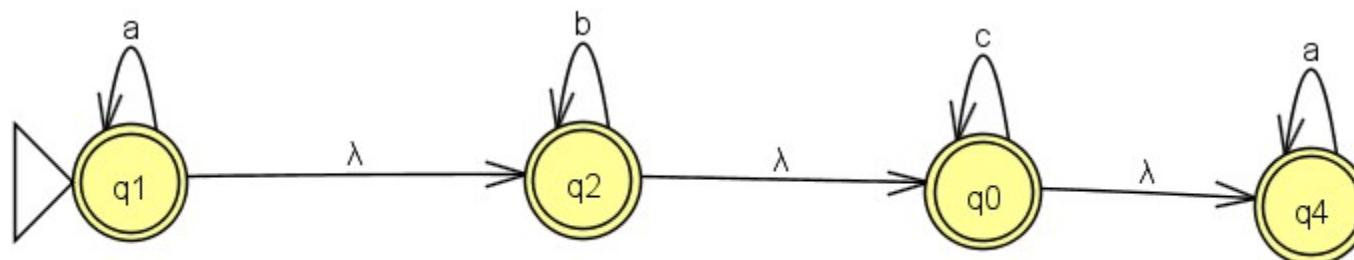
AF que reconhece a linguagem $a^*b^*c^*a^*$.

- sem movimento vazio



AF que reconhece a linguagem $a^*b^*c^*a^*$.

- com movimento vazio



AF com e sem movimento vazio - Linguagem

- Todo autômato com transições em vazio gera uma linguagem que é aceita por algum autômato finito que não contém transições em vazio

Eliminação de transições em
vazio

Eliminação de transições em vazio

- Algoritmo: "Obtenção de um autômato finito N , sem transições em vazio, a partir de um autômato finito M , com transições em vazio."
- Entrada: um autômato finito com transições em vazio M ;
- Saída: um autômato finito sem transições em vazio N , tal que $L(N) = L(M)$;
- Método:

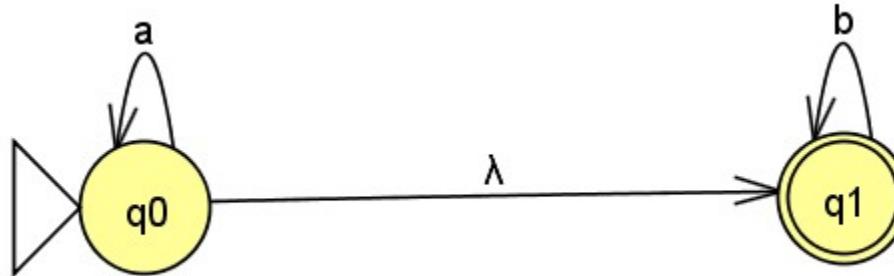
1. Eliminação das transições em vazio

- Considere-se um estado qualquer $q_i \in Q$. Se houver uma transição em vazio de q_i para q_j , deve-se eliminá-la, copiando-se para a linha que representa o estado q_i todas as transições que partem dos estados q_j para os quais é feita a transição em vazio.
- Esse procedimento corresponde, em notação tabular, à realização de uma fusão ("merge") entre a linha do estado q_i que contém a transição em vazio para o estado-destino q_j e a própria linha do estado q_j , armazenando-se o resultado novamente na linha correspondente ao estado q_i .
- Havendo mais de uma transição em vazio indicadas, deve-se repetir cumulativamente o procedimento para todas elas.
- Se $\delta(q_i, \lambda) \in F$, então $F' \leftarrow F' \cup \{q_i\}$, sendo que inicialmente $F' \leftarrow F$.

2. Iteração

- Repetir o passo anterior para os demais estados do autômato, até que todos eles tenham sido considerados (ou seja, até que a última linha tenha sido atingida).
- Nos casos em que houver transições em vazio para estados que por sua vez também transitam em vazio para outros estados, será necessário iterar o procedimento várias vezes sobre a tabela, até que todas as transições em vazio tenham sido eliminadas.

Exemplo: eliminação da transição em vazio



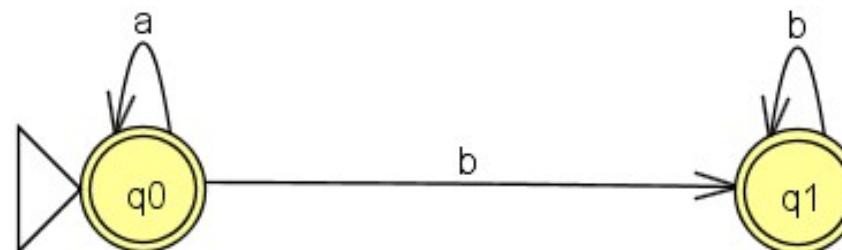
notação tabular

δ	a	b	λ
$\rightarrow q_0$	{q0,q1}	{q1}	{q1}
$*q_1$		{q1}	

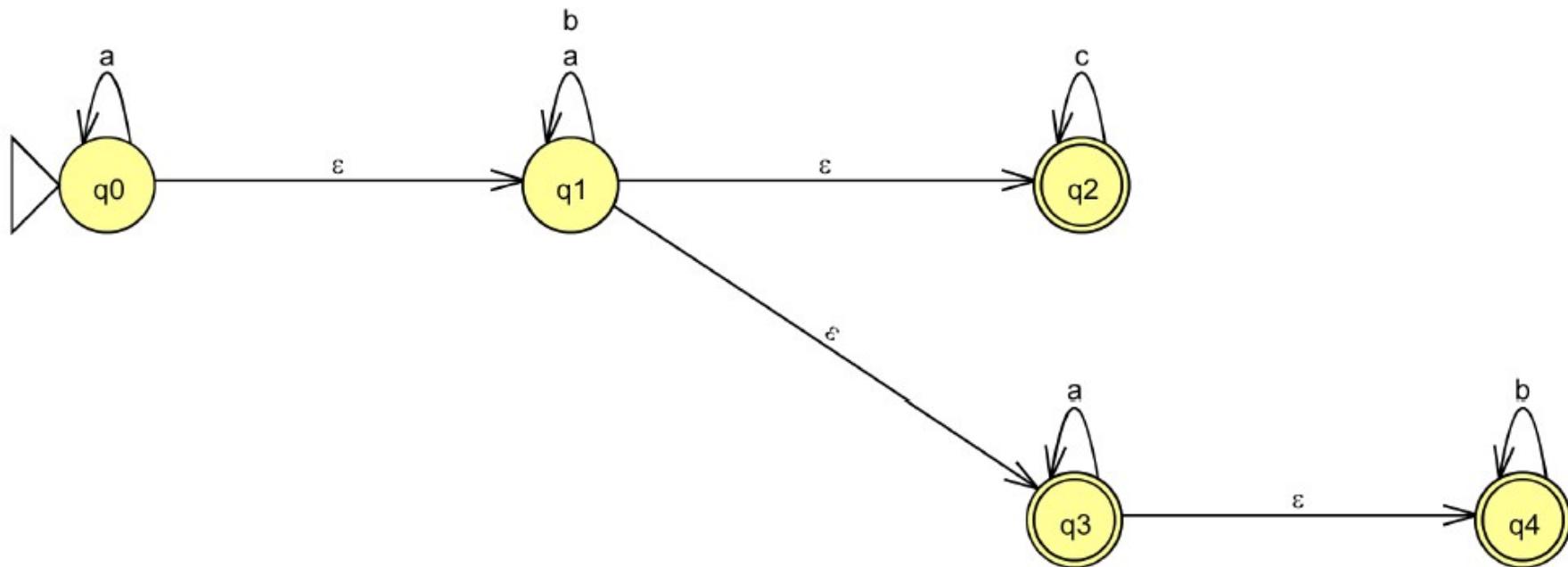
Como há uma transição em vazio de q_0 para q_1 , deve-se copiar as transições de q_1 para q_0 (apenas, neste caso) e, além disso, considerar q_0 como estado final, uma vez que q_1 é estado final. Abaixo, a tabela e o AF sem movimento vazio.

notação tabular

δ	a	b
$\rightarrow *q_0$	{q0}	{q1}
$*q_1$		{q1}



Exercício: Obter um AF sem movimento vazio





Exercícios

1. Construir 2 AFND's usando diagrama de transições (1 com movimento vazio e 1 sem movimento vazio), 1 ER e 1 GR que reconheça cada uma das linguagens sobre o alfabeto $\{a,b\}$ e cujas sentenças estão descritas a seguir.

a. Começam com aa;	h. Possuem comprimento diferente de 3;
b. Não começam com aa;	i. Possuem comprimento par;
c. Terminam com bbb;	j. Possuem comprimento ímpar;
d. Não terminam com bbb;	k. Possuem comprimento múltiplo de 4;
e. Contém a subcadeia aabbb;	l. Possuem quantidade par de símbolos a;
f. Possuem comprimento maior ou igual a 3;	m. Possuem quantidade ímpar de símbolos b.
g. Possuem comprimento menor ou igual a 3;	

2. Escolha três sentenças qualquer e aplique o algoritmo de conversão de AFNDE para AFND
3. Escolha três sentenças (exceto a e b) e construa o AFND (com ou sem ϵ) na forma de tabela de transições



Exercícios: Construa AFND's com ou sem movimentos vazios para as linguagens abaixo. Para os que apresentarem movimentos vazios, aplique o algoritmo de eliminação de movimentos vazios.

L1 = Aceita cadeias $\in \{1,2\}^*$ tal que o último símbolo na cadeia tenha aparecido anteriormente. Em seguida, construa a Tabela de Transição de Estados e a Função de Transição de Estados

L2 = 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. Em seguida, construa a Tabela de Transição de Estados e a Função de Transição de Estados

L3 = { $w | w \in \{a,b,c\}^*, aa$ ou bb é subpalavra e $cccc$ é sufixo de w }

L4 = { $w | w \in \{a,b\}^*$ e o quarto símbolo da direita para a esquerda de w é a }

L5 = { $w_1w_2w_1 | w_1, w_2 \in \{0,1\}^* e |w_1|=2$ }



L6 - o conjunto de strings sobre o alfabeto $\{0,1,\dots,9\}$ tal que o dígito final não tenha aparecido antes

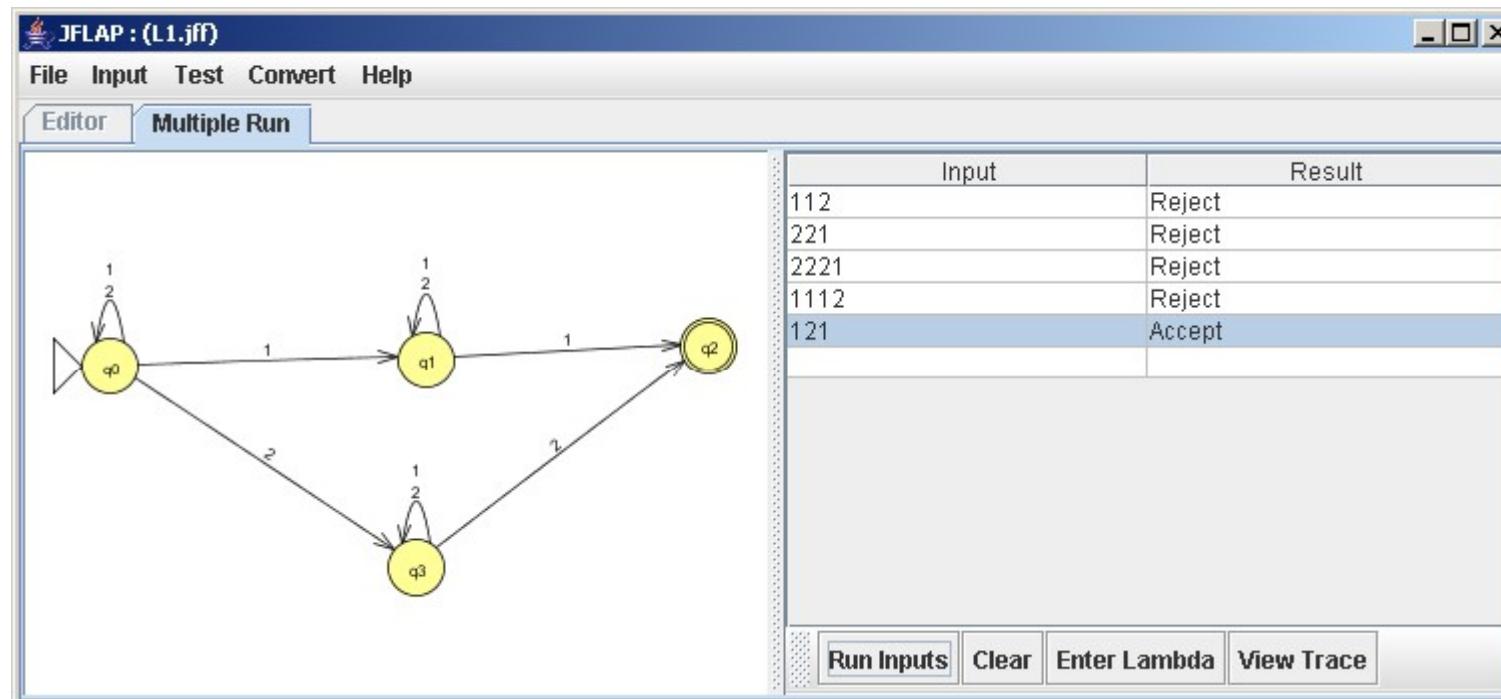
L7 - o conjunto de strings de 0's e 1's tais que não existam dois 0's separados por um número de posições que seja múltiplo de 4. Observe que 0 é um múltiplo permitido de 4.

L8 - Defina um AFND que tenha no máximo cinco estados e cuja linguagem seja o conjunto das sequências de x's, y's e z's do tipo $\alpha z \beta$ ou $\gamma x \beta$ onde $\alpha \beta \gamma \in \{x,y,z\}^*$, α é não vazia e não contém y's, β é não vazia e não contém z's e γ tem pelo menos um y e não contém z's.

- Verifique se a sequência xzx pertence a L8.

Possível resposta

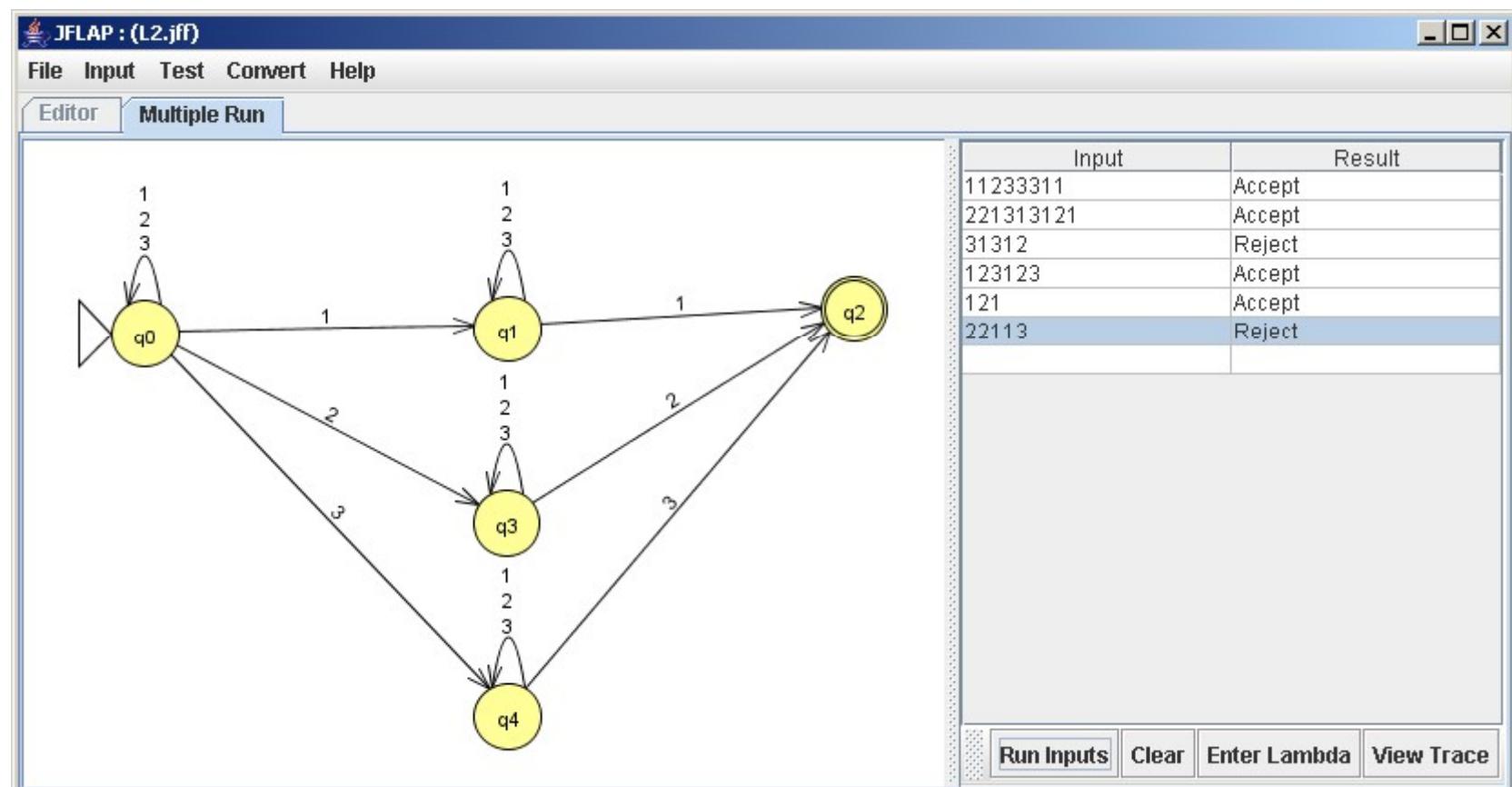
- L_1



$$A = (\{q_0, q_1, q_2, q_3\}, \{1, 2\}, \delta, q_0, \{q_2\})$$

Possível resposta

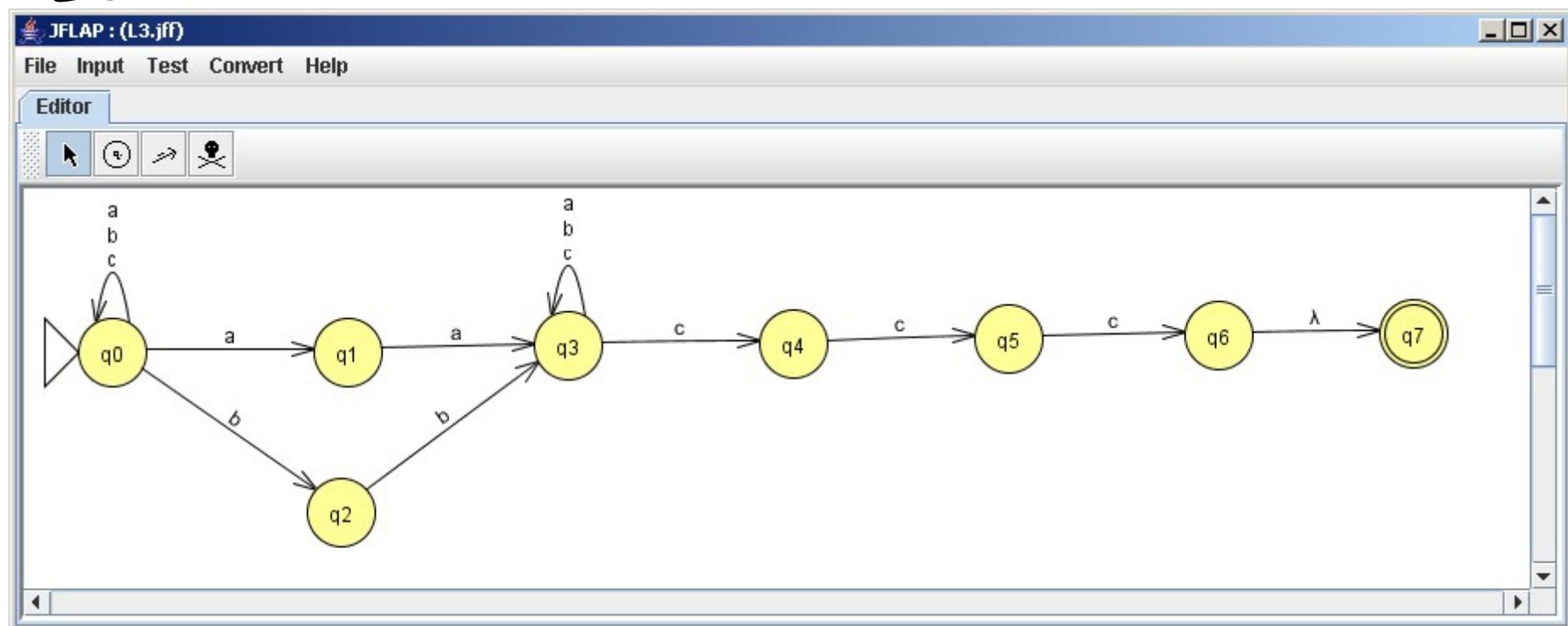
- L2



$$A = (\{q_0, q_1, q_2, q_3, q_4\}, \{1, 2, 3\}, \delta, q_0, \{q_2\})$$

Possível resposta

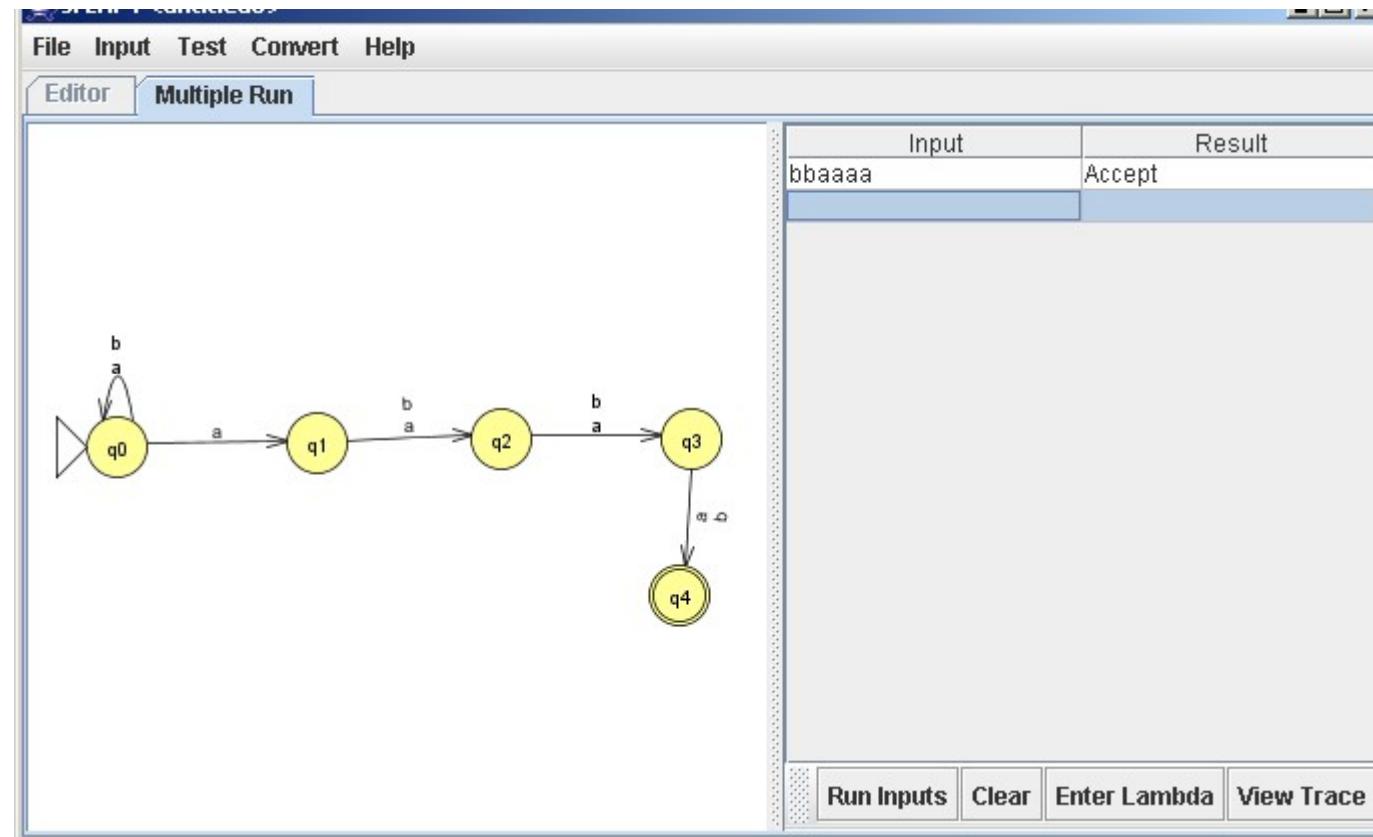
- L3



$$A = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}, \{a, b, c\}, \delta, q_0, \{q_7\})$$

Possível resposta

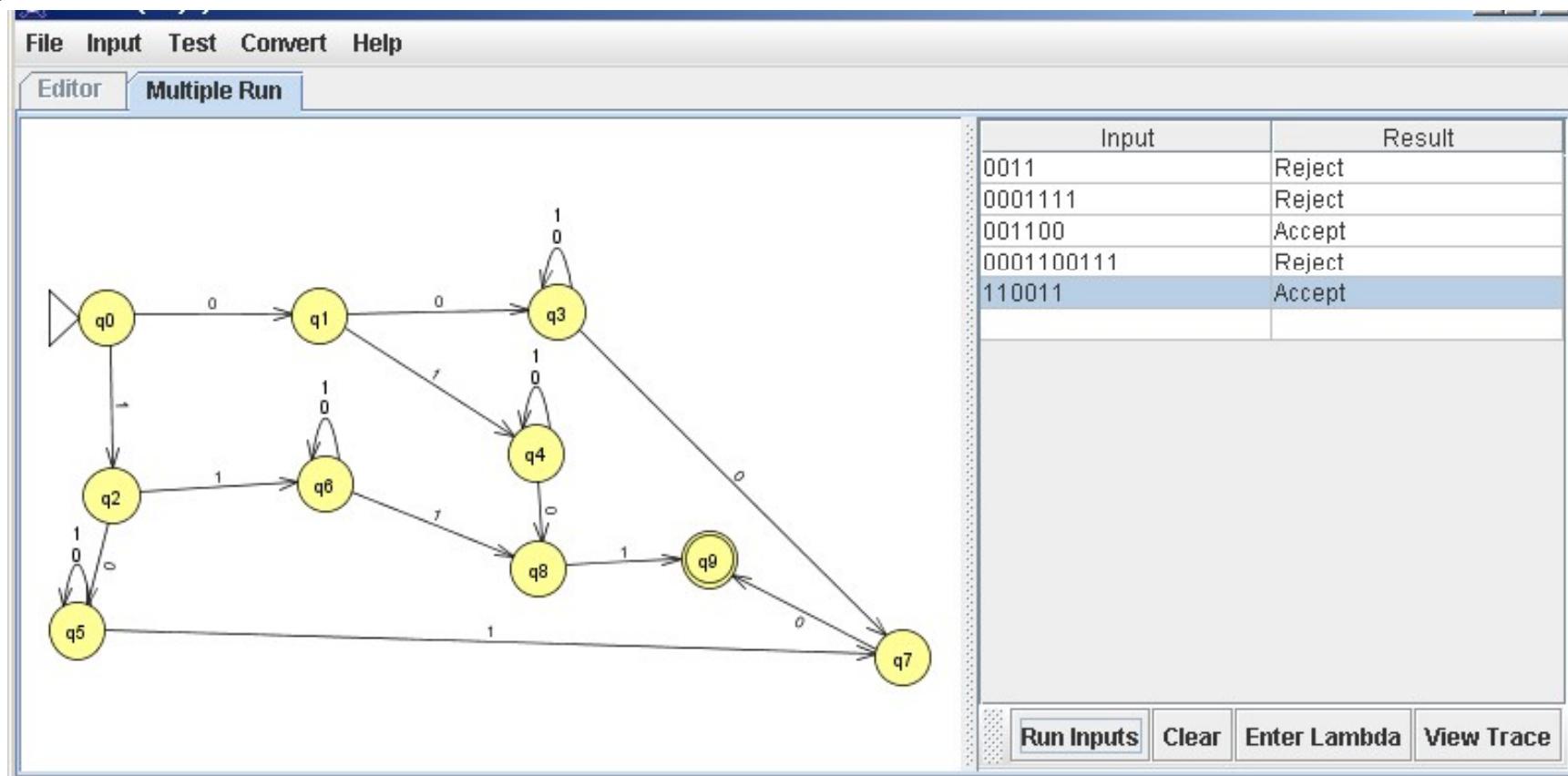
- L4



$$A = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \delta, q_0, \{q_4\})$$

Possível resposta

- L5



$$A = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9\}, \{0, 1\}, \delta, q_0, \{q_9\})$$

LFA - Aula 06

Equivalência entre AFD e AFND

Equivalência entre ER's e AF's

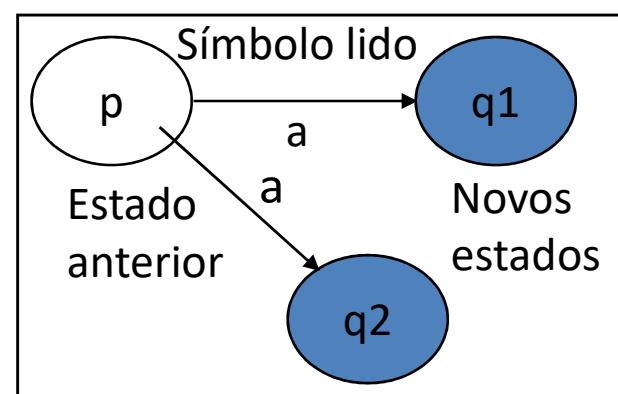
Equivalência entre GR's e AF's

Celso Olivete Júnior

celso.olivete@unesp.br

Na aula passada...

- Autômato finito não-determinístico com e sem movimentos vazios
 - O autômato tem o poder de estar em **vários estados ao mesmo tempo**



No estado **p** ao ler o símbolo **a** assume **q1** e **q2** como novos estados atuais

Na aula de hoje:

- Equivalência entre AFND e AFD
- Conversões:
 - ER's em AF's e AF's em ER's
 - GR's em AF's e AF's em GR's
- Referência bibliográfica
 - HOPCROFT, J. E.; ULLMAN, J. D.; MOTWANI, R. *Introdução à Teoria de Autômatos, Linguagens e Computação*. Editora Campus, 2002 → Capítulos 2 e 3

Equivalência entre AFD's e AFND's

Equivalência entre AFD e AFND

- Teorema: Seja L o conjunto aceito por um AFND, então existe um AFD que aceita L - são equivalentes.
- Embora muitas vezes seja mais fácil construir um AFND para uma L , o AFD tem na prática quase o mesmo número de estados que um AFND, embora ele tenha mais transições.
- No pior caso, o menor AFD pode ter 2^n estados, enquanto o menor AFND (para a mesma linguagem) tem apenas n estados

Equivalência entre AFD e AFND

- A prova de que os AFD's podem fazer tudo o que os AFND's podem fazer envolve a **construção de subconjuntos**
- A construção dos subconjuntos começa a partir de um AFND $N = (Q_n, \Sigma, \delta_n, \{q_0\}, F_n)$. O objetivo é a descrição de um AFD $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D) \rightarrow$ tal que $L(D) = L(N)$
 - Σ é o mesmo
 - O estado inicial de D é o conjunto que contém apenas o estado inicial de N

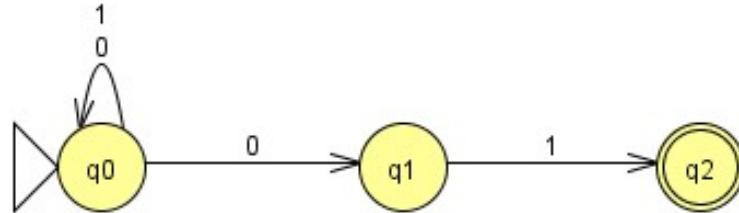
Equivalência entre AFD e AFND

- Construção dos outros elementos de D
 - Q_D é o conjunto de subconjuntos de Q_N
 - Q_D representa o conjunto de potências de Q_N . Ex:
 - Se Q_N tem n estados Q_D terá 2^n estados (no pior caso)
 - F_D é o conjunto de subconjuntos de S de $Q_N \rightarrow$ representa todos os conjuntos de estados de N que incluem pelo menos um estado de aceitação de N

$$\delta_D(S,a) = \bigcup \delta_N(p,a)$$

- Para calcular $\delta_D(S,a)$, basta observar todos os estados de p em S , e ver para quais estados N vai para p sobre a entrada “ a ” e fazemos a união de todos esses estados.

AFND aceita cadeia de 0's e 1's com final 01



Estados de N → {q0,q1,q2}

- 3 estados: logo envolverá a construção de $2^3 = 8$ subconjuntos. O AFD terá no máximo 8 estados.

Construção dos subconjuntos

Entrada

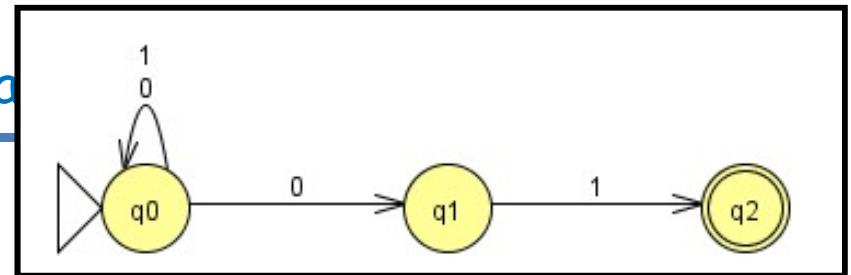
Estado	0	1
∅	∅	∅
→{q0}	{q0,q1}	{q0}
{q1}	∅	{q2}
*{q2}	∅	∅
{q0,q1}	{q0,q1}	{q0,q2}
*{q0,q2}	{q0,q1}	{q0}
*{q1,q2}	∅	{q2}
*{q0,q1,q2}	{q0,q1}	{q0,q2}

Renomeando os subconjuntos

Entrada

Estado	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

Equivalência entre AFD e AFND



Eliminando estados inacessíveis

A partir de B (est.inicial) só é possível chegar em B, E e F. Os outros estados são inacessíveis e devem ser removidos.

Para cada entrada a calcula-se $\delta_D(S,a)$ → obtêm os acessíveis

Para o exemplo anterior

$$B \quad \delta_D(\{q0\}, 0) = \{q0, q1\}$$

$$\delta_D(\{q0\}, 1) = \{q0\}$$

$$E \quad \delta_D(\{q0, q1\}, 0) = \{q0, q1\}$$

$$\delta_D(\{q0, q1\}, 1) = \{q0, q2\}$$

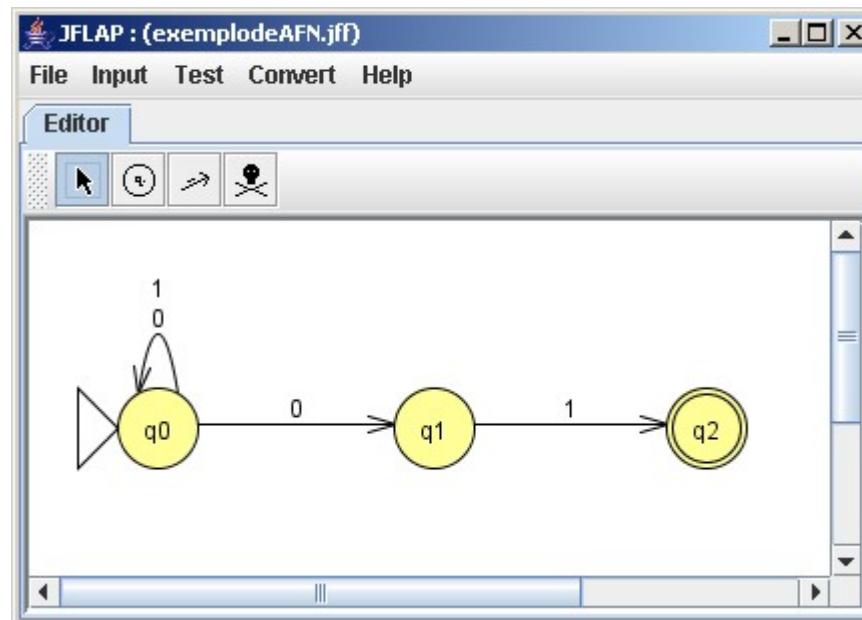
$$F \quad \text{Pois: } \delta_D(\{q0, q2\}, 0) = \delta_N(\{q0\}, 0) \cup \delta_N(\{q2\}, 0) \rightarrow \{q0, q1\} \cup \emptyset = \{q0, q1\}$$

$$\delta_D(\{q0, q2\}, 1) = \delta_N(\{q0\}, 1) \cup \delta_N(\{q2\}, 1) \rightarrow \{q0\} \cup \emptyset = \{q0\}$$

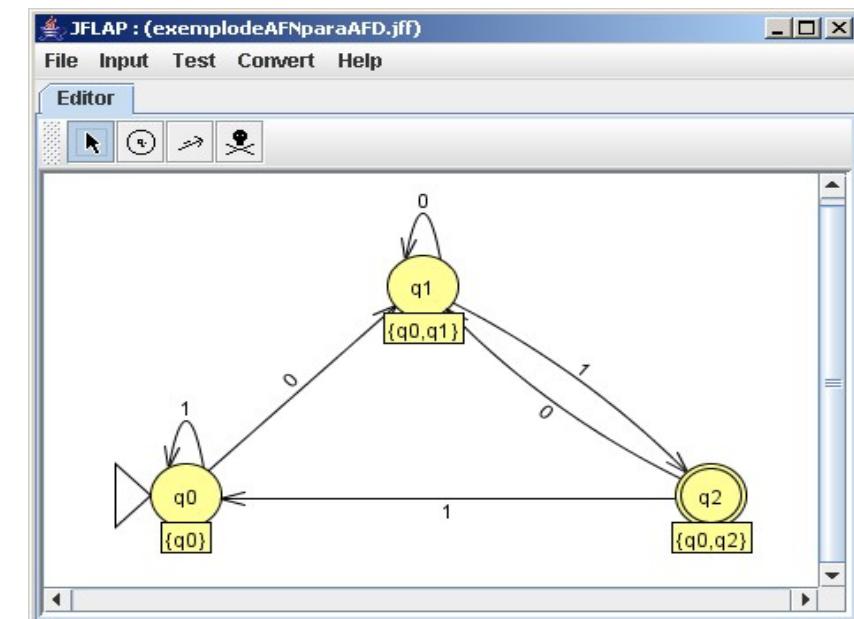
Como $\{q0, q1\}$ e $\{q0\}$ já foram encontrados → a simplificação pára (convergiu), conhecemos todos os estados acessíveis e suas transições

Equivalência entre AFD e AFND

AFND



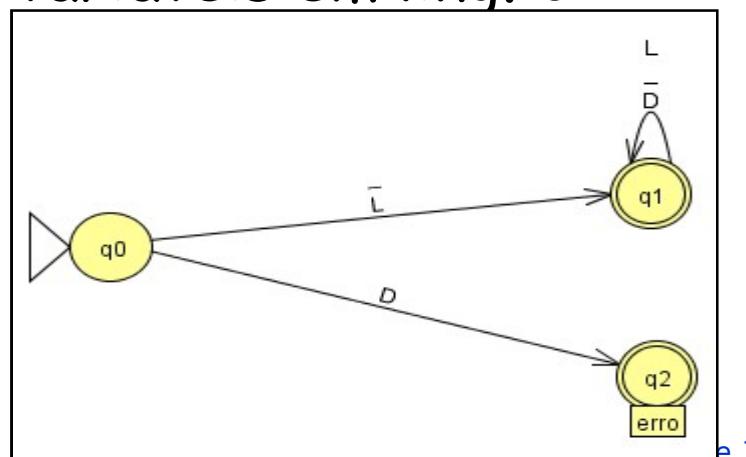
AFD



- mesmo número de estados, porém o AFD tem um número maior de transições

Prevendo entradas erradas

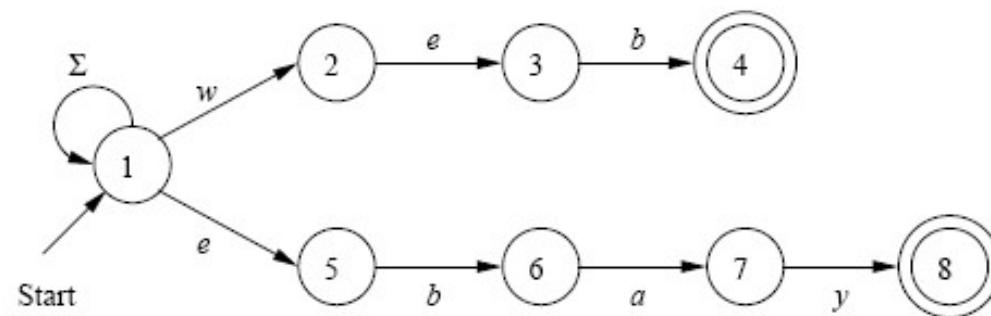
- A definição de um AF EXIGE que todo estado tenha uma transição para cada símbolo ($\in \text{ao } \Sigma$) lido da entrada
- Podemos criar um estado de não aceitação (erro) para prever um possível dado inválido de uma determinada linguagem (AF “morre”). Ex: reconhece identificadores de variáveis em ling. C



$$\begin{aligned}L &= \{A..Z, a..z\} \\D &= \{0..9\}\end{aligned}$$

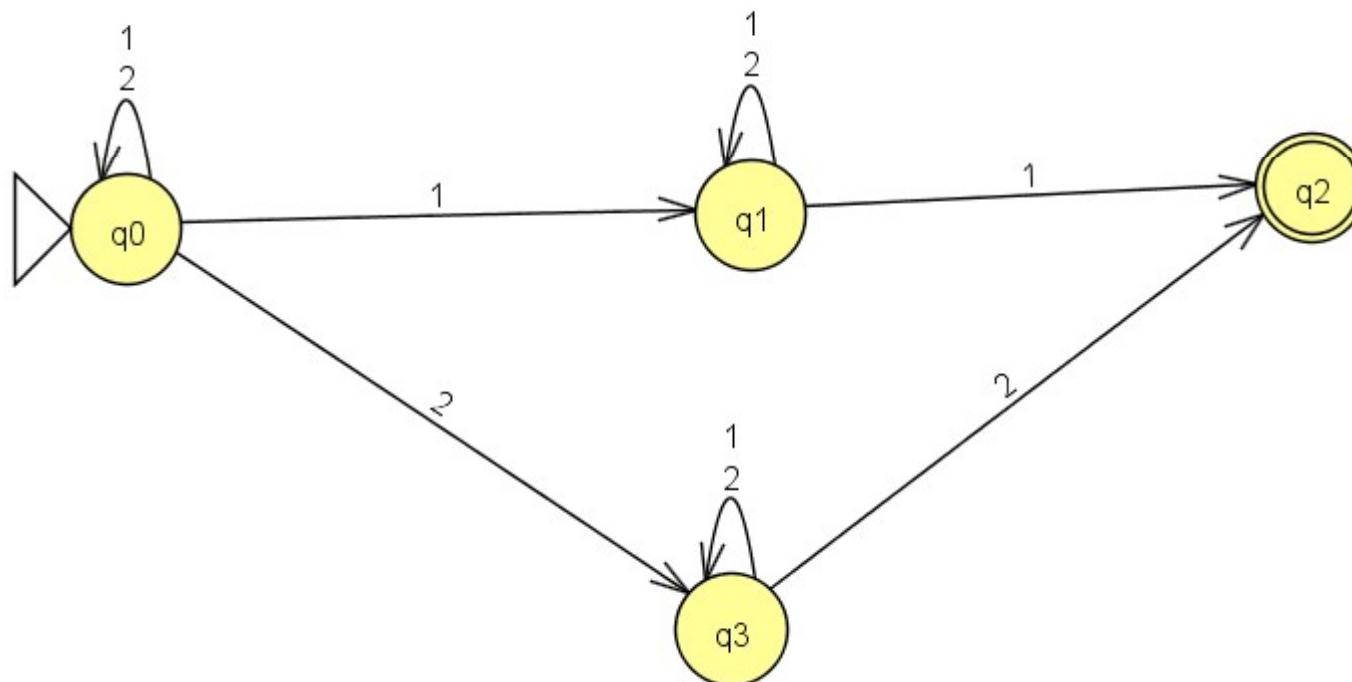
AFND para busca em textos

- Exemplo: Através de palavras-chave encontrar ocorrência de quaisquer dessas palavras em um repositório de documentos on-line.
 - Passos
 - O texto do documento é transferido um caractere de cada vez
 - O AFD deverá reconhecer as palavras-chave. Ex: web, ebay



Exercícios

1. Converta o AFND para AFD



Exercícios

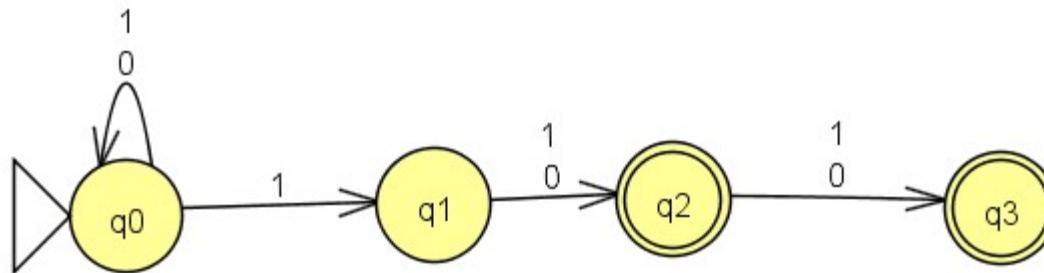
2. Construa um AFND (sem mov. vazios) que seja capaz de reconhecer as seguintes palavras-chave {he, she, hers, his}. Utilize tratamento de erros, por exemplo: caso seja encontrado um caractere inválido ($\Sigma=\{h, e, r, s, i\}$) retorne para o estado inicial, reiniciando a leitura.
3. Converta para AFD o exercício anterior.

4. Seção 2.3 (páginas 71 a 73)

5. Seção 2.4 (página 78)

Equivalência entre AF's e ER's

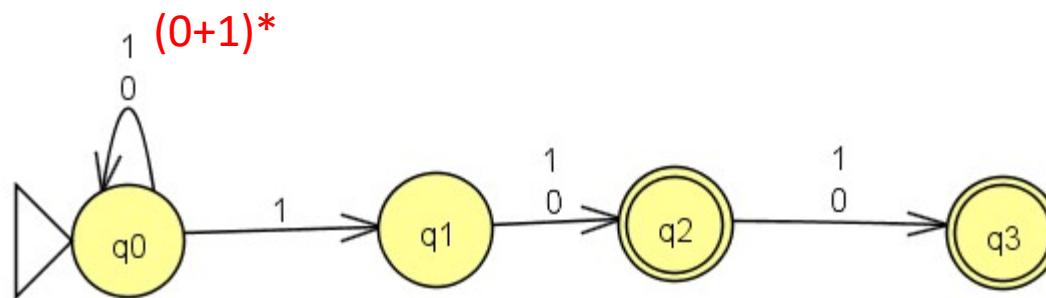
Conversão de AF para ER



ER correspondente. Passos:

1. Encontrar a ER para cada estado
2. Encontrar, a partir da união de todas as ER's, uma ER que vai do estado inicial para o estado final
3. Caso tenha mais de um estado final, a ER resultante será a união das ER's obtidas no passo 2

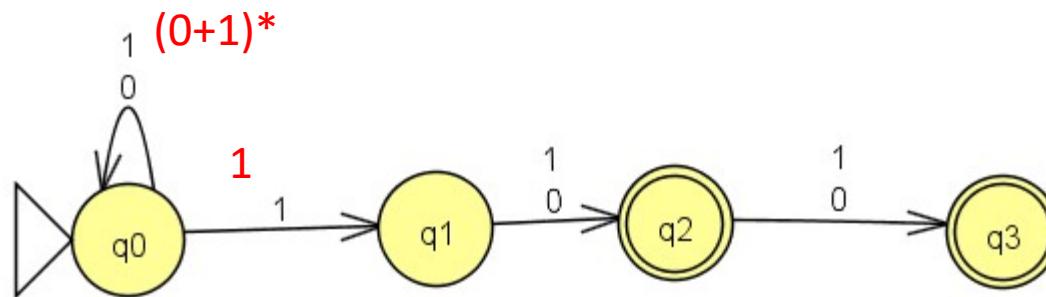
Conversão de AF para ER



ER correspondente. Passos:

1. Encontrar a ER para cada estado
2. Encontrar, a partir da união de todas as ER's, uma ER que vai do estado inicial para o estado final
3. Caso tenha mais de um estado final, a ER resultante será a união das ER's obtidas no passo 2

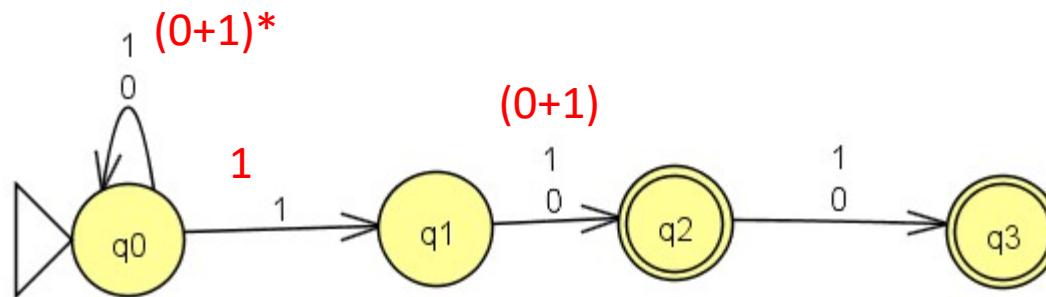
Conversão de AF para ER



ER correspondente. Passos:

1. Encontrar a ER para cada estado
2. Encontrar, a partir da união de todas as ER's, uma ER que vai do estado inicial para o estado final
3. Caso tenha mais de um estado final, a ER resultante será a união das ER's obtidas no passo 2

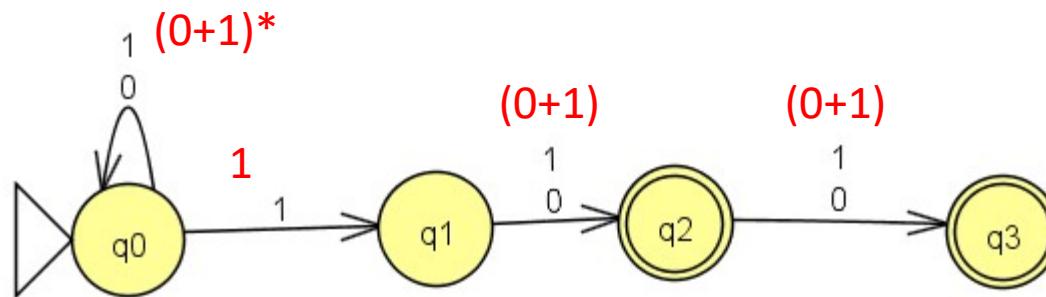
Conversão de AF para ER



ER correspondente. Passos:

1. Encontrar a ER para cada estado
2. Encontrar, a partir da união de todas as ER's, uma ER que vai do estado inicial para o estado final
3. Caso tenha mais de um estado final, a ER resultante será a união das ER's obtidas no passo 2

Conversão de AF para ER

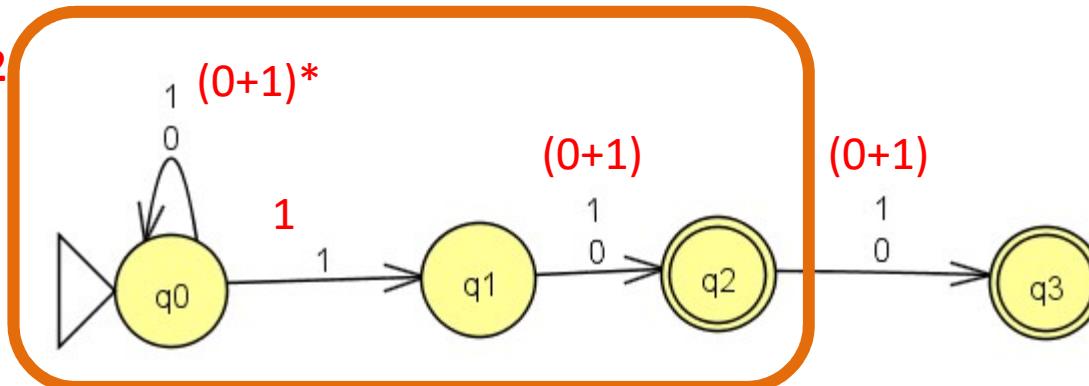


ER correspondente. Passos:

1. Encontrar a ER para cada estado
2. Encontrar, a partir da união de todas as ER's, uma ER que vai do estado inicial para o estado final
3. Caso tenha mais de um estado final, a ER resultante será a união das ER's obtidas no passo 2

Conversão de AF para ER

Passo 2
ER1

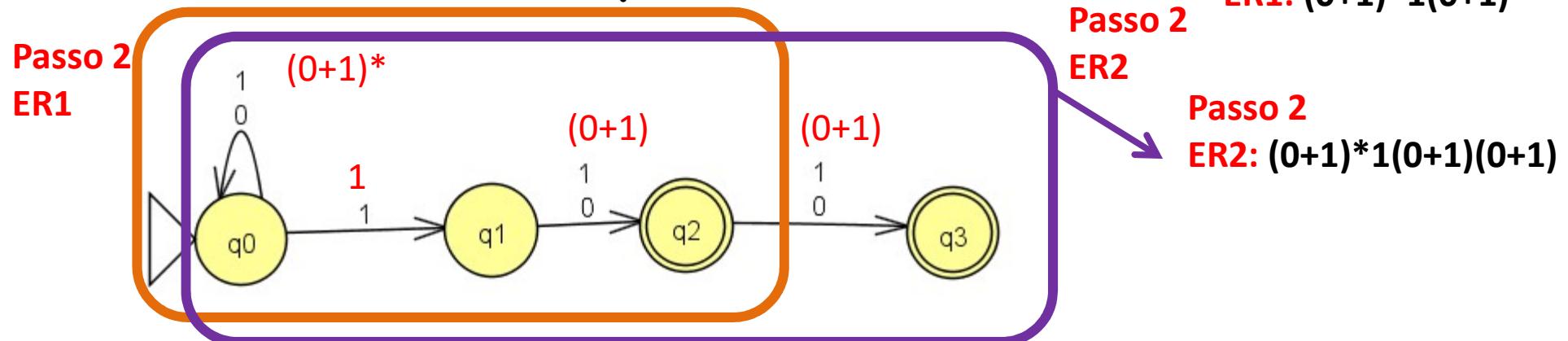


Passo 2
ER1: $(0+1)^*1(0+1)$

ER correspondente. Passos:

1. Encontrar a ER para cada estado
2. Encontrar, a partir da união de todas as ER's, uma ER que vai do estado inicial para o estado final
3. Caso tenha mais de um estado final, a ER resultante será a união das ER's obtidas no passo 2

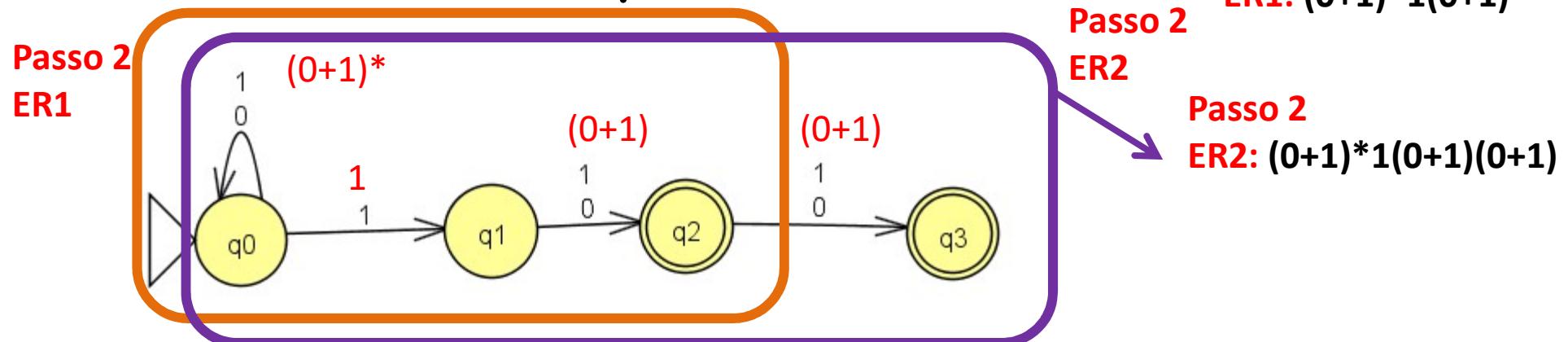
Conversão de AF para ER



ER correspondente. Passos:

1. Encontrar a ER para cada estado
2. Encontrar, a partir da união de todas as ER's, uma ER que vai do estado inicial para o estado final
3. Caso tenha mais de um estado final, a ER resultante será a união das ER's obtidas no passo 2

Conversão de AF para ER



ER correspondente. Passos:

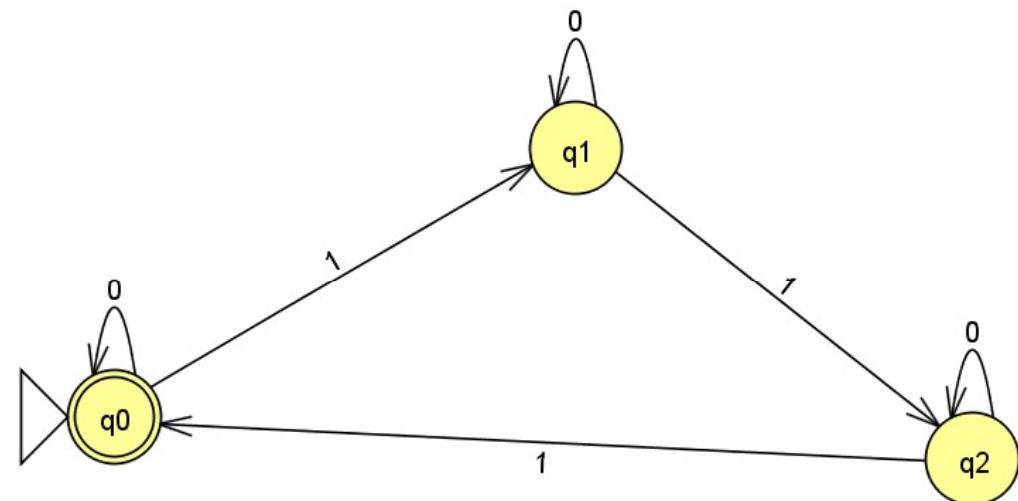
1. Encontrar a ER para cada estado
2. Encontrar, a partir da união de todas as ER's, uma ER que vai do estado inicial para o estado final
3. Caso tenha mais de um estado final, a ER resultante será a união das ER's obtidas no passo 2

ER final. Passo 3

ER: $ER1 \cup ER2$

ER= $(0+1)^*1(0+1) + (0+1)^*1(0+1)(0+1)$

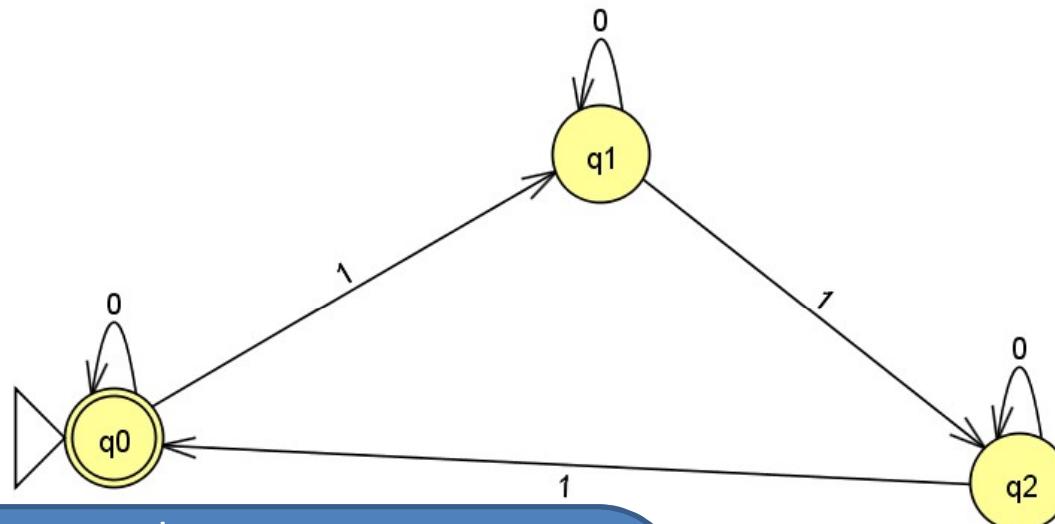
Dado o AFD que reconhece a $L=\{0^n1^m \mid n \geq 0 \text{ e } m \text{ é múltiplo de } 3\}$, encontre a ER correspondente.



ER correspondente. Passos:

1. Encontrar a ER para cada estado
2. Encontrar, a partir da união de todas as ER's, uma ER que vai do estado inicial para o estado final
3. Caso tenha mais de um estado final, a ER resultante será a união das ER's obtidas no passo 2

Dado o AFD que reconhece a $L=\{0^n1^m \mid n \geq 0 \text{ e } m \text{ é múltiplo de } 3\}$, encontre a ER correspondente.



ER
 $0^* + (0^*10^*10^*1)^*$

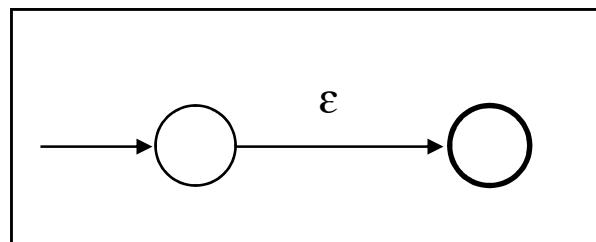
ER correspondente. Passos:

1. Encontrar a ER para cada estado
2. Encontrar, a partir da união de todas as ER's, uma ER que vai do estado inicial para o estado final
3. Caso tenha mais de um estado final, a ER resultante será a união das ER's obtidas no passo 2

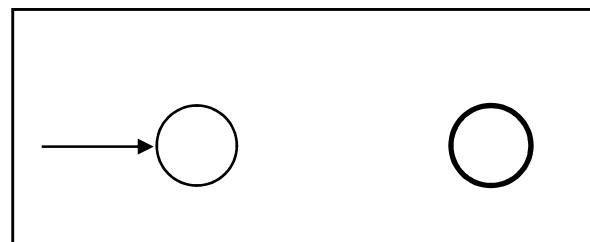
Conversão de ER's para AF's

Conversão de ER's para AF's

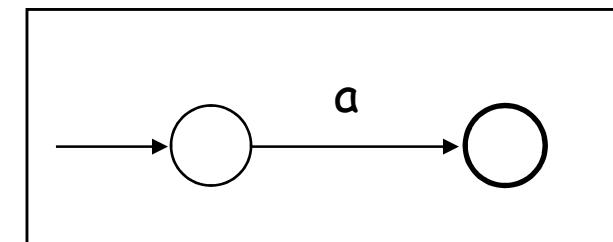
- Toda linguagem definida por um ER também é definida por um AF.
- Construção de um AF a partir de uma ER → componentes básicos:



$$ER = \varepsilon$$



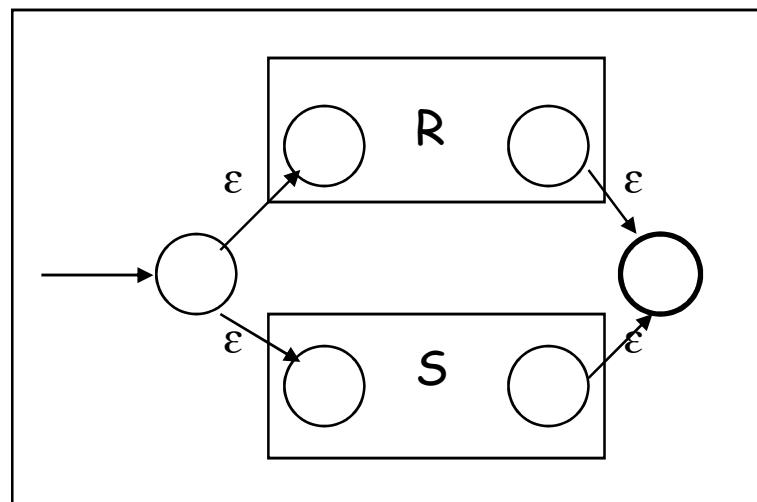
$$ER = \theta$$



$$ER = a$$

Conversão de ER's para AF's

Caso a ER tenha mais de um operador
(união, concatenação e fechamento).

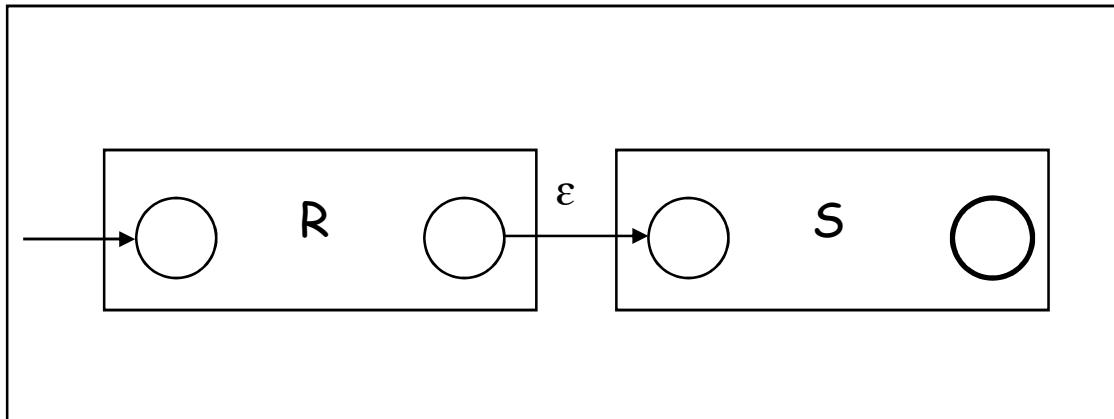


União

$$ER = R + S$$

Conversão de ER's para AF's

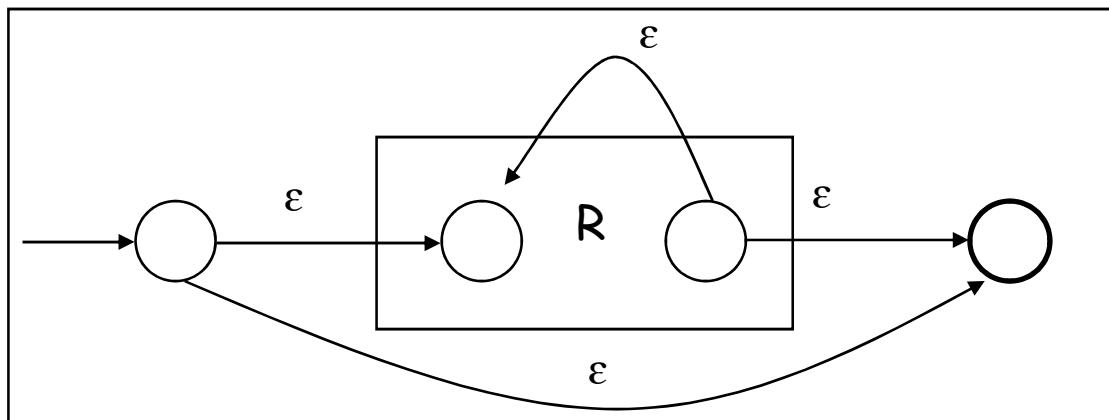
Caso a ER tenha mais de um operador
(união, concatenação e fechamento).



Concatenação
 $ER = RS$

Conversão de ER's para AF's

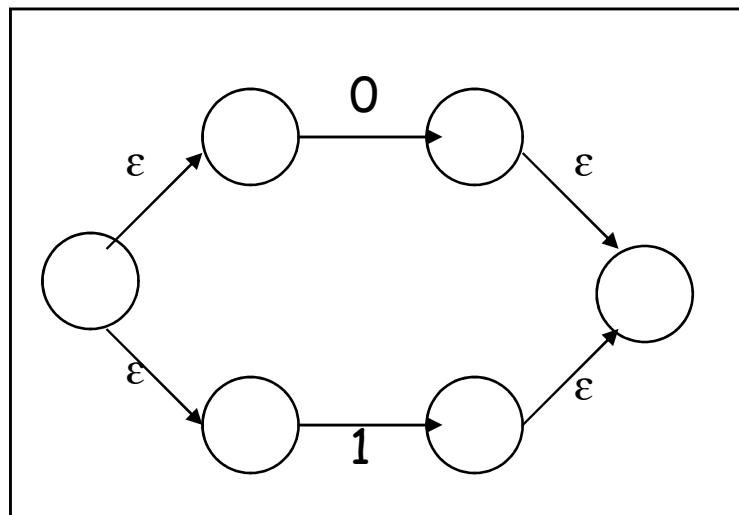
Caso a ER tenha mais de um operador
(união, concatenação e **fechamento**).



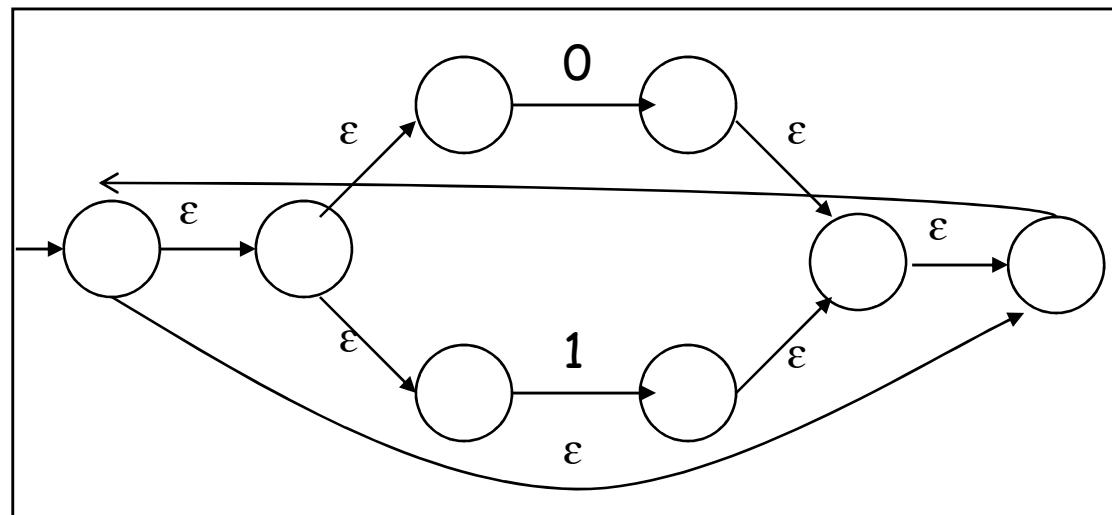
Fechamento
 $ER = R^*$

Conversão de ER's para AF's

Converter a ER = $(0 + 1)^* 1(0 + 1)$ em um AFND com movimentos vazios.



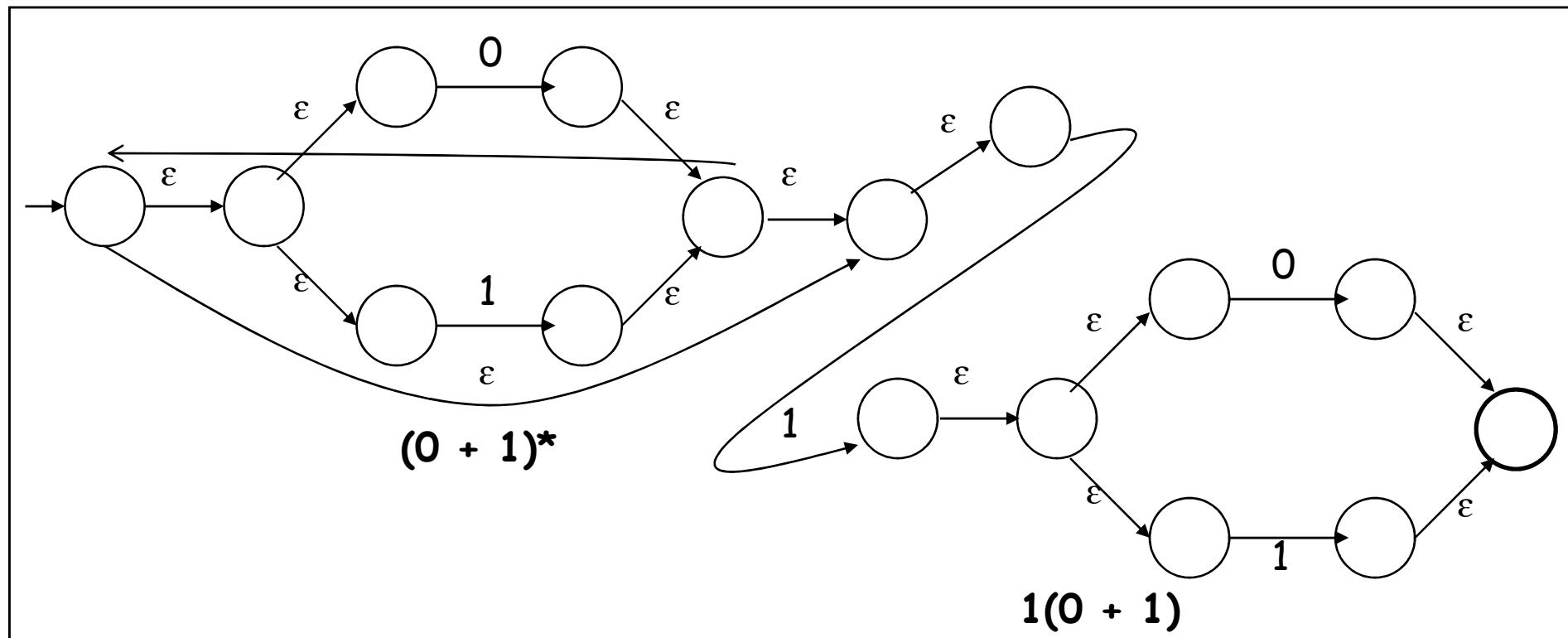
$0 + 1$



$(0 + 1)^*$

Conversão de ER's para AF's

Converter a ER = $(0 + 1)^* \ 1(0 + 1)$ em um AFND com movimentos vazios.



Exercícios

5. Da Seção 3.2 (exercícios 3.2.1, 3.2.2, 3.2.3
e 3.2.4) - páginas 113 a 115

Conversão entre GR's e AF's

Equivalência entre GR's e AF's

- Dada uma **gramática linear à direita** é possível construir um **autômato finito** capaz de reconhecer a mesma linguagem.

Seja G uma gramática linear à direita. Então é possível definir um autômato finito M de tal modo que $L(G) = L(M)$.

Equivalência entre GR's e AF's

- Dada a gramática $G = (V, T, P, S)$, onde P são do tipo:

1. $X \rightarrow aY$
 2. $X \rightarrow Y$
 3. $X \rightarrow a$
 4. $X \rightarrow \epsilon$
- com $S, Y \in V$, $a \in T$



Algoritmo de equivalência entre GR's e AF's

• Algoritmo de conversão $GR \rightarrow AF$

• Entrada: uma gramática linear à direita G ;

• Saída: um autômato finito M tal que $L(M) = L(G)$;

• Método:

1. Conjunto de estados:

- Cada estado de M corresponde a um dos símbolos não-terminais de G . A esse conjunto acrescenta-se um novo símbolo (estado) $Z \notin V$, ou seja, $\{Q\} = V \cup \{Z\}$. O estado inicial de M é S , a raiz da gramática. O estado final de M é Z , o novo estado acrescentado.

2. Alfabeto de entrada:

- O alfabeto de entrada Σ de M é o mesmo alfabeto Σ de G .

1. $X \rightarrow aY$

2. $X \rightarrow Y$ $G = (V, T, P, S)$

3. $X \rightarrow a$

4. $X \rightarrow \epsilon$

• com $X, Y \in V$, $a \in T$

Algoritmo de equivalência entre GR's e AF's

• Algoritmo de conversão GR → AF

3. Função de transição:

• $\delta = \emptyset$;

• Para cada regra de produção em P da gramática G , e conforme seu tipo:

1. Se $X \rightarrow aY$ então $\delta = \delta \cup \{(X,a) \rightarrow Y\}$;
2. Se $X \rightarrow Y$ então $\delta = \delta \cup \{(X,\epsilon) \rightarrow Y\}$;
3. Se $X \rightarrow a$ então $\delta = \delta \cup \{(X,a) \rightarrow Z\}$;
4. Se $X \rightarrow \epsilon$ então $\delta = \delta \cup \{(X,\epsilon) \rightarrow Z\}$;

$$G = (V, T, P, S)$$

1. $X \rightarrow aY$
2. $X \rightarrow Y$
3. $X \rightarrow a$
4. $X \rightarrow \epsilon$

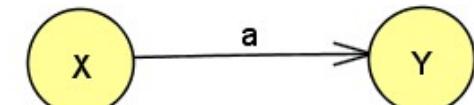
• com $X, Y \in V$, $a \in T$

Algoritmo de equivalência entre GR's e AF's

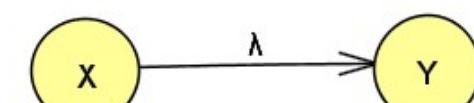
3. Função de transição:

- $\delta = \emptyset$;
- Para cada regra de produção em P da gramática G , e conforme seu tipo:

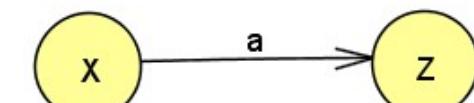
1. Se $X \rightarrow aY$ então $\delta = \delta \cup \{(X,a) \rightarrow Y\}$;



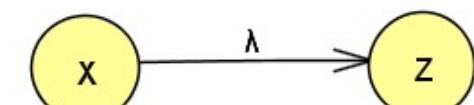
2. Se $X \rightarrow Y$ então $\delta = \delta \cup \{(X,\epsilon) \rightarrow Y\}$;



3. Se $X \rightarrow a$ então $\delta = \delta \cup \{(X,a) \rightarrow Z\}$;



4. Se $X \rightarrow \epsilon$ então $\delta = \delta \cup \{(X,\epsilon) \rightarrow Z\}$;



Exemplo de equivalência entre GR's e AF's

- Seja G uma gramática linear à direita:

$$G = (V, T, P, S)$$

$$V = \{S, K, L\}$$

$$T = \{a, b, c\}$$

$$P = \{S \rightarrow a, S \rightarrow aK, K \rightarrow bK, K \rightarrow L, L \rightarrow cL, L \rightarrow \varepsilon\}$$

$$\delta = \delta \cup \{(S, a) \rightarrow Z\};$$

- O AF correspondente é dado por:

Exemplo de equivalência entre GR's e AF's

- Seja G uma gramática linear à direita:

$$G = (V, T, P, S)$$

$$V = \{S, K, L\}$$

$$\delta = \delta \cup \{(S, a) \rightarrow K\};$$

$$T = \{a, b, c\}$$

$$P = \{S \rightarrow a, S \rightarrow aK, K \rightarrow bK, K \rightarrow L, L \rightarrow cL, L \rightarrow \varepsilon\}$$

- O AF correspondente é dado por:

Exemplo de equivalência entre GR's e AF's

- Seja G uma gramática linear à direita:

$$G = (V, T, P, S)$$

$$V = \{S, K, L\}$$

$$T = \{a, b, c\}$$

$$P = \{S \rightarrow a, S \rightarrow aK, K \rightarrow bK, K \rightarrow L, L \rightarrow cL, L \rightarrow \varepsilon\}$$

$$\delta = \delta \cup \{(K, b) \rightarrow K\};$$

- O AF correspondente é dado por:

Exemplo de equivalência entre GR's e AF's

- Seja G uma gramática linear à direita:

$$G = (V, T, P, S)$$

$$V = \{S, K, L\}$$

$$T = \{a, b, c\}$$

$$P = \{S \rightarrow a, S \rightarrow aK, K \rightarrow bK, K \rightarrow L, L \rightarrow cL, L \rightarrow \varepsilon\}$$

$$\delta = \delta \cup \{(K, \varepsilon) \rightarrow L\};$$

- O AF correspondente é dado por:

Exemplo de equivalência entre GR's e AF's

- Seja G uma gramática linear à direita:

$$G = (V, T, P, S)$$

$$V = \{S, K, L\}$$

$$T = \{a, b, c\}$$

$$P = \{S \rightarrow a, S \rightarrow aK, K \rightarrow bK, K \rightarrow L, L \rightarrow cL, L \rightarrow \varepsilon\}$$

$$\delta = \delta \cup \{(L, c) \rightarrow L\};$$

- O AF correspondente é dado por:

Exemplo de equivalência entre GR's e AF's

- Seja G uma gramática linear à direita:

$$G = (V, T, P, S)$$

$$V = \{S, K, L\}$$

$$T = \{a, b, c\}$$

$$P = \{S \rightarrow a, S \rightarrow aK, K \rightarrow bK, K \rightarrow L, L \rightarrow cL, L \rightarrow \varepsilon\}$$

- O AF correspondente é dado por:

$$\delta = \delta \cup \{(L, \varepsilon) \rightarrow Z\};$$

Exemplo de equivalência entre GR's e AF's

- Seja G uma gramática linear à direita:

$$G = (V, T, P, S)$$

$$V = \{S, K, L\}$$

$$T = \{a, b, c\}$$

$$P = \{S \rightarrow a, S \rightarrow aK, K \rightarrow bK, K \rightarrow L, L \rightarrow cL, L \rightarrow \varepsilon\}$$

$$\delta = \delta \cup \{(S, a) \rightarrow Z\};$$

$$\delta = \delta \cup \{(S, a) \rightarrow K\};$$

$$\delta = \delta \cup \{(K, b) \rightarrow K\};$$

$$\delta = \delta \cup \{(K, \varepsilon) \rightarrow L\};$$

$$\delta = \delta \cup \{(L, c) \rightarrow L\};$$

$$\delta = \delta \cup \{(L, \varepsilon) \rightarrow Z\};$$

- O AF correspondente é dado por:

Exemplo de equivalência entre GR's e AF's

- Seja G uma gramática linear à direita:

$$G = (V, T, P, S)$$

$$V = \{S, K, L\}$$

$$T = \{a, b, c\}$$

$$P = \{S \rightarrow a, S \rightarrow aK, K \rightarrow bK, K \rightarrow L, L \rightarrow cL, L \rightarrow \varepsilon\}$$

$$\delta = \delta \cup \{(S, a) \rightarrow Z\};$$

$$\delta = \delta \cup \{(S, a) \rightarrow K\};$$

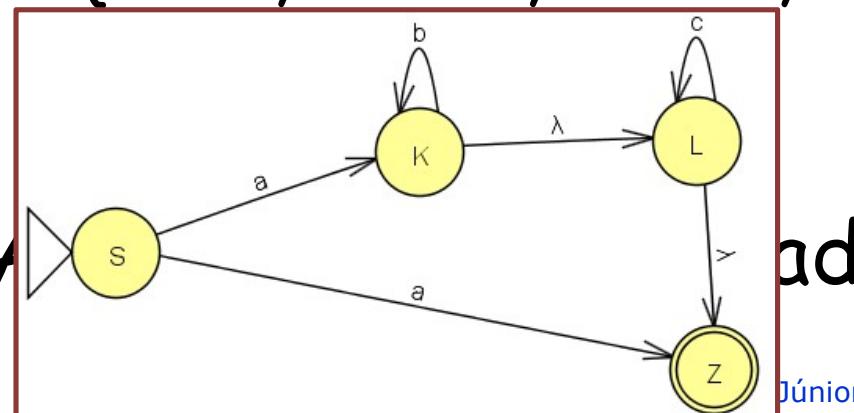
$$\delta = \delta \cup \{(K, b) \rightarrow K\};$$

$$\delta = \delta \cup \{(K, \varepsilon) \rightarrow L\};$$

$$\delta = \delta \cup \{(L, c) \rightarrow L\};$$

$$\delta = \delta \cup \{(L, \varepsilon) \rightarrow Z\};$$

- O AFS é dado por:



Júnior

- Seja G uma gramática linear à direita:

$$G = (V, T, P, S)$$

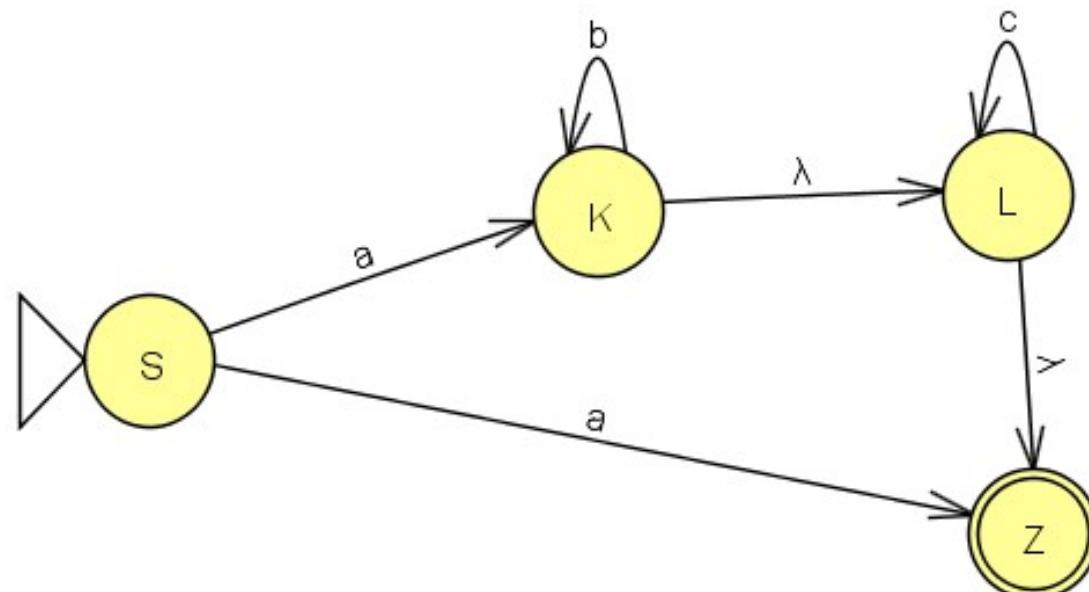
$$V = \{S, K, L\}$$

$$T = \{a, b, c\}$$

$$P = \{S \rightarrow a, S \rightarrow aK, K \rightarrow bK, K \rightarrow L, L \rightarrow cL, L \rightarrow \varepsilon\}$$

Qual a $L(G)$ e a $L(M)$?

- O AF corresponde
é dado por:



- Seja G uma gramática linear à direita:

$$G = (V, T, P, S)$$

$$V = \{S, K, L\}$$

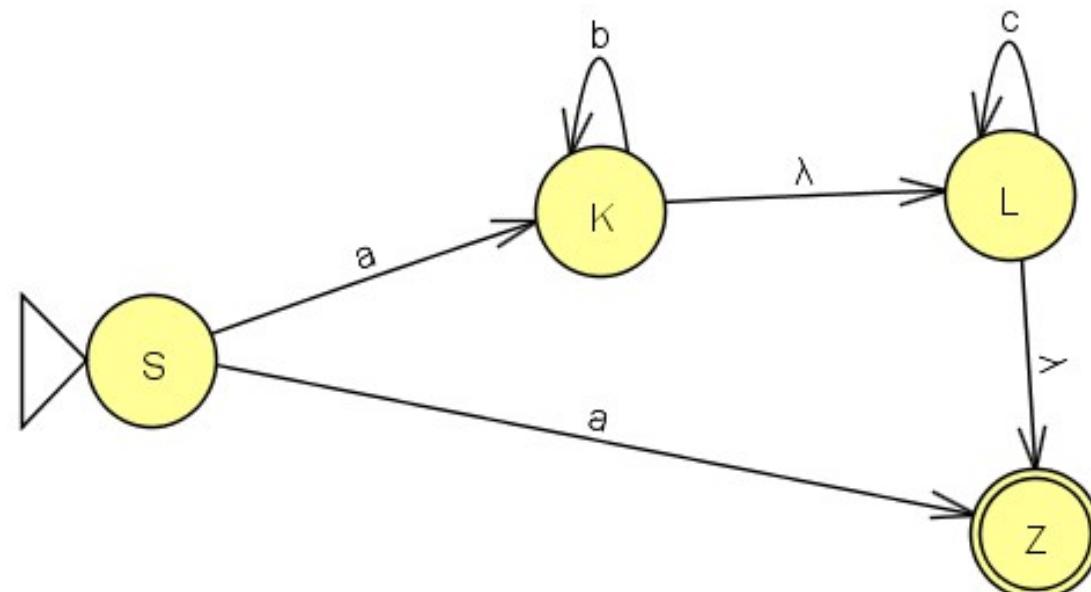
$$T = \{a, b, c\}$$

$$P = \{S \rightarrow a, S \rightarrow aK, K \rightarrow bK, K \rightarrow L, L \rightarrow cL, L \rightarrow \varepsilon\}$$

Qual a $L(G)$ e a $L(M)$?

R.: ab^*c^*

- O AF corresponde
é dado por:



- Seja G uma gramática linear à direita:

$$G = (V, T, P, S)$$

$$V = \{S, K, L\}$$

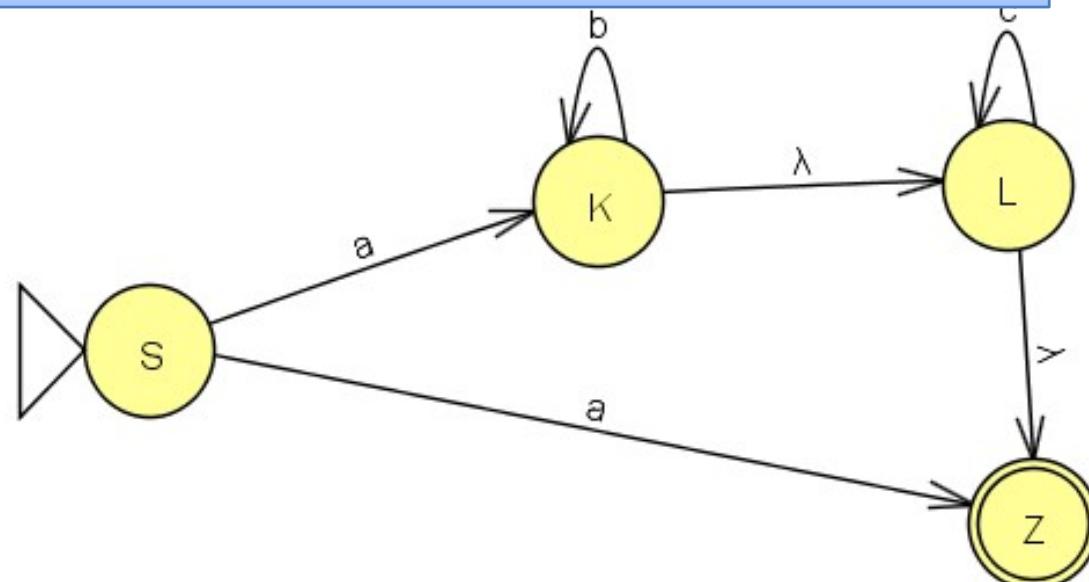
~~T = {a, b, c}~~

Qual a $L(G)$ e a $L(M)$?

R.: ab^*c^*

Seja G uma gramática linear à direita. Então é possível definir um autômato finito M de tal modo que $L(G) = L(M)$.

- O AF corresponde
é dado por:



- Seja G uma gramática linear à direita:

$$G = (V, T, P, S)$$

$$V = \{S, K, L\}$$

$$T = \{a, b, c\}$$

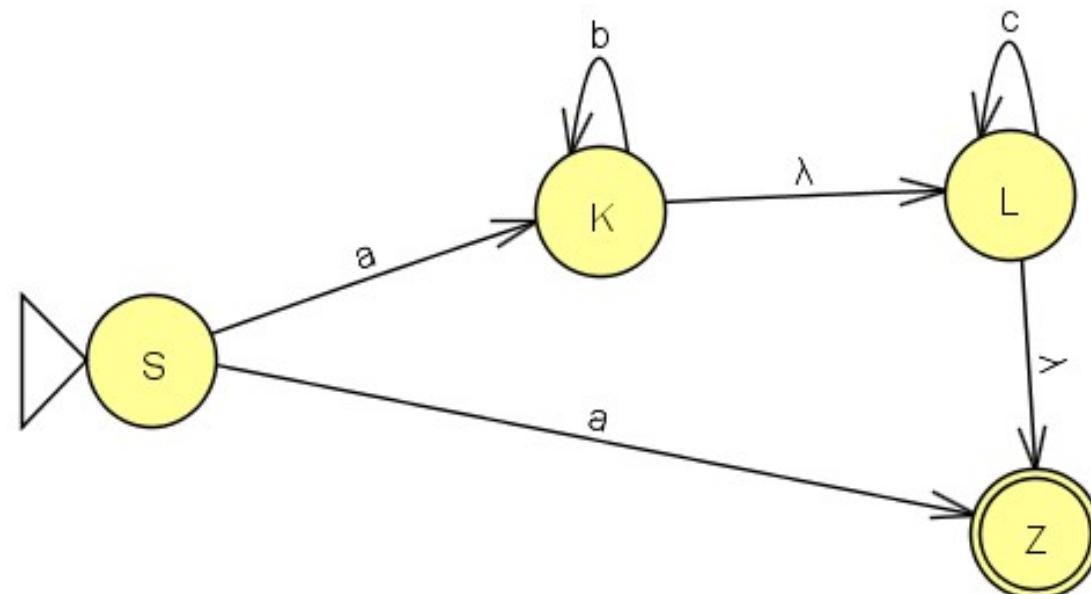
$$P = \{S \rightarrow a, S \rightarrow aK, K \rightarrow bK, K \rightarrow L, L \rightarrow cL, L \rightarrow \varepsilon\}$$

Qual a $L(G)$ e a $L(M)$?

R.: ab^*c^*

Exemplo de sentença aceita por L : $abbcc$

- O AF corresponde é dado por:



Conversão entre AF's e GR's

Equivalência entre AF's e GR's

Seja M um autômato finito qualquer. Então é possível definir uma gramática linear à direita G , de tal modo que $L(M) = L(G)$.

- Dado $M = (\{Q\}, \Sigma, \delta, q_0, \{F\})$ um AFND- ε é possível construir uma gramática linear à direita $G = (V, T, P, S)$ a partir de M .

Algoritmo de equivalência entre AF's e GR's

- **Algoritmo de conversão AF → GR**
- **Entrada:** um autômato finito M ;
- **Saída:** uma gramática linear à direita G tal que $L(G) = L(M)$;
- **Método:**

1. Definição do conjunto de símbolos não-terminais:

- Os símbolos não-terminais de G correspondem aos estados de M . A raiz da gramática é q_0 (estado inicial).

2. Alfabeto de entrada:

- O alfabeto Σ de G é o próprio alfabeto de entrada Σ de M .

Algoritmo de equivalência entre AF's e GR's

• Algoritmo de conversão AF → GR

3. Produções:

- $P \leftarrow \emptyset;$

- Para cada elemento de δ do AFND- ε M, e conforme o tipo das transições de M:

- 1 Se $\delta(X, a) = Y$, então $P \{X \rightarrow aY\};$

- 2 Se $\delta(X, \varepsilon) = Y$, então $P \{X \rightarrow Y\}.$

- Para cada elemento de Q do AFND- ε M:

- 1 Se $X \in F$, então $P \{X \rightarrow \varepsilon\}.$

Se X é um estado final

Equivalência entre AF's e GR's

- Exemplo: Dado o AFND- ε M definido e representado abaixo

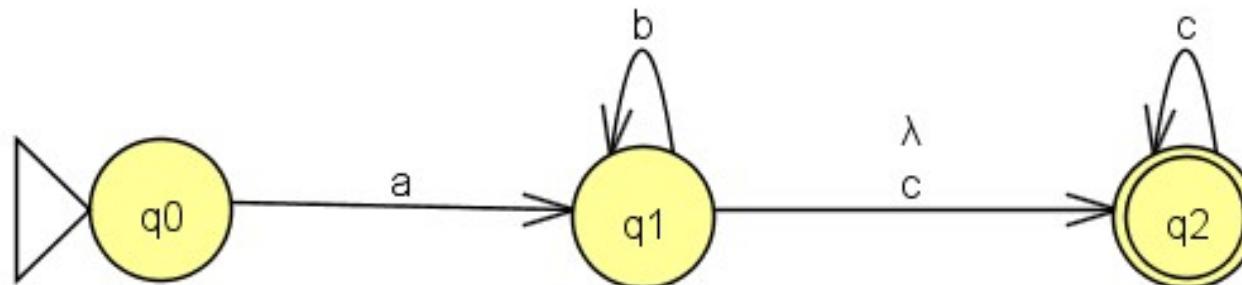
$$M = (\{Q\}, \Sigma, \delta, q_0, \{F\})$$

$$Q = \{q_0, q_1, q_2\}$$

$$\Sigma = \{a, b, c\}$$

$$\delta = \{(q_0, a) = q_1, (q_1, b) = q_1, (q_1, c) = q_2, (q_1, \varepsilon) = q_2, (q_2, c) = q_2\}$$

$$F = \{q_2\}$$



Equivalência entre AF's e GR's

- Aplicando-se o algoritmo de conversão ao AFND- ϵ M , obtém-se a gramática linear à direita G , cujo conjunto de produções P corresponde à segunda coluna da mesma. Note que $L(M) = L(G) = ab^*c^*$.

$$G = (V, \Sigma, P, q_0)$$

$$V = \{q_0, q_1, q_2\}$$

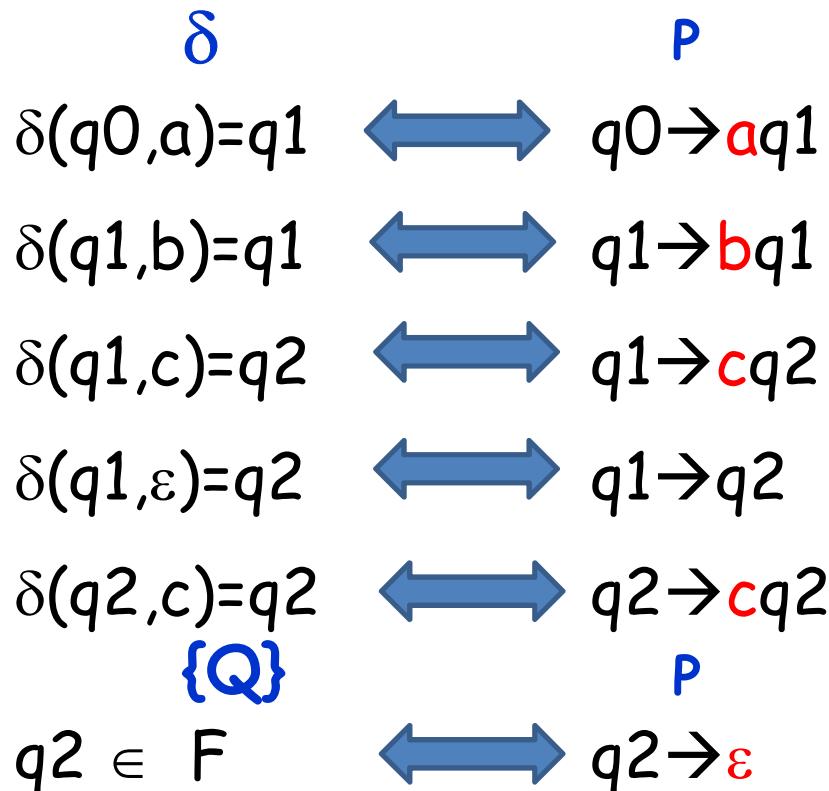
$$\Sigma = \{a, b, c\}$$

Linguagens Formais e Teoria da Computação

$$G = (V, \Sigma, P, q_0)$$

$$V = \{q_0, q_1, q_2\}$$

$$\Sigma = \{a, b, c\}$$



• Para cada elemento de δ do AFND- ε M, é feita a mutação correspondente conforme o tipo das transições de M:

1. Se $\delta(X, a) = Y$, então $P \leftarrow \{X \rightarrow aY\}$;

2. Se $\delta(X, \varepsilon) = Y$, então $P \leftarrow \{X \rightarrow Y\}$.

• Para cada elemento de Q do AFND- ε M:

1. Se $X \in F$, então $P \leftarrow \{X \rightarrow \varepsilon\}$.

$$G = (\{A, B, C\}, \{a, b, c\}, P, A)$$

P:{

$A \rightarrow aB$

$B \rightarrow bB$

$B \rightarrow cC$

$B \rightarrow C$

$C \rightarrow cC$

$C \rightarrow \varepsilon$

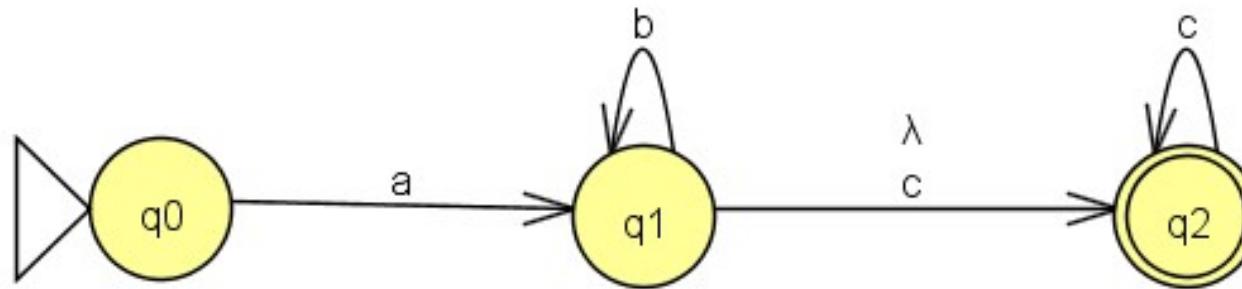
}

Renomeando os estados

$q_0 \rightarrow A$

$q_1 \rightarrow B$

$q_2 \rightarrow C$



$$G = (\{A, B, C\}, \{a, b, c\}, P, A)$$

P:{

$A \rightarrow aB$

$B \rightarrow bB$

$B \rightarrow cC$

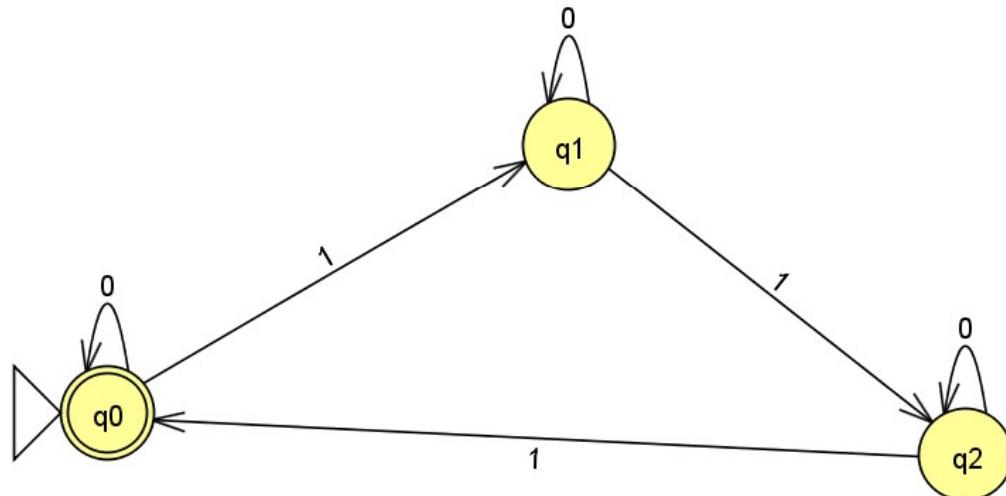
$B \rightarrow C$

$C \rightarrow cC$

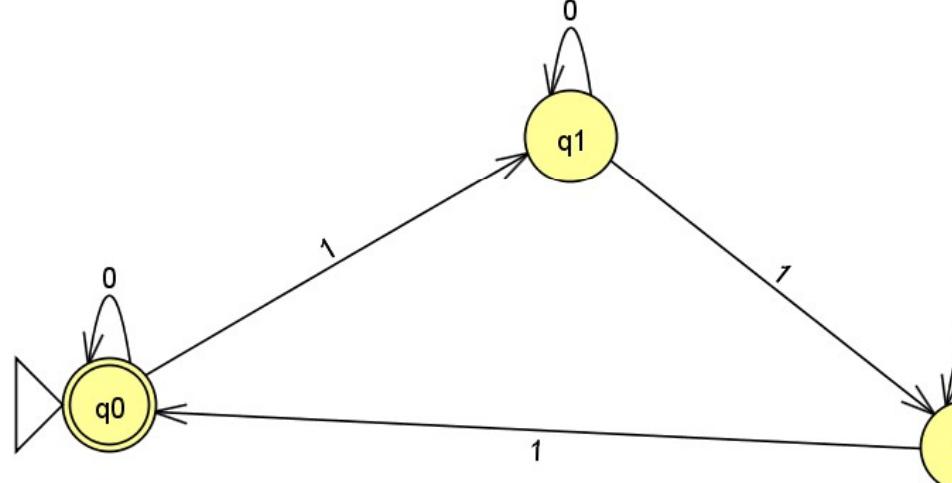
$C \rightarrow \epsilon$

}

Dado o AFD que reconhece a $L=\{0^n1^m \mid n \geq 0 \text{ e } m \text{ é múltiplo de } 3\}$, encontre a GR correspondente.



Dado o AFD que reconhece a $L=\{0^n1^m \mid n \geq 0 \text{ e } m \text{ é múltiplo de } 3\}$, encontre a GR correspondente.



δ

$$\delta(q_0, 0) = q_0$$

$$\delta(q_0, 1) = q_1$$

$$\delta(q_1, 0) = q_1$$

$$\delta(q_1, 1) = q_2$$

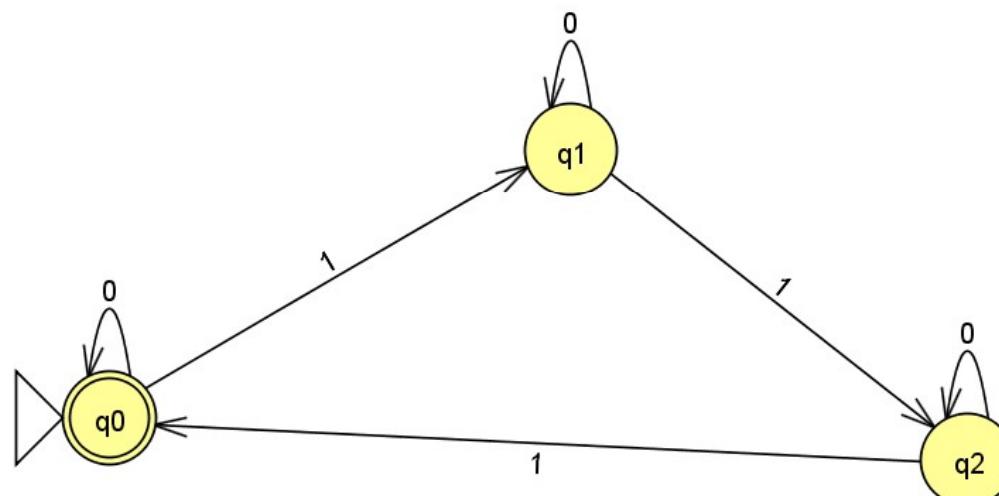
$$\delta(q_2, 0) = q_2$$

$$\delta(q_2, 1) = q_0$$

{Q}

$q_0 \in F$

Dado o AFD que reconhece a $L=\{0^n1^m \mid n \geq 0 \text{ e } m \text{ é múltiplo de } 3\}$, encontre a GR correspondente.

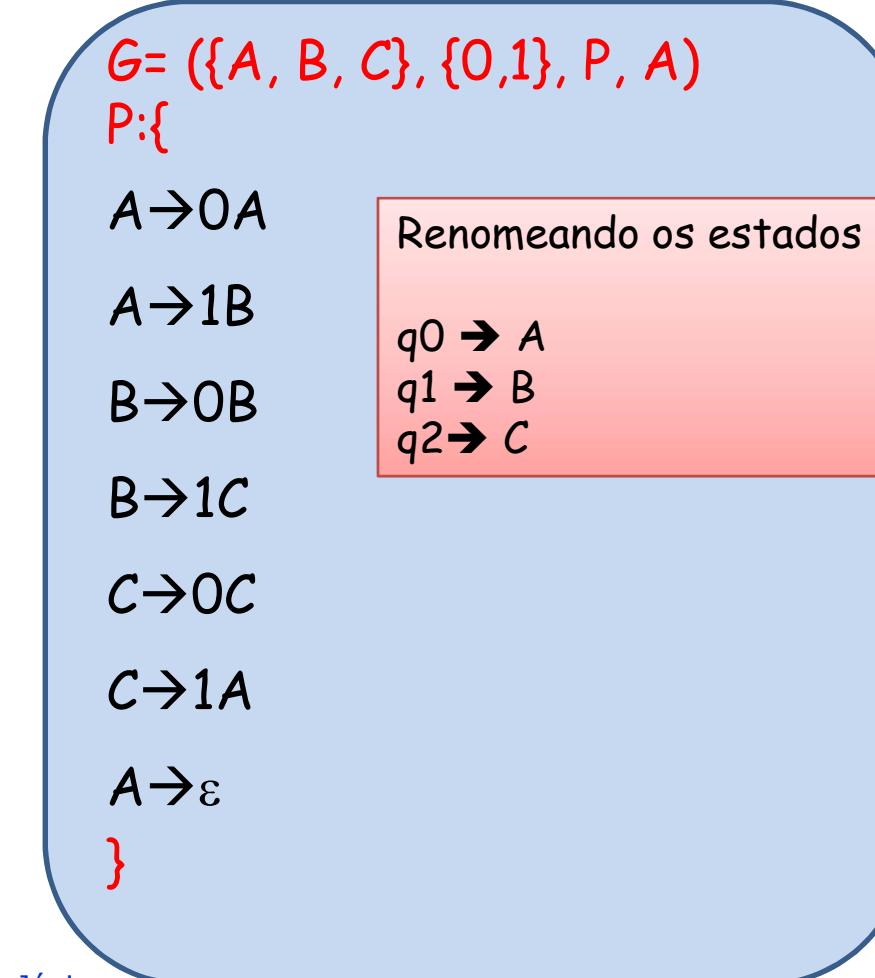


δ	P
$\delta(q0, 0) = q0$	$q0 \xrightarrow{} 0q0$
$\delta(q0, 1) = q1$	$q0 \xrightarrow{} 1q1$
$\delta(q1, 0) = q1$	$q1 \xrightarrow{} 0q1$
$\delta(q1, 1) = q2$	$q1 \xrightarrow{} 1q2$
$\delta(q2, 0) = q2$	$q2 \xrightarrow{} 0q2$
$\delta(q2, 1) = q0$	$q2 \xrightarrow{} 1q0$

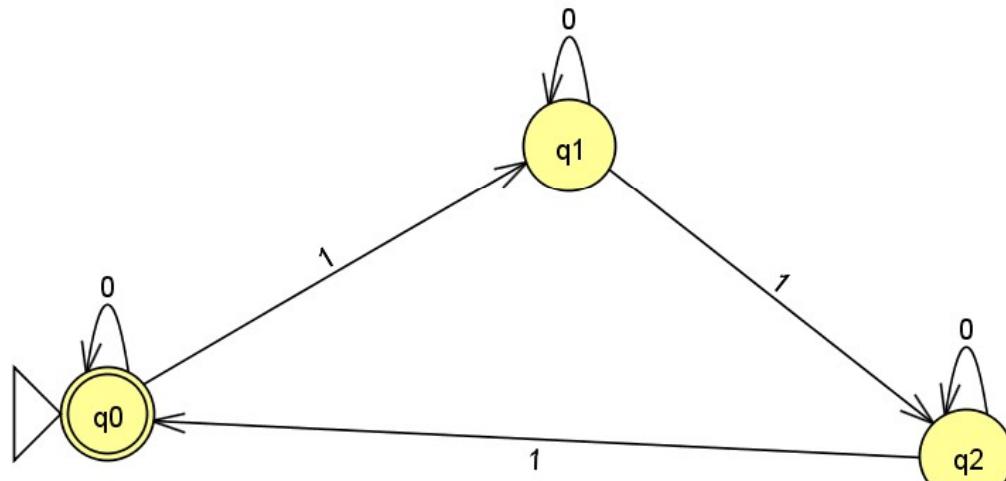
$\{Q\}$	P
$q0 \in F$	$q0 \xrightarrow{} \epsilon$

Dado o AFD que reconhece a $L=\{0^n1^m \mid n \geq 0 \text{ e } m \text{ é múltiplo de } 3\}$, encontre a GR correspondente.

δ	P
$\delta(q_0, 0) = q_0$	$q_0 \xrightarrow{} 0q_0$
$\delta(q_0, 1) = q_1$	$q_0 \xrightarrow{} 1q_1$
$\delta(q_1, 0) = q_1$	$q_1 \xrightarrow{} 0q_1$
$\delta(q_1, 1) = q_2$	$q_1 \xrightarrow{} 1q_2$
$\delta(q_2, 0) = q_2$	$q_2 \xrightarrow{} 0q_2$
$\delta(q_2, 1) = q_0$	$q_2 \xrightarrow{} 1q_0$
{Q}	
$q_0 \in F$	$q_0 \xrightarrow{} \epsilon$



Dado o AFD que reconhece a $L=\{0^n1^m \mid n \geq 0 \text{ e } m \text{ é múltiplo de } 3\}$, encontre a GR correspondente.



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

P:{

$$A \rightarrow 0A$$

$$A \rightarrow 1B$$

$$B \rightarrow 0B$$

$$B \rightarrow 1C$$

$$C \rightarrow 0C$$

$$C \rightarrow 1A$$

$$A \rightarrow \epsilon$$

}

Renomeando os estados

$$q_0 \rightarrow A$$

$$q_1 \rightarrow B$$

$$q_2 \rightarrow C$$

Exercícios

- Escolha 5 enunciados (dos 39 propostos) dos exercícios da Aula 3 e aplique os algoritmos de conversão

LFTC - Aula 07

Minimização de AFD Autômatos Finitos com saídas

Celso Olivete Júnior

celso.olivete@unesp.br

Na aula de hoje

- Minimização de autômatos finitos determinísticos
- Autômatos Finitos com saídas: Máquina de Moore e Máquina de Mealy

Minimização de autômatos finitos determinísticos

Minimização de AFD's

- O objetivo da minimização é gerar um Autômato Finito Determinístico equivalente com o menor número de estados possíveis.
- Em algumas aplicações especiais, a minimização do número de estados não implica necessariamente no menor custo de implementação

Minimização de AFD's

- Basicamente, o algoritmo de minimização unifica os estados equivalentes.
- Dois estados q e p são ditos equivalentes se, e somente se, para qualquer palavra w pertencente ao Σ^* , $\delta(q,w)$ e $\delta(p,w)$ resultam simultaneamente em estados finais, ou não-finais.
- O processamento de uma entrada qualquer a partir de estados equivalentes gera, em qualquer caso, o mesmo resultado aceita/rejeita.



Algoritmo de minimização de AFD's

- Pré-requisitos:

- Um AF a ser minimizado deve **satisfazer**:

- Deve ser determinístico;
 - Não pode ter estados inacessíveis (não-atingíveis a partir do estado inicial);
 - A função programa deve ser total (a partir de qualquer estado são previstas transições para todos os símbolos do alfabeto).

- Obs:

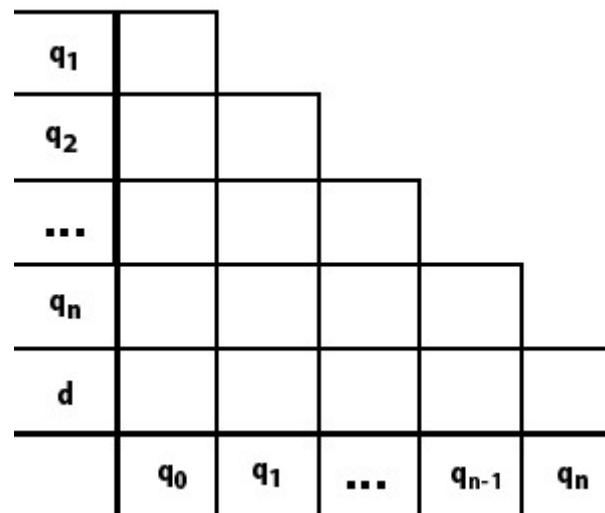
- Se os pré-requisitos não forem atendidos:

- Caso o AF não seja AFD, deve-se transformá-lo
 - eliminar os estados inacessíveis
 - tornar a função programa total (é suficiente introduzir um novo **estado não final D** e incluir as transições não previstas, tendo **D** como estado destino. Por fim, incluir um ciclo em **D** para todos os símbolos do alfabeto.)

Algoritmo de minimização de AFD's

- Passos do algoritmo

1. Construir uma tabela relacionando os estados distintos onde cada par de estados ocorre apenas uma vez



Algoritmo de minimização de AFD's

2. Marcação dos estados trivialmente não-equivalentes

- **Marcar todos os pares do tipo { estado final, estado não-final }, pois, obviamente, estados finais não são equivalentes a não-finais;**
- **Obs: verificar os estados de aceitação (finais) e marcar todos os pares que apresentam um destes estados;**

Algoritmo de minimização de AFD's

3. Marcação dos estados não-equivalentes.

Para cada par $\{qu, qv\}$ não-marcado e para cada símbolo $a \in \Sigma$, suponha que $\delta(qu, a) = pu$ e $\delta(qv, a) = pv$ e:

- 3.1 se $pu = pv$, então qu é equivalente a qv para o símbolo a e não deve ser marcado;
- 3.2 se $pu \neq pv$ e o par $\{ pu, pv \}$ não está marcado, então $\{ qu, qv \}$ é incluído em uma lista a partir de $\{ pu, pv \}$ para posterior análise;
- 3.3 se $pu \neq pv$ e o par $\{ pu, pv \}$ está marcado, então:
 - ❖ $\{qu, qv\}$ não é equivalente e deve ser marcado;
 - ❖ se $\{qu, qv\}$ encabeça uma lista de pares, então marcar todos os pares da lista (e, recursivamente, se algum par da lista encabeça outra lista);

Algoritmo de minimização de AFD's

4. Unificação dos estados equivalentes.

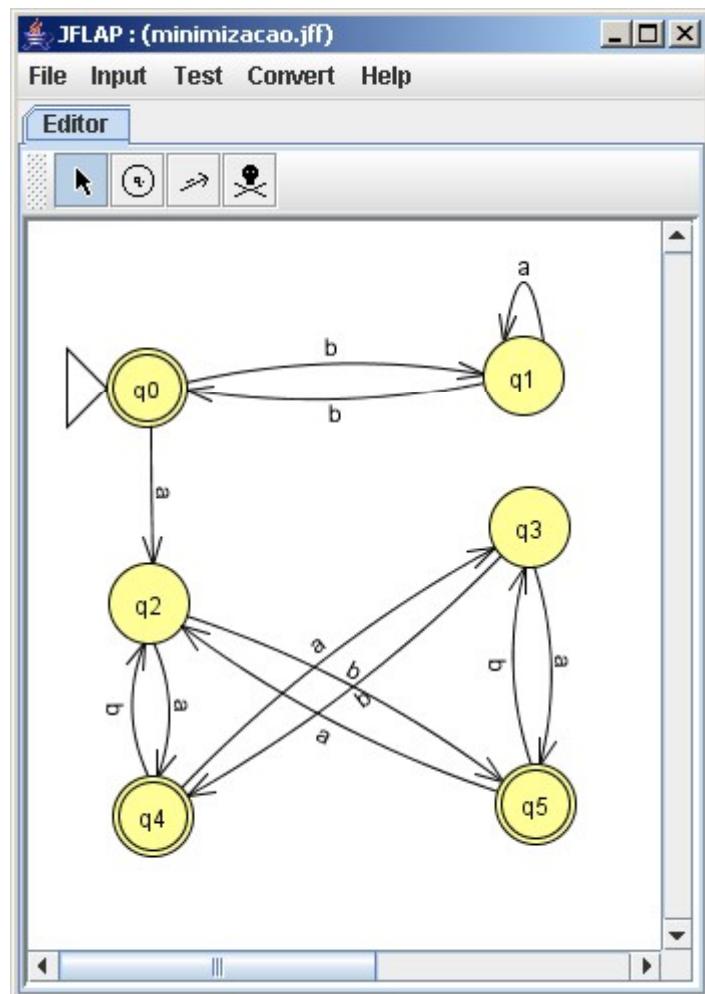
- Os estados dos pares não-marcados são equivalentes e podem ser unificados como segue:
 - pares de estados não-finais equivalentes podem ser unificados como um único estado não-final;
 - pares de estados finais equivalentes podem ser unificados como um único estado final;
 - se algum dos estados equivalentes é inicial, então o correspondente estado unificado é inicial;

Algoritmo de minimização de AFD's

5. Exclusão dos estados inúteis.

- Por fim, os estados chamados inúteis devem ser excluídos.
 - Um estado q é inútil se é não-final e a partir de q não é possível atingir um estado final.
 - Deve-se reparar que o estado **D** (se incluído) sempre é inútil.

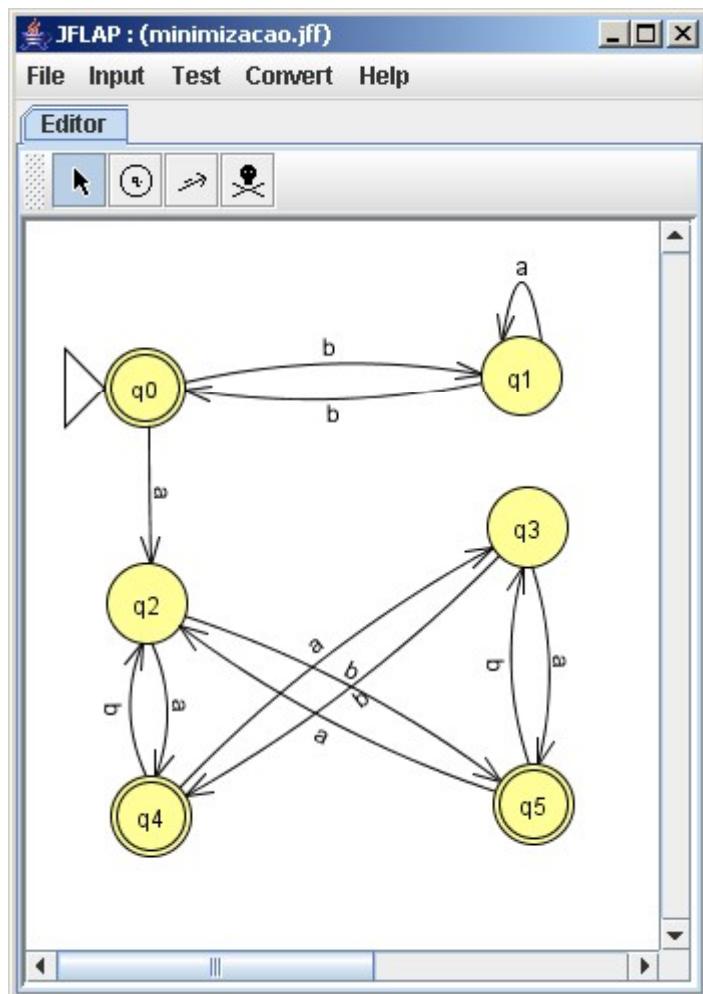
Algoritmo de minimização de AFD's - exemplo



•pré-requisitos:

- Deve ser determinístico;
- Não pode ter estados inacessíveis (não-atingíveis a partir do estado inicial);
- A função programa deve ser total (a partir de qualquer estado são previstas transições para todos os símbolos do alfabeto).

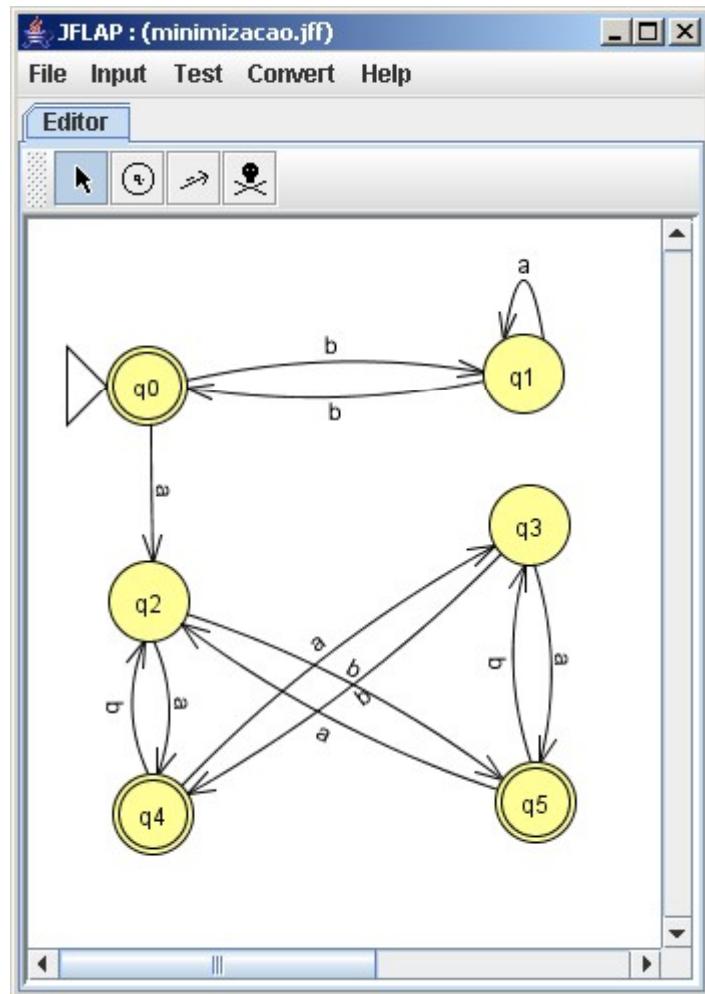
Algoritmo de minimização de AFD's - exemplo



•pré-requisitos:

- Deve ser determinístico; **OK**
- Não pode ter estados inacessíveis (não-atingíveis a partir do estado inicial); **OK**
- A função programa deve ser total (a partir de qualquer estado são previstas transições para todos os símbolos do alfabeto). **OK**

Algoritmo de minimização de AFD's - exemplo



Construção da tabela (Passo 1)
e marcação dos estados do tipo
{estado final, estado não-final}
(Passo 2)

q_1	X	$\rightarrow (q_0, q_1)$			
q_2	X	final, não-final			
q_3	X	$\rightarrow (q_0, q_3)$			
q_4		X	X	X	
q_5	X	X	X		
	q_0	q_1	q_2	q_3	q_4

Obs: como q_0 , q_4 e q_5 são finais, inserimos X em cada par que os envolve

Algoritmo de minimização de AFD's

q_1	\times				
q_2	\times				
q_3	\times				
q_4	●	\times	\times	\times	
q_5		\times	\times	\times	
	q_0	q_1	q_2	q_3	q_4

Análise dos pares de estados não-marcados

c-1) Análise do par { q_0, q_4 }:

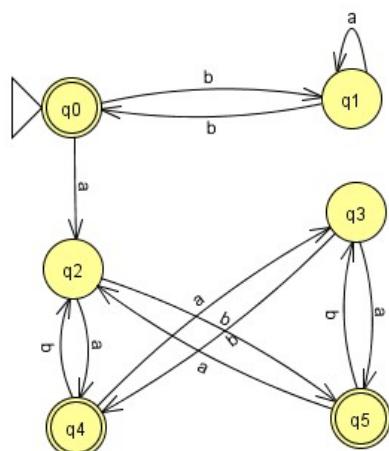
$$(q_0, a) = q_2$$

$$(q_0, b) = q_1$$

$$(q_4, a) = q_3$$

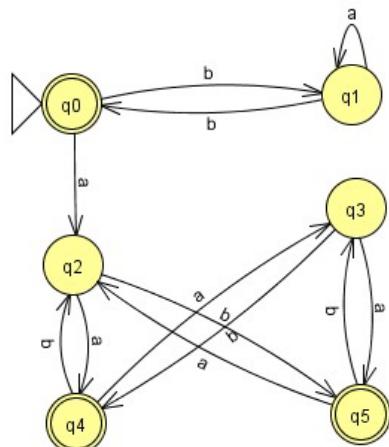
$$(q_4, b) = q_2$$

- Como { q_1, q_2 } e { q_2, q_3 } são não-marcados, então { q_0, q_4 } é incluído nas listas encabeçadas por { q_1, q_2 } e { q_2, q_3 };



Algoritmo de minimização de AFD's

q_1	\times				
q_2	\times				
q_3	\times				
q_4	●	\times	\times	\times	
q_5		\times	\times	\times	
	q_0	q_1	q_2	q_3	q_4



Análise dos pares de estados não-marcados

c-1) Análise do par { q_0, q_4 }:

$$(q_0, a) = q_2$$

$$(q_0, b) = q_1$$

$$(q_4, a) = q_3$$

$$(q_4, b) = q_2$$

- Como { q_1, q_2 } e { q_2, q_3 } são não-marcados, então { q_0, q_4 } é incluído nas listas encabeçadas por { q_1, q_2 } e { q_2, q_3 };

Algoritmo de minimização de AFD's

q_1	\times					(q_0, q_4)
q_2	\times					
q_3	\times					(q_0, q_4)
q_4	○	\times	\times	\times		$(q_0, a) = q_2$
q_5	●	\times	\times	\times		$(q_0, b) = q_1$
	q_0	q_1	q_2	q_3	q_4	$(q_5, a) = q_2$

Análise dos pares de estados não-marcados

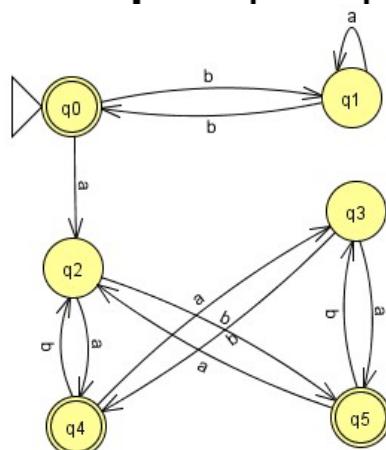
c-2) Análise do par { q_0, q_5 }:

$$(q_0, a) = q_2$$

$$(q_0, b) = q_1$$

$$(q_5, a) = q_2$$

$$(q_5, b) = q_3$$



- Como { q_1, q_3 } é não-marcado (e como { q_2, q_2 } é trivialmente equivalente), então { q_0, q_5 } é incluído na lista encabeçada por { q_1, q_3 };

Algoritmo de minimização de AFD's

q_1	\times						
q_2	\times						
q_3	\times						
q_4	\circ	\times	\times	\times			
q_5	\bullet	\times	\times	\times			
	q_0	q_1	q_2	q_3	q_4		

Análise dos pares de estados não-marcados

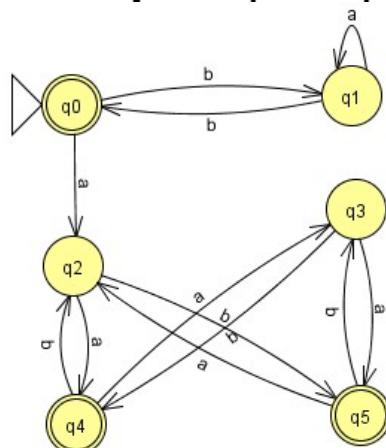
c-2) Análise do par { q_0, q_5 }:

$$(q_0, a) = q_2$$

$$(q_0, b) = q_1$$

$$(q_5, a) = q_2$$

$$(q_5, b) = q_3$$



- Como { q_1, q_3 } é não-marcado (e como { q_2, q_2 } é trivialmente equivalente), então { q_0, q_5 } é incluído na lista encabeçada por { q_1, q_3 };

Algoritmo de minimização de AFD's

q_1	\times				(q_0, q_4)
q_2	\times	●	(q_0, q_5)		
q_3	\times			(q_0, q_4)	
q_4	○	\times	\times	\times	$(q_1, a) = q_1$
q_5	○	\times	\times	\times	$(q_1, b) = q_0$
	q_0	q_1	q_2	q_3	$(q_2, a) = q_4$
					$(q_2, b) = q_5$

Análise dos pares de estados não-marcados

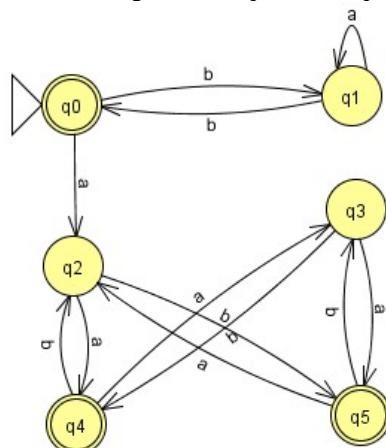
c-3) Análise do par { q_1, q_2 }:

$$(q_1, a) = q_1$$

$$(q_1, b) = q_0$$

$$(q_2, a) = q_4$$

$$(q_2, b) = q_5$$



- Como { q_1, q_4 } é marcado, então { q_1, q_2 } também é marcado.
- Como { q_1, q_2 } encabeça uma lista, o par { q_0, q_4 } também é marcado;

Algoritmo de minimização de AFD's

q_1	x				(q_0, q_4)
q_2	x		(q_0, q_5)		
q_3	x			(q_0, q_4)	
q_4	(\otimes)	x	x	x	
q_5	(\odot)	x	x	x	
	q_0	q_1	q_2	q_3	q_4

Análise dos pares de estados não-marcados

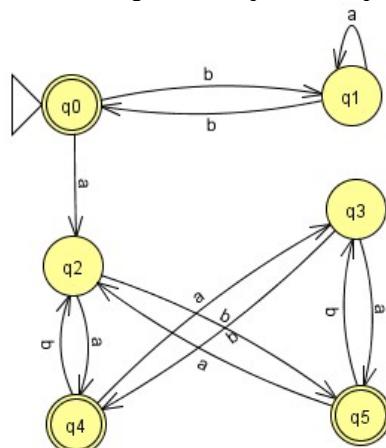
c-3) Análise do par { q_1, q_2 }:

$$(q_1, a) = q_1$$

$$(q_1, b) = q_0$$

$$(q_2, a) = q_4$$

$$(q_2, b) = q_5$$



- Como { q_1, q_4 } é marcado, então { q_1, q_2 } também é marcado.
- Como { q_1, q_2 } encabeça uma lista, o par { q_0, q_4 } também é marcado;

Algoritmo de minimização de AFD's

q_1	\times				
q_2	\times	\otimes			
q_3	\times		\bullet		
q_4	\otimes	\times	\times	\times	
q_5	\circ	\times	\times	\times	
	q_0	q_1	q_2	q_3	q_4

Análise dos pares de estados não-marcados

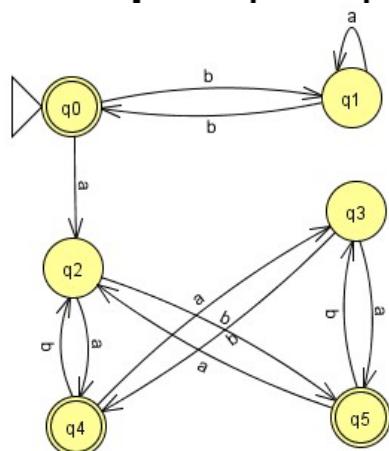
c-4) Análise do par { q_1, q_3 }:

$$(q_1, a) = q_1$$

$$(q_1, b) = q_0$$

$$(q_3, a) = q_5$$

$$(q_3, b) = q_4$$

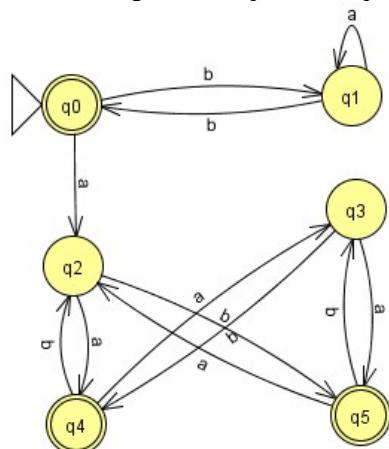


Como { q_1, q_5 } bem como { q_0, q_4 } são marcados, então { q_1, q_3 } também é marcado.

Como { q_1, q_3 } encabeça uma lista, o par { q_0, q_5 } também é marcado;

Algoritmo de minimização de AFD's

q_1	\times				
q_2	\times	\otimes			(q_0, q_5)
q_3	\times	\otimes			(q_0, q_4)
q_4	\otimes	\times	\times	\times	
q_5	\otimes	\times	\times	\times	
	q_0	q_1	q_2	q_3	q_4



Análise dos pares de estados não-marcados

c-4) Análise do par { q_1, q_3 }:

$$(q_1, a) = q_1$$

$$(q_1, b) = q_0$$

$$(q_3, a) = q_5$$

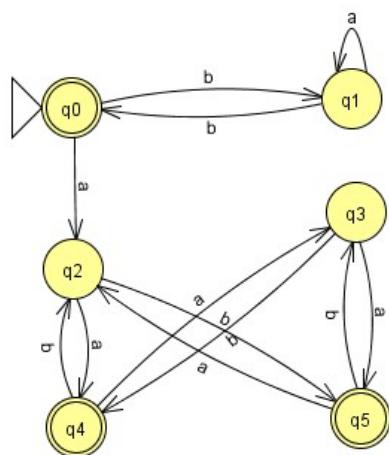
$$(q_3, b) = q_4$$

Como { q_1, q_5 } bem como { q_0, q_4 } são marcados, então { q_1, q_3 } também é marcado.

Como { q_1, q_3 } encabeça uma lista, o par { q_0, q_5 } também é marcado;

Algoritmo de minimização de AFD's

q_1	\times			
q_2	\times	\otimes		
q_3	\times	\otimes	\bullet	(q_0, q_4)
q_4	\otimes	\times	\times	(q_2, q_3)
q_5	\otimes	\times	\times	
	q_0	q_1	q_2	q_3
				q_4



Análise dos pares de estados não-marcados

c-5) Análise do par { q_2, q_3 }:

$$(q_2, a) = q_4$$

$$(q_2, b) = q_5$$

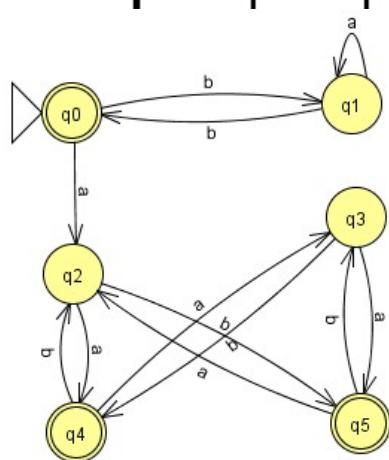
$$(q_3, a) = q_5$$

$$(q_3, b) = q_4$$

Como { q_4, q_5 } é não-marcado, então { q_2, q_3 } é incluído na lista encabeçada por { q_4, q_5 };

Algoritmo de minimização de AFD's

q_1	\times				
q_2	\times	\otimes			
q_3	\times	\otimes	\circ		
q_4	\otimes	\times	\times	\times	
q_5	\otimes	\times	\times	\times	\bullet
	q_0	q_1	q_2	q_3	q_4



Análise dos pares de estados não-marcados

c-6) Análise do par { q_4, q_5 }:

$$(q_4, a) = q_3$$

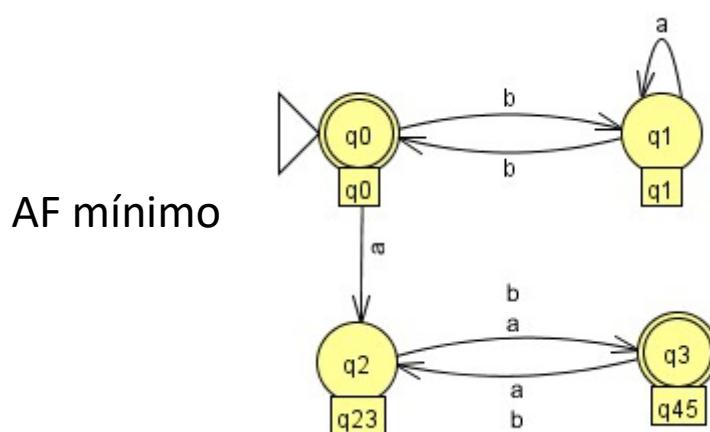
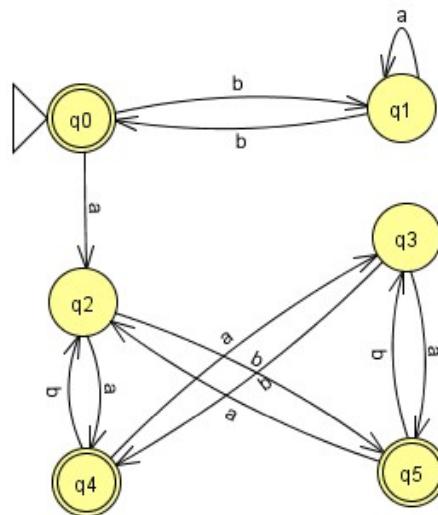
$$(q_4, b) = q_2$$

$$(q_5, a) = q_2$$

$$(q_5, b) = q_3$$

Como { q_2, q_3 } é não-marcado, então { q_4, q_5 } é incluído na lista encabeçada por { q_2, q_3 };

Algoritmo de minimização de AFD's



Celso Olivete Júnior

Unificação

d) Como os pares $\{q_2, q_3\}$ e $\{q_4, q_5\}$ são não-marcados, as seguintes unificações podem ser feitas:

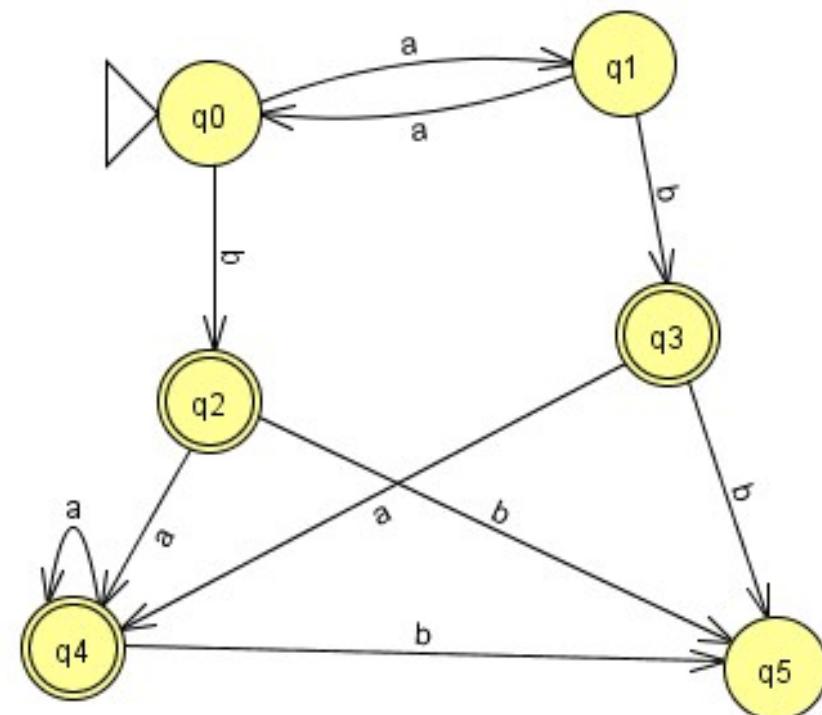
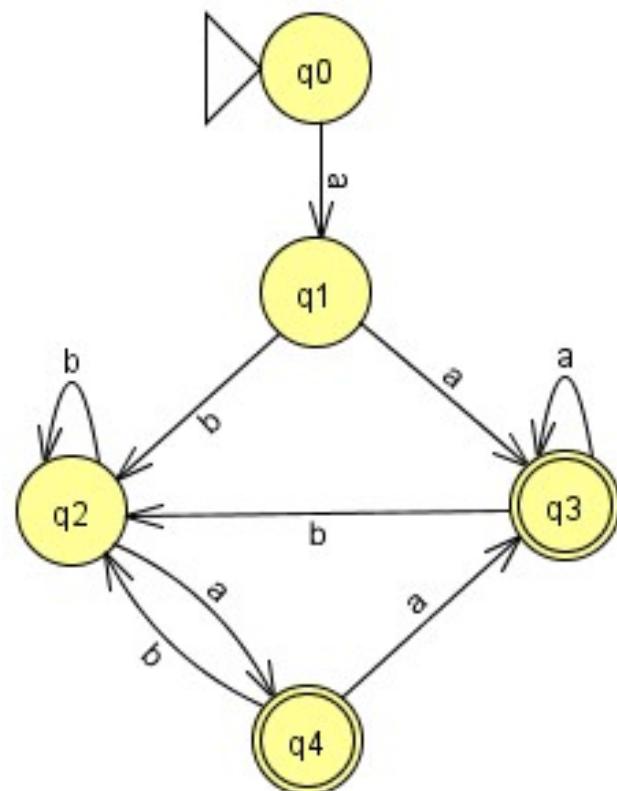
q_{23} representa a unificação dos estados não-finais q_2 e q_3 ;

q_{45} representa a unificação dos estados finais q_4 e q_5 .

q_1	\times				
q_2	\times	\otimes			
q_3	\times	\otimes			
q_4	\otimes	\times	\times	\times	
q_5	\otimes	\times	\times	\times	
	q_0	q_1	q_2	q_3	q_4

Exercícios

- Minimize os seguintes AFD's utilizando o algoritmo apresentado anteriormente



Exercícios

2. Considere o AF não determinista $M = (\{1,2,3,4,5\}, \{x,y\}, d, 1, \{5\})$, com a função de transição dada na tabela

d	x	y
→ 1	{1}	{1,2}
2	Ø	{3}
3	{3}	{3,4}
4	Ø	{5}
* 5	{5}	{5}

Construa o AFD correspondente

Construa o AFD *mínimo* correspondente, utilizando o algoritmo apresentado.

Exercícios

3. Minimize o AFD:

$$(\rightarrow q_0, a) = (q_1)$$

$$(q_0, b) = (q_0)$$

$$(q_1, a) = (q_2)$$

$$(q_1, b) = (q_0)$$

$$(q_2, a) = (q_3)$$

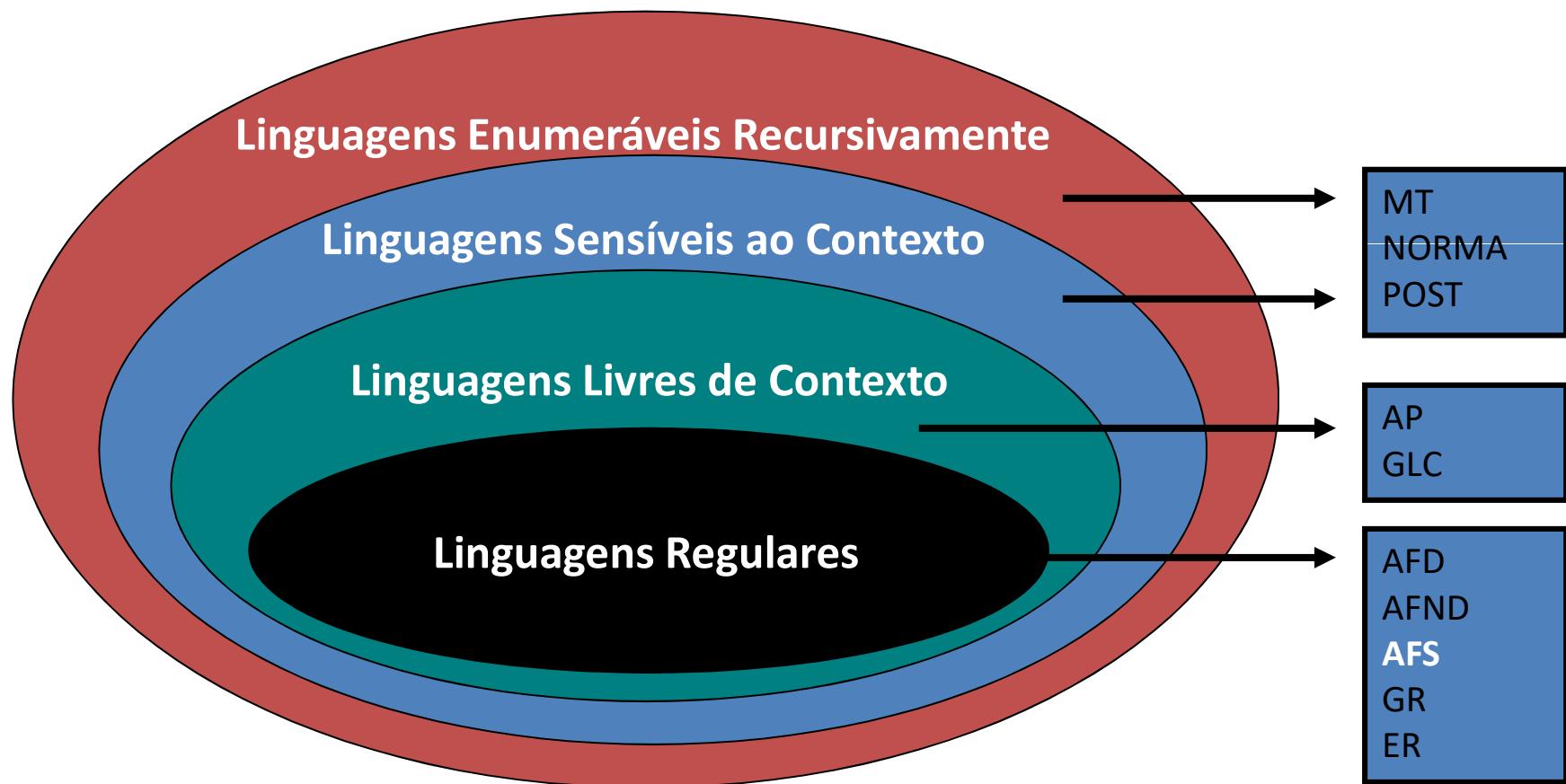
$$(q_2, b) = (q_2)$$

$$(*q_3, a) = (q_3)$$

4. Construa um AFD que reconheça $(a + b)^+ b^*(a+b)^*$ e construa um AFD-mínimo, se possível.



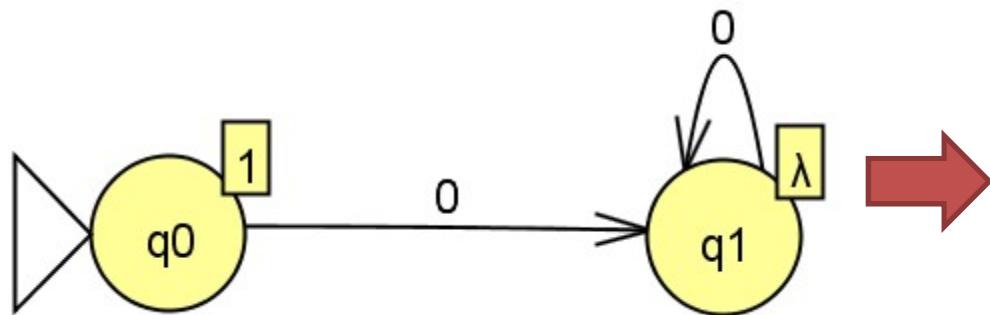
Hierarquia de Chomsky



Autômatos Finitos com Saídas (AFS): Máquina de Moore e Máquina de Mealy

AFS - Máquina de Moore

- Extensões para um AF que associam, para cada estado uma saída sobre o alfabeto, eventualmente distinto do alfabeto de entrada;
- Máquina de Moore: **para cada estado existe uma saída**



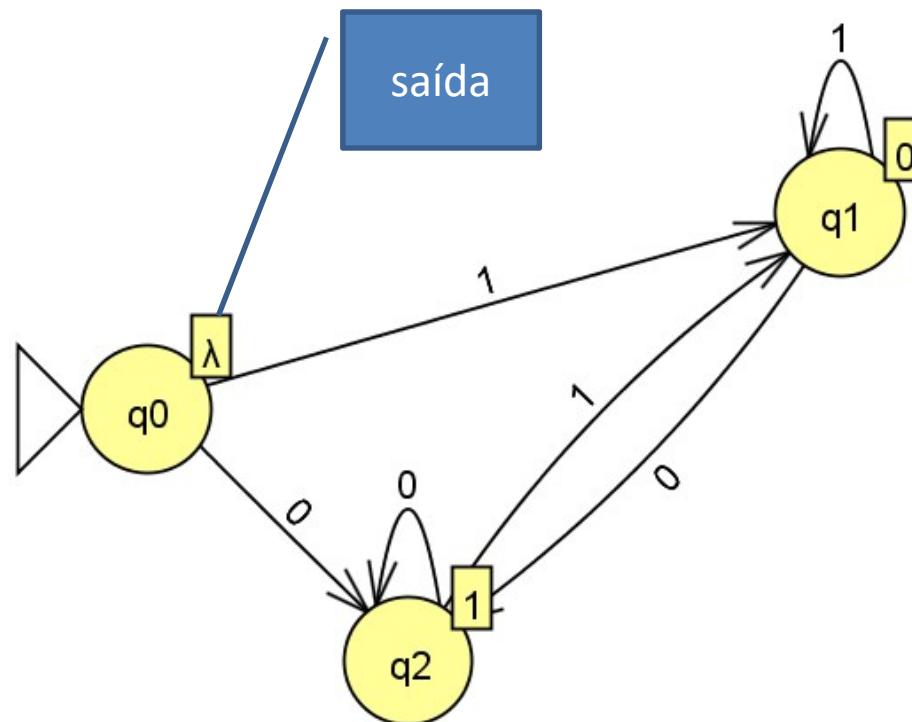
Ao ler o símbolo **0** (q_0) gera 1 na saída (q_0) e avança para o estado q_1 . Em q_1 , a cada 0 lido, gera λ na saída(q_1).

Diferenças entre Máquina de Moore e AF:

- Máquina de Moore não tem estado final;
- Cada estado produz uma saída;
- Não aceita ou rejeita a entrada, em vez disso, ele gera uma saída para a entrada;
- Máquina de Moore deve ser determinística.

Máquina de Moore

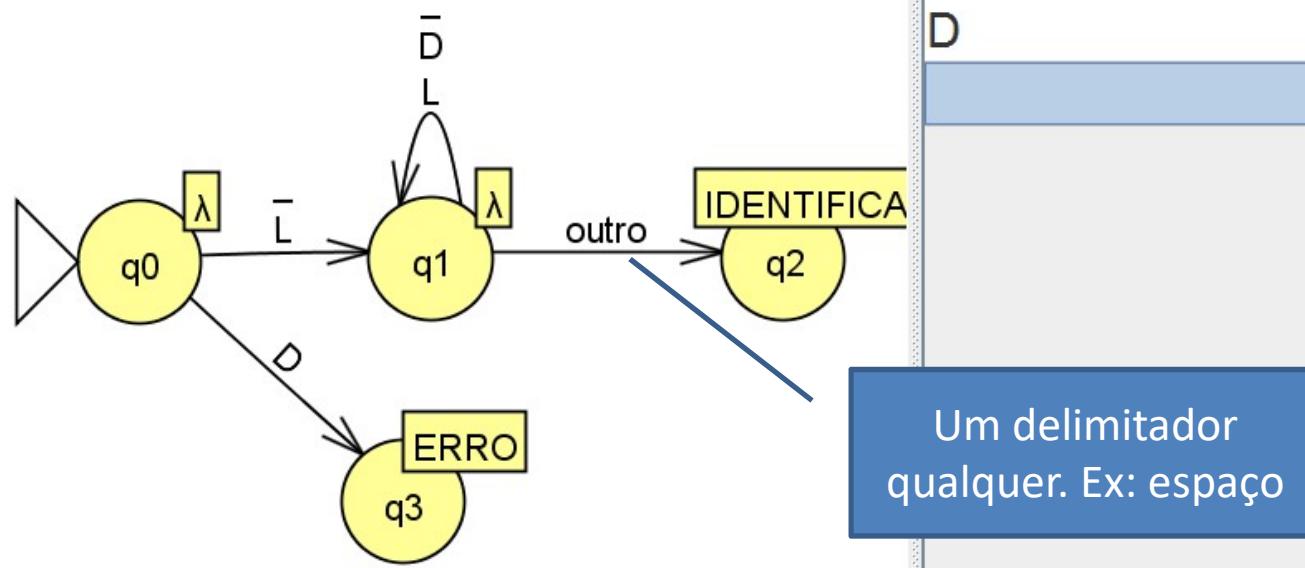
- Exemplo 1: NOT(b): recebe 0 ou 1 e devolve a operação NOT



Input	Result
11	00
101010	010101
00	11
10	01

Máquina de Moore

- Exemplo 2: reconhece *IDENTIFICADOR*

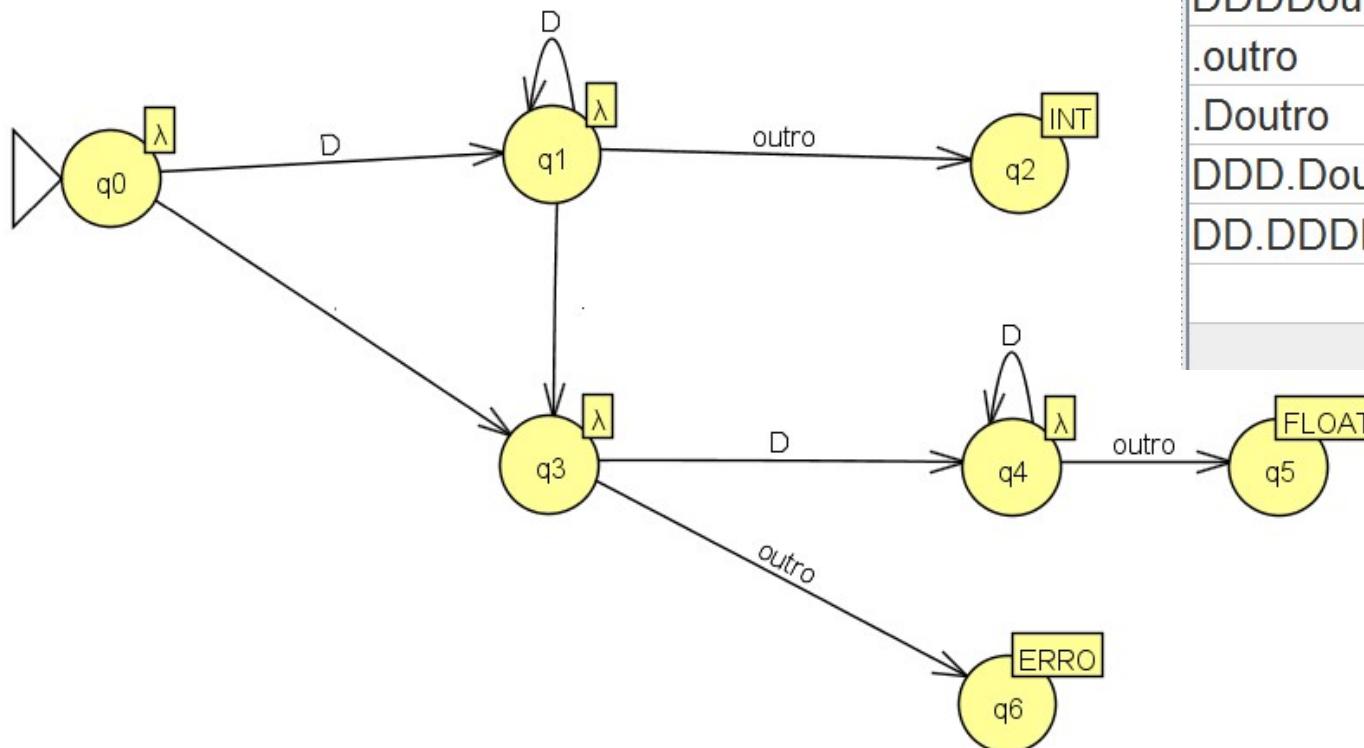


Input	Result
LD_outro	IDENTIFICADOR
Loutro	IDENTIFICADOR
D	ERRO

Um delimitador qualquer. Ex: espaço

Máquina de Moore

- Exemplo 3: reconhece *INT* ou *FLOAT*



Input	Result
DDDDoutro	INT
.outro	ERRO
.Doutro	FLOAT
DDD.Doutro	FLOAT
DD/DDDDoutro	FLOAT

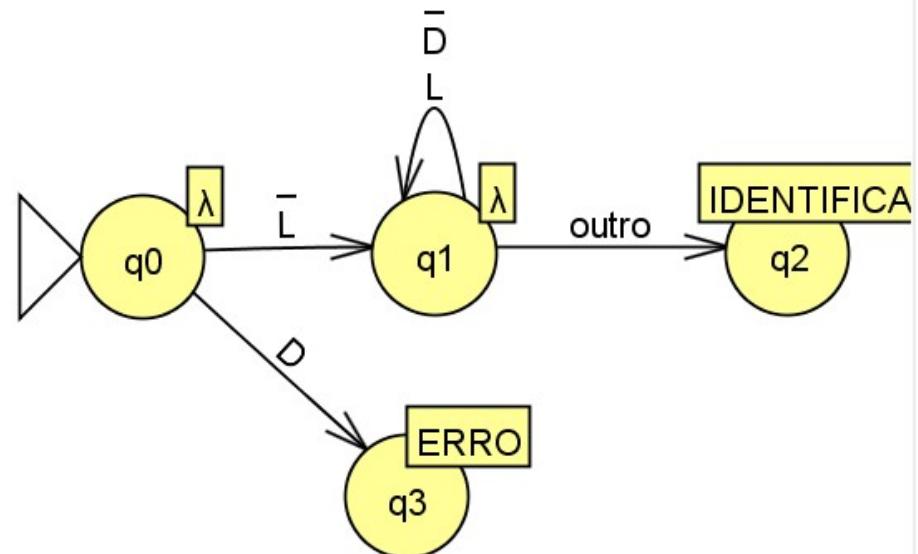
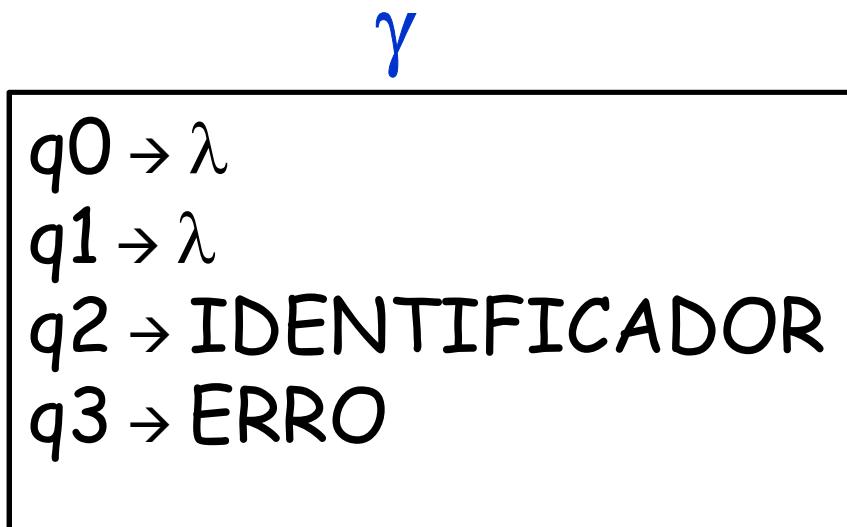
Máquina de Moore

- Denotado por $AF_{Moore} = (\{Q\}, \Sigma, \alpha, \delta, \gamma, q_0)$
 - Um conjunto finito de estados, frequentemente denotado por Q
 - Um alfabeto de entrada, denotado pelo Σ
 - Alfabeto de saída para cada estado $\in \{Q\}$
 - Uma função transição δ que toma como argumentos um estado e um símbolo de entrada e retorna um novo estado.
 - Função transição $\gamma: Q \rightarrow \alpha$
 - q_0 - Um estado inicial (pertencente a Q)

Máquina de Moore

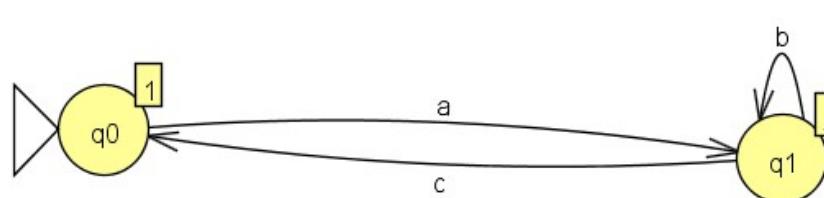
$$AF_{Moore} = (\{Q\}, \Sigma, \alpha, \delta, \gamma, q_0, \{F\})$$

- $AF_{MooreID} = (\{q_0, q_1, q_2, q_3\}, \{L, _, D, \text{outro}\}, \{\lambda, \text{ERRO}, \text{IDENTIFICADOR}\}, \delta, \gamma, q_0)$



Máquina de Moore

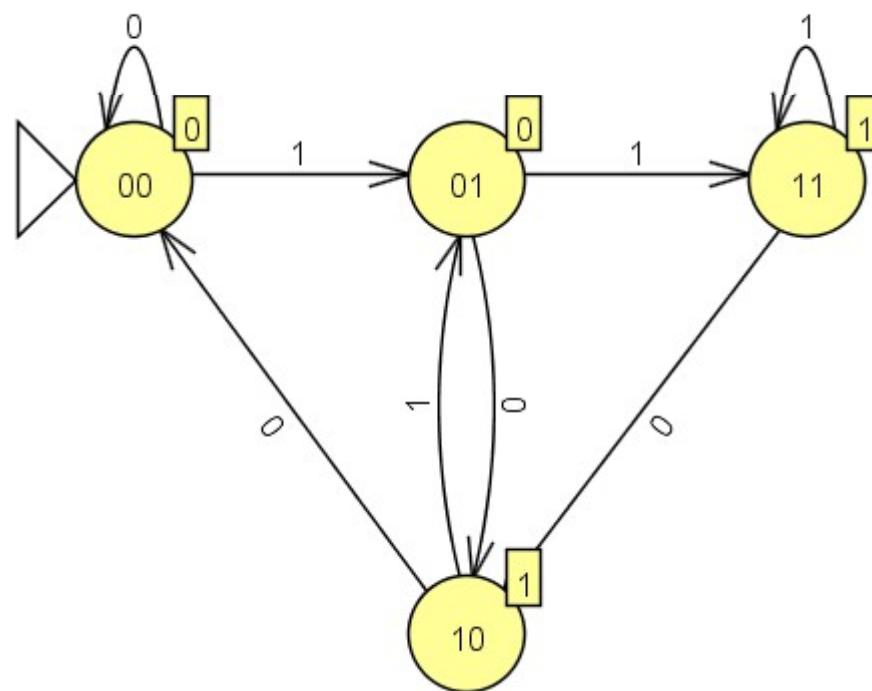
- Exemplo 4: Uma máquina de Moore que aceita a linguagem $ab^*(cab^*)^*$, ou seja, uma sequência de uma ou mais cadeias ab separadas pelo símbolo c . A função de transição γ , neste caso, faz com que a máquina emita o símbolo “1” toda vez que estiver iniciando o reconhecimento de uma nova cadeia com o formato ab^* . Assim, a máquina funciona como um contador do número de subcadeias ab presentes na cadeia de entrada.



Input	Result
ab	1
abcab	11

Máquina de Moore

- Exemplo 5: Número binário dividido por 2

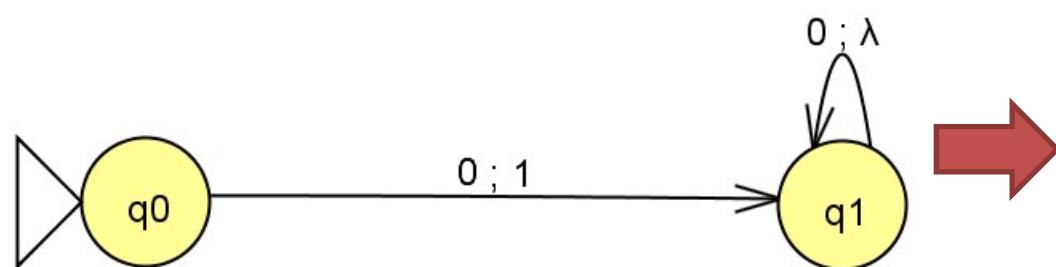


Input	Result
1010	00101
100	0010

Autômatos Finitos com saídas: Máquina de Moore e Máquina de Mealy

AFS - Máquina de Mealy

- Extensões para um AF que associam, a cada transição uma correspondente cadeia de saída sobre um segundo alfabeto, eventualmente distinto do alfabeto de entrada;
- para cada transição existe uma saída



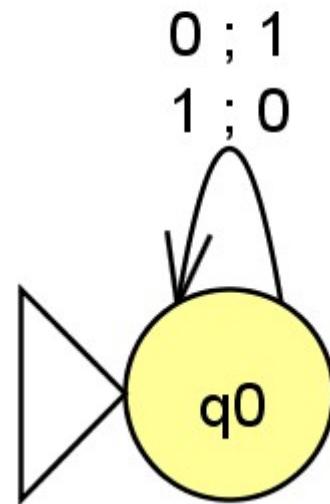
Ao ler o símbolo **0** (q_0) gera 1 na transição (q_0-q_1) e avança para o estado q_1 . Em q_1 , a cada 0 lido, gera λ na transição (q_1-q_1) .

Diferenças entre Máquina de Mealy e AF:

- Não tem estado final;
- Cada transição produz uma saída;
- Não aceita ou rejeita a entrada, em vez disso, ela gera uma saída para a entrada;
- Máquina de Mealy deve ser determinística.

Máquina de Mealy

- Exemplo 1: NOT(b): recebe 0 ou 1 e devolve NOT



Input	Result
101	010
00	11
11	00

Máquina de Mealy

- Denotado por $AF_{Moore} = (\{Q\}, \Sigma, \alpha, \delta, \gamma, q_0)$
 - Um conjunto finito de estados, frequentemente denotado por Q
 - Um alfabeto de entrada, denotado pelo Σ
 - Alfabeto de saída para cada estado $\in \{Q\}$
 - Uma função transição δ que toma como argumentos um estado e um símbolo de entrada e retorna um novo estado.
 - Função transição $\gamma: Q \times \Sigma \rightarrow \alpha$
 - q_0 - Um estado inicial (pertencente a Q)

Máquina de Moore e Mealy - Equivalência

- Apesar de se tratar de dois modelos distintos de AF, pode-se demonstrar a plena equivalência de ambos: toda e qualquer Máquina de Moore pode ser simulada por uma Máquina de Mealy e vice-versa. Dessa maneira, portanto, a opção por um ou outro tipo de máquina pode ser feita levando-se em conta exclusivamente a conveniência de manipulação e a facilidade de representação obtidas conforme o caso em questão.



Exercícios

1. Construa uma máquina (Mealy ou Moore) para o problema abaixo:

A empresa de refrigerantes X deseja projetar um circuito que realize o controle de venda de 1 lata de refrigerante na sua máquina de refrigerantes. Para isto o empresa o contratou, você deve especificar o diagrama de uma máquina estados finitos que realize o controle da entrada de moedas na máquina. Se entrar o valor correto a latinha deve sair da máquina, caso contrário, deve voltar para o estado inicial e devolver as moedas. Sabe-se que o preço do refrigerante é um real, e também que a máquina somente aceita moedas de 1 real, 50 centavos e 25 centavos. Porém, a máquina pode aceitar qualquer sequência de moedas.

2. Construa uma Maquina de Mealy que reconheça uma palavra w , $|w| \geq 1$, tal que $w = x^+$ e produza uma sequencia de saída $v = (ab)^+$, em que $|v| = 2|w|$.



Exercícios

3. Construa o diagrama de estados de uma máquina de estados finitos que realize o controle de um elevador. O elevador deverá respeitar a seguinte especificação:

- Se o elevador está parado e o andar requisitado é igual ao andar corrente , então o elevador continua parado.
- Se o elevador está parado e o andar requisitado é menor que o andar corrente, então o elevador deve descer para o andar desejado.
- Se o elevador está parado e o andar requisitado é maior que o andar corrente, então o elevador deve subir para o andar desejado. Para mais informações consulte o artigo "**Modelos Orientados a Estado na Especificação de Software**" no endereço <http://www.uel.br/revistas/uel/index.php/semejatas/article/view/1571/1322>

Sendo assim:

- a) Desenhe o diagrama da máquina de estados que realiza esta operação.
- b) Esta máquina é Moore ou Mealy? Por quê ?



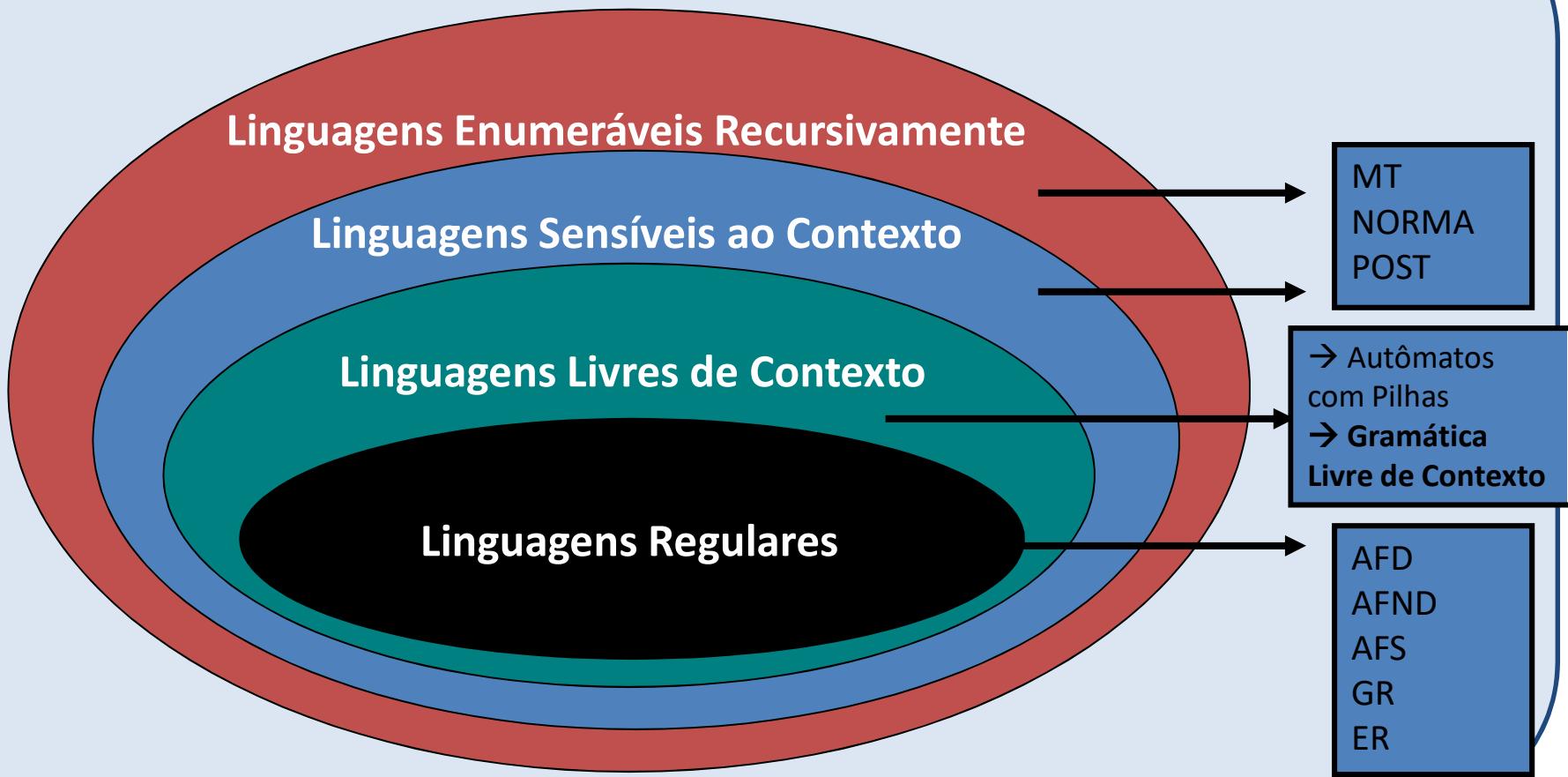
LFA - Aula 08

Gramáticas e Linguagens Livres
de Contexto
(Hopcroft, 2002)

Celso Olivete Júnior

celso.olivete@unesp.br

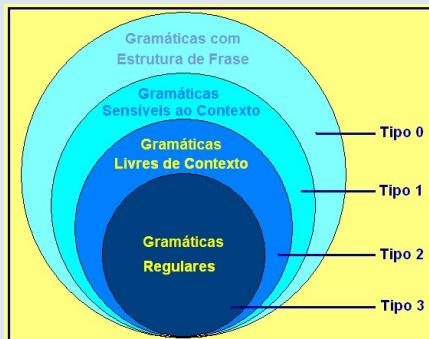
Classes Gramaticais



Classes Gramaticais

a. **Gramáticas com Estrutura de Frase ou Tipo 0:** atuam no reconhecimento das Linguagens Enumeráveis Recursivamente

- São aquelas às quais nenhuma restrição é imposta.
 - Exemplo de reconhecedor: Máquina de Turing com fita de entrada infinita
- Produções da forma
$$\alpha \rightarrow \beta$$
Onde: $\alpha \in (V_n \cup V_t)^+$
$$\beta \in (V_n \cup V_t)^*$$
- Lado esquerdo da regra de produção pode conter N símbolos (terminais ou não terminais);
- Lado direito da regra de produção pode conter N símbolos (terminais ou não terminais ou **vazio**);



$$G = (V_n, V_t, P, S)$$

Classes Gramaticais

a. Gramáticas com Estrutura de Frase ou Tipo 0

- Exemplo de GEF:

$$G = (\{A, B, C\}, \{a, b\}, P, A)$$

$$P: \quad A \rightarrow BC$$

$$BC \rightarrow CB$$

$$B \rightarrow b$$

$$C \rightarrow a$$

- Qual a linguagem gerada?

- $L(G) = \{ba, ab\}$

Classes Gramaticais

b. Gramáticas Sensíveis ao Contexto ou Tipo 1: atuam no reconhecimento das Linguagens Sensíveis ao Contexto

- Restrição: nenhuma substituição pode **reduzir o comprimento** da forma sentencial à qual a substituição é aplicada.

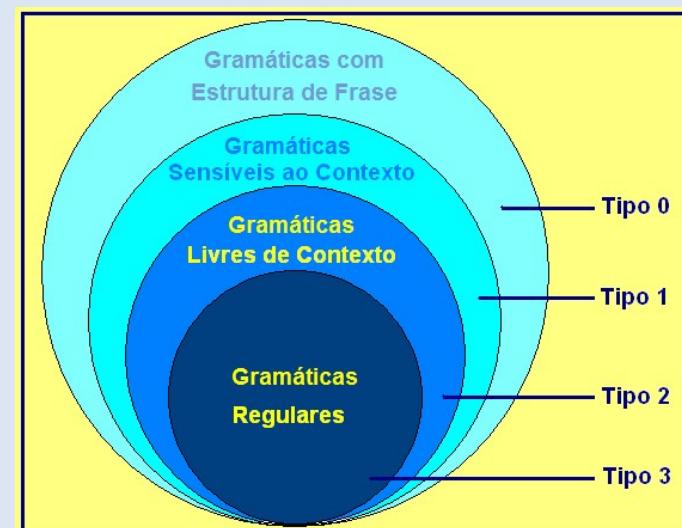
- Produções da forma

$$\alpha \rightarrow \beta$$

Onde: $\alpha \in (\mathbf{Vn} \cup \mathbf{Vt})^+$

$$\beta \in (\mathbf{Vn} \cup \mathbf{Vt})^+$$

$$|\alpha| \leq |\beta|$$



Classes Gramaticais

b. Gramáticas Sensíveis ao Contexto ou Tipo 1

□ Exemplo de GSC:

$$G = (\{S, B, C\}, \{a, b, c\}, P, S)$$

$$P : \begin{aligned} S &\rightarrow aSBC \mid aBC \\ CB &\rightarrow BC \\ aB &\rightarrow ab \\ bB &\rightarrow bb \\ bC &\rightarrow bc \\ cC &\rightarrow cc \end{aligned}$$

$$\alpha \rightarrow \beta$$

Onde: $\alpha \in (V_n \cup V_t)^+$
 $\beta \in (V_n \cup V_t)^+$
 $|\alpha| \leq |\beta|$

Faça a derivação (mais
à esquerda ou mais à
direita)

□ Qual a linguagem gerada?

$$\square L(G) = \{a^n b^n c^n\}$$

Classes Gramaticais

c. Gramáticas Livres de Contexto ou Tipo 2: atuam no reconhecimento das Linguagens Livres de Contexto

- As **Gramáticas Livres de Contexto (GLC) ou do Tipo 2** são aquelas que no lado esquerdo da regra há apenas um símbolo não-terminal e no lado direito pode haver n terminais ou não-terminais.

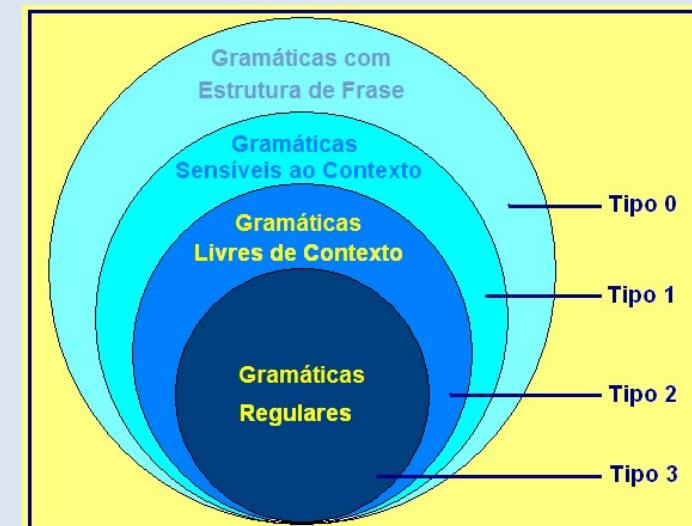
- Produções da forma

$$\alpha \rightarrow \beta$$

Onde: $\alpha \in (\mathbf{Vn})$

$$\beta \in (\mathbf{Vn} \cup \mathbf{Vt})^+$$

$$|\alpha| = 1 \quad |\beta| > 0$$



Classes Gramaticais

c. Gramáticas Livres de Contexto ou Tipo 2

□ Qual a linguagem gerada para:

$$G = (\{S, A, B\}, \{a, b\}, P, S)$$

$$P: S \rightarrow AB$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b$$

Faça a derivação (mais
à esquerda ou mais à
direita) para a entrada
abbb

$$L(G) = \{a^n b^m\}$$

Classes Gramaticais

d. Gramáticas Regulares ou Tipo 3: atuam no reconhecimento das Linguagens Regulares

□ Aplicando-se mais uma restrição sobre a forma das produções, pode-se criar uma nova classe de gramáticas, as Gramáticas Regulares (GR), de grande importância no estudo dos compiladores por possuírem propriedades adequadas para a obtenção de reconhecedores simples. Nas GRs, as produções são restritas às formas seguintes:

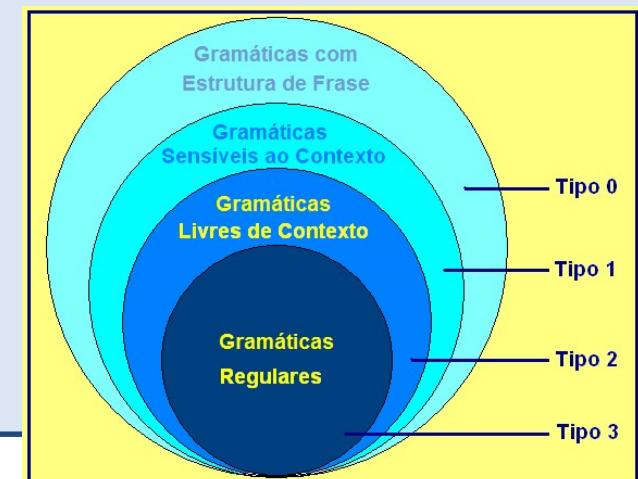
$$A \rightarrow aB$$

$$A \rightarrow a$$

$$A \rightarrow \epsilon$$

onde $A, B \in V_n$ e $a \in V_t$, com $|A|=1$ e $|B| \leq 1$

Compiladores



d. Gramáticas Regulares ou Tipo 3

- Exemplo gramática em EBNF:

$G = (\{<\text{Dig}>, <\text{Int}>\}, \{+, -, 0, \dots, 9\}, P, <\text{Int}>)$

P: $<\text{Int}> ::= +<\text{Dig}> \mid -<\text{Dig}>$

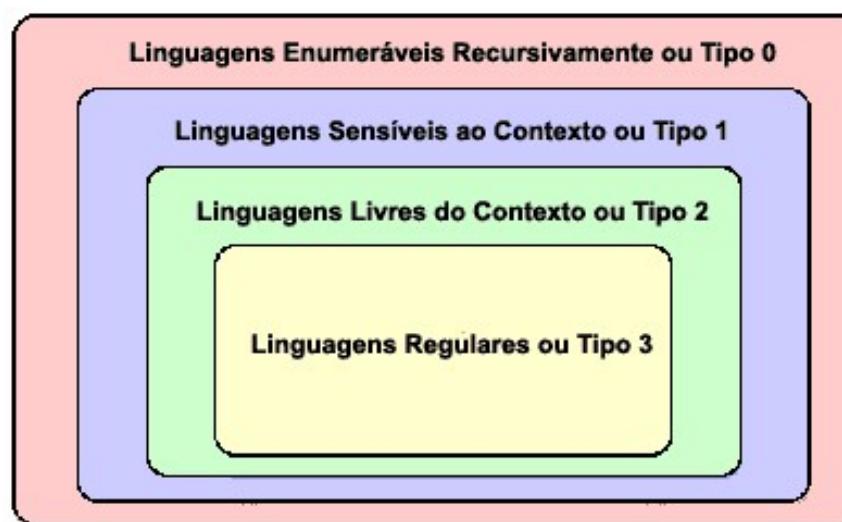
$<\text{Dig}> ::= 0<\text{Dig}> \mid 1<\text{Dig}> \mid \dots \mid 9<\text{Dig}> \mid 0 \mid 1 \mid 2 \mid \dots \mid 9$

- Qual linguagem gerada?

➤ $L(G) = \text{conj. números inteiros com sinal } \pm[0..9]$

Introdução

- Linguagens livre de contexto: abrange uma classe maior de linguagens.
 - A maior aplicação das gramáticas livres de contexto (GLC) ocorre na formalização sintática das linguagens de programação de alto nível;



Gramáticas livre de contexto

- Exemplo de utilização:
 - no processo de compilação → análise sintática
 - para descrever formatos de documentos (DTD), utilizados para troca de informações na Web (XML)

Gramáticas livre de contexto - aplicação na análise sintática

- Cada linguagem de programação possui regras que descrevem a estrutura sintática dos programas. Em C, por exemplo, um programa é constituído por blocos, um bloco por comandos, comandos por expressões, uma expressão por tokens, e assim por diante. Desta forma o analisador sintático cuida exclusivamente da forma das sentenças da linguagem, baseando-se na gramática que define a linguagem.



Gramáticas livre de contexto - exemplo

Exemplo 1: expressões constituídas por dígitos e sinais de mais e menos, como "9-5+2" e "3-1+6" podem ser descritas através da gramática:

$$\begin{aligned} <\text{expr}> &\rightarrow <\text{num}> \mid <\text{num}> + <\text{expr}> \mid <\text{num}> - <\text{expr}> \\ <\text{num}> &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{aligned}$$

ou

$$\begin{aligned} <\text{expr}> &\rightarrow <\text{num}> \\ <\text{expr}> &\rightarrow <\text{num}> + <\text{expr}> \\ <\text{expr}> &\rightarrow <\text{num}> - <\text{expr}> \\ <\text{num}> &\rightarrow 0 \\ <\text{num}> &\rightarrow 1 \\ <\text{num}> &\rightarrow \dots \\ <\text{num}> &\rightarrow 9 \end{aligned}$$



- Uma **gramática** livre de contexto é uma **notação formal** para **expressar** uma **linguagem**.
- Uma **gramática** consiste em uma ou mais **variáveis** que **representam** linguagens.

Gramáticas e Linguagens Livre de Contexto

- Formalmente as gramáticas são caracterizadas como quádruplas ordenadas

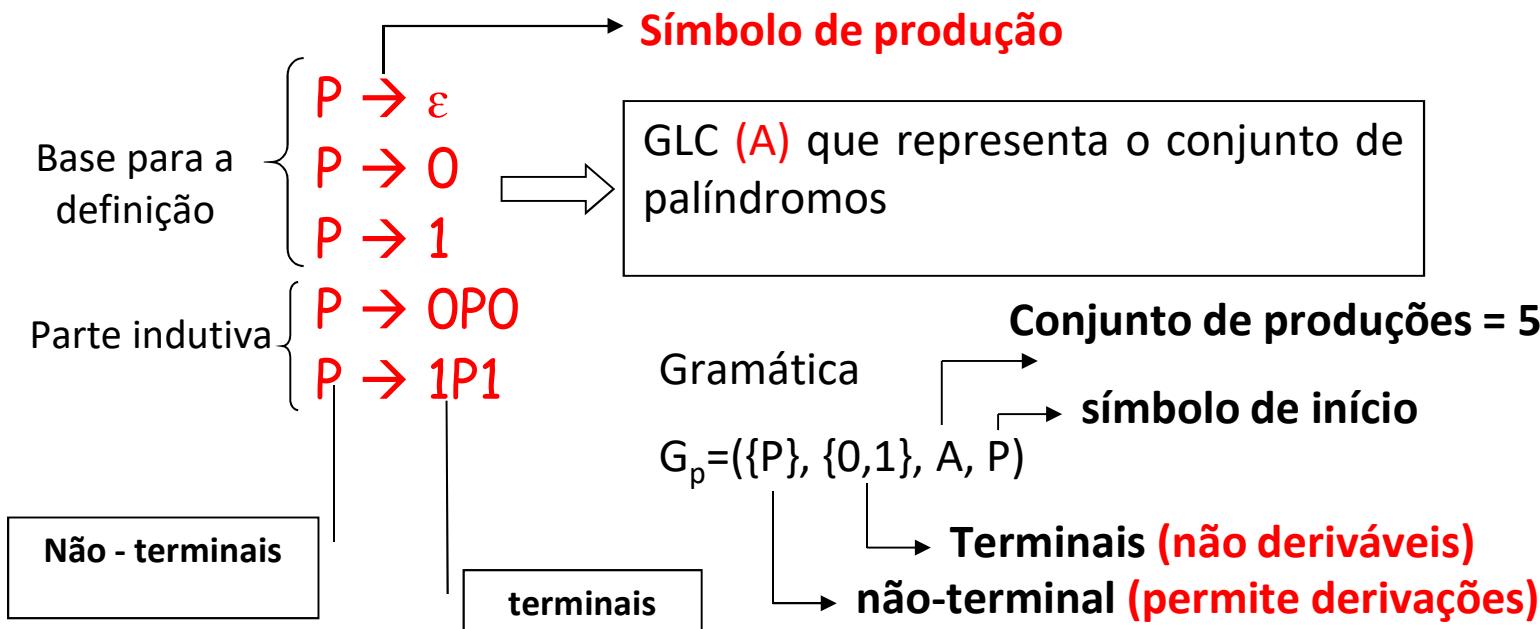
$$G = (\{V\}, T, P, S)$$

- onde:

- V representa o vocabulário não terminal da gramática - **variáveis**.
- T é o vocabulário **terminal**, contendo os símbolos que constituem as sentenças da linguagem.
- P representa o conjunto de todas as leis de formação (**regras de produção**) utilizadas pela gramática para definir a linguagem.
- S representa o símbolo de **íncio**

Gramáticas e Linguagens Livre de Contexto

- Ex: definição de uma GLC para ling. palíndromos
 - Caracteres base para o palíndromo: ϵ , 0 e 1
 - Se w é um palíndromo, então OPO e 1P1 também são.
 - Definição de uma GLC que representa o conjunto de palíndromos a partir dos caracteres base:



Gramáticas e Linguagens Livre de Contexto

- Exemplo 2: uma GLC que representa uma simplificação de expressões em uma linguagem de programação.
 - Operadores da linguagem + e *
 - Identificadores → formados por letras seguidas de zero ou mais letras e dígitos

$$(a + b) (a + b + 0 + 1)^*$$
 - Letras são formadas apenas por a e b
 - Dígitos apenas por 0 e 1
 - Variáveis da gramática: E (expressões) e I (identificadores)
 - Conjunto de Produções que representam essa ER

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$

- > base
- > E conectada por um sinal de adição
- > E conectado com o simb. multiplicação
- > E entre parênteses
- > identificador

6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

->identificador

Gramáticas e Linguagens Livre de Contexto

■ Notação / Convenções

- Não-terminais: letras do alfabeto maiúsculas $\{A, B, \dots, Z\}$
- Terminais: letras do início do alfabeto minúsculas $\{a, b, c, \dots\}$, dígitos $\{0..9\}$ e outros caracteres como $+, -, *, /$



Gramáticas e Linguagens Livre de Contexto

■ Exercícios

1. Crie as regras de produção (P) e defina a gramática G (quádrupla) das linguagens abaixo:

a) Uma linguagem que define expressões envolvendo elementos de 0 a 9, somas, subtrações, multiplicações, divisões e expressões entre parênteses.

b) $L(G) = \{ba, ab\}$

c) $L(G) = \{0^n 1^m \mid n, m \geq 0\}$ ou $0^* 1^*$

d) $L(G) = \{(01)^n \mid n \geq 1\}$

e) $L(G) = \{(011)^n \mid n \geq 1\}$

f) $L(G) = \{a^n b^n c^i \mid n \geq 1 \text{ e } i \geq 0\}$

As **Gramáticas Livres de Contexto (GLC)** ou do **Tipo 2** são aquelas que no lado esquerdo da regra há apenas um símbolo não-terminal. Do lado direito pode existir **n** não-terminais e terminais (**$n \geq 0$**)

Gramáticas e Linguagens Livre de Contexto

■ Resolução

- a) Uma linguagem que define expressões envolvendo elementos de 0 a 9, somas, subtrações, multiplicação, divisões e expressões entre parênteses.

$$G = (\{E, D\}, \{0..9, +, *, -, /, (,)\}, P, E)$$

1. $E \rightarrow E + E$
2. $E \rightarrow E - E$
3. $E \rightarrow E * E$
4. $E \rightarrow E / E$
5. $E \rightarrow (E)$
6. $E \rightarrow D$
7. $D \rightarrow 0$
8. $D \rightarrow 1$
9. $D \rightarrow \dots 9$

GLC: reconhecimento de cadeias

- Exemplo:

1. $S \rightarrow A1B$
2. $A \rightarrow 0A \mid \epsilon$
3. $B \rightarrow 0B \mid 1B \mid \epsilon$

- O processo de reconhecimento de uma cadeia por uma GLC pode ser feita fazendo a leitura das regras de produção, desde a primeira até a última regra. Esse processo é conhecido por *inferência recursiva*.
- No exemplo acima começamos com uma regra que nos leva a um **A** (não-terminal) e esse **A** é formado por **0** seguido de um **A**, que pode ficar em loop (**A^***) ou não aparecer nenhuma vez (ϵ), seguido de **1** resultando em **0^*1**
- Seguido de um **B**, que nos leva a um **0B** ou **1B**. Esse **B** é um não terminal que pode aparecer uma, nenhuma ou várias vezes, resultando em **$(0 + 1)^*$**
- Linguagem aceita pela GLC = **$0^*1 (0 + 1)^*$**

GLC: reconhecimento de cadeias

- Existe uma outra maneira de fazer o reconhecimento, conhecida por **derivação**, onde é feita uma expansão (derivação) de uma das primeiras regras de produção que formam a base. Esse processo é definido pelo símbolo \Rightarrow

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$

6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

Exemplo: reconhecer a
 $ER = a * (a + b00)$

Ex:

$$\begin{aligned}
 E &\Rightarrow E * E \Rightarrow I * E \Rightarrow a * E \Rightarrow \\
 a * (E) &\Rightarrow a * (E + E) \Rightarrow a * (I + E) \Rightarrow \\
 a * (a + E) &\Rightarrow a * (a + I) \Rightarrow \\
 a * (a + IO) &\Rightarrow a * (a + I00) \Rightarrow \\
 a * (a + b00)
 \end{aligned}$$

- **Observe** que no decorrer da substituição prevaleceu a substituição de uma variável mais à esquerda

GLC: reconhecimento de cadeias

- O processo de **derivação** pode ocorrer mais à **esquerda** ou mais à **direita**.
 - Dá-se o nome de GLC com **derivação à esquerda** e GLC com **derivação à direita**



Derivação mais à esquerda

- A variável mais à esquerda de uma string sempre é substituída por uma regra de produção. Usa-se o símbolo \xrightarrow{E} para representar essa derivação. Ex: derivar mais à esquerda a produção para gerar a ER = $a^* (a + b00)$

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$

6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow IO$
10. $I \rightarrow I1$

Ex:

$$\begin{aligned}
 E &\xrightarrow{E} E^* E \xrightarrow{E} I^* E \xrightarrow{E} a^* E \xrightarrow{E} \\
 a^* (E) &\xrightarrow{E} a^* (E + E) \xrightarrow{E} a^* (I + E) \xrightarrow{E} \\
 a^* (a + E) &\xrightarrow{E} a^* (a + I) \xrightarrow{E} \\
 a^* (a + IO) &\xrightarrow{E} a^* (a + I00) \xrightarrow{E} \\
 a^* (a + b00)
 \end{aligned}$$



Derivação mais à direita

- A variável mais à direita de uma string sempre é substituída por uma regra de produção. Usa-se \xrightarrow{D} o símbolo para representar essa derivação. Ex: $ER = a^* (a + b00)$

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$

6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow IO$
10. $I \rightarrow I1$

Ex:

$$\begin{aligned}
 E &\xrightarrow{D} E * E \xrightarrow{D} E * (E) \xrightarrow{D} E * (E + E) \xrightarrow{D} \\
 &E * (E + I) \xrightarrow{D} E * (E + IO) \xrightarrow{D} \\
 &E * (E + IOO) \xrightarrow{D} E * (E + b00) \xrightarrow{D} \\
 &E * (I + b00) \xrightarrow{D} E * (a + b00) \xrightarrow{D} I \\
 &* (a + b00) \xrightarrow{D} a * (a + b00)
 \end{aligned}$$

Formas sentenciais de uma GLC

- As derivações à partir do símbolo inicial da produção são conhecidas como formas sentenciais. Podendo ser uma *forma sentencial à esquerda* ou *uma forma sentencial à direita*.
- Exemplo de forma sentencial à esquerda

Ex:

$$E \xrightarrow{E} E^* E \xrightarrow{E} I^* E \xrightarrow{E} a^* E$$

- Forma sentencial à direita

Ex:

$$E \xrightarrow{D} E^* E \xrightarrow{D} E^* (E) \xrightarrow{D} E^* (E + E)$$

Exercícios

- 2) Exercício 5.1.2 e 5.1.4 (pág. 192)
- 3) Forneça a linguagem gerada e a Gramática da seguinte regra de produção

1. A → 0B 0
2. B → 1C
3. C → 0B 0

- 4) A partir da Gramática (G) e da produção (P) abaixo, aplique o processo de derivação (à esquerda ou à direita) para saber qual é a linguagem gerada.

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

P

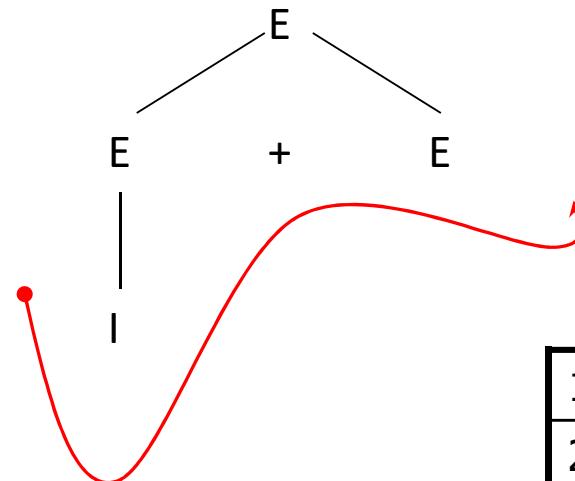
1. S → 0B1
2. B → 01

Árvores de análise sintática

- As derivações de uma gramática também podem ser representadas através de **árvores**.
- **Construção** da árvore de análise sintática
 1. Cada nó é rotulado por uma variável em V
 2. Cada folha é rotulada por uma variável, um terminal ou ϵ , ela deve ser o único filho do seu pai
 3. Se um nó é rotulado por A e seus filhos são rotulados por $X_1, X_2, X_3, \dots, X_k$. Então: $A \rightarrow X_1, X_2, X_3, \dots, X_k$ é uma produção em P. O único momento em que ϵ pode aparecer é quando $A \rightarrow \epsilon$ for uma produção em G

Árvores de Análise Sintática

- Exemplo 1: árvore mostrando a derivação de $I + E$ a partir de E



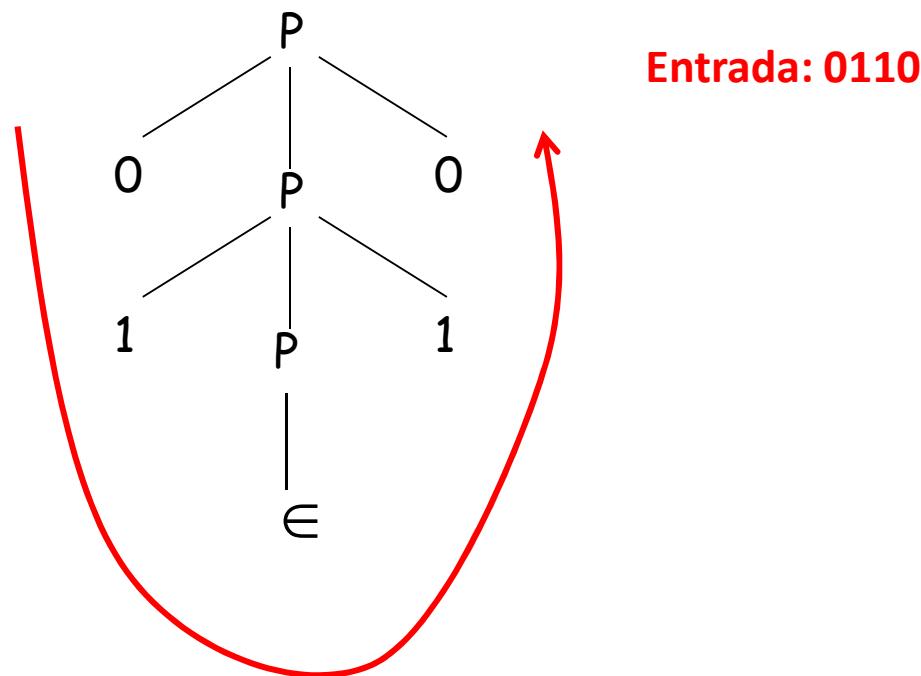
1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$

6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

Árvores de Análise Sintática

- Exemplo 2: árvore mostrando a derivação para a gramática de palíndromos

$P \rightarrow \epsilon$
 $P \rightarrow 0$
 $P \rightarrow 1$
 $P \rightarrow OPO$
 $P \rightarrow 1P1$

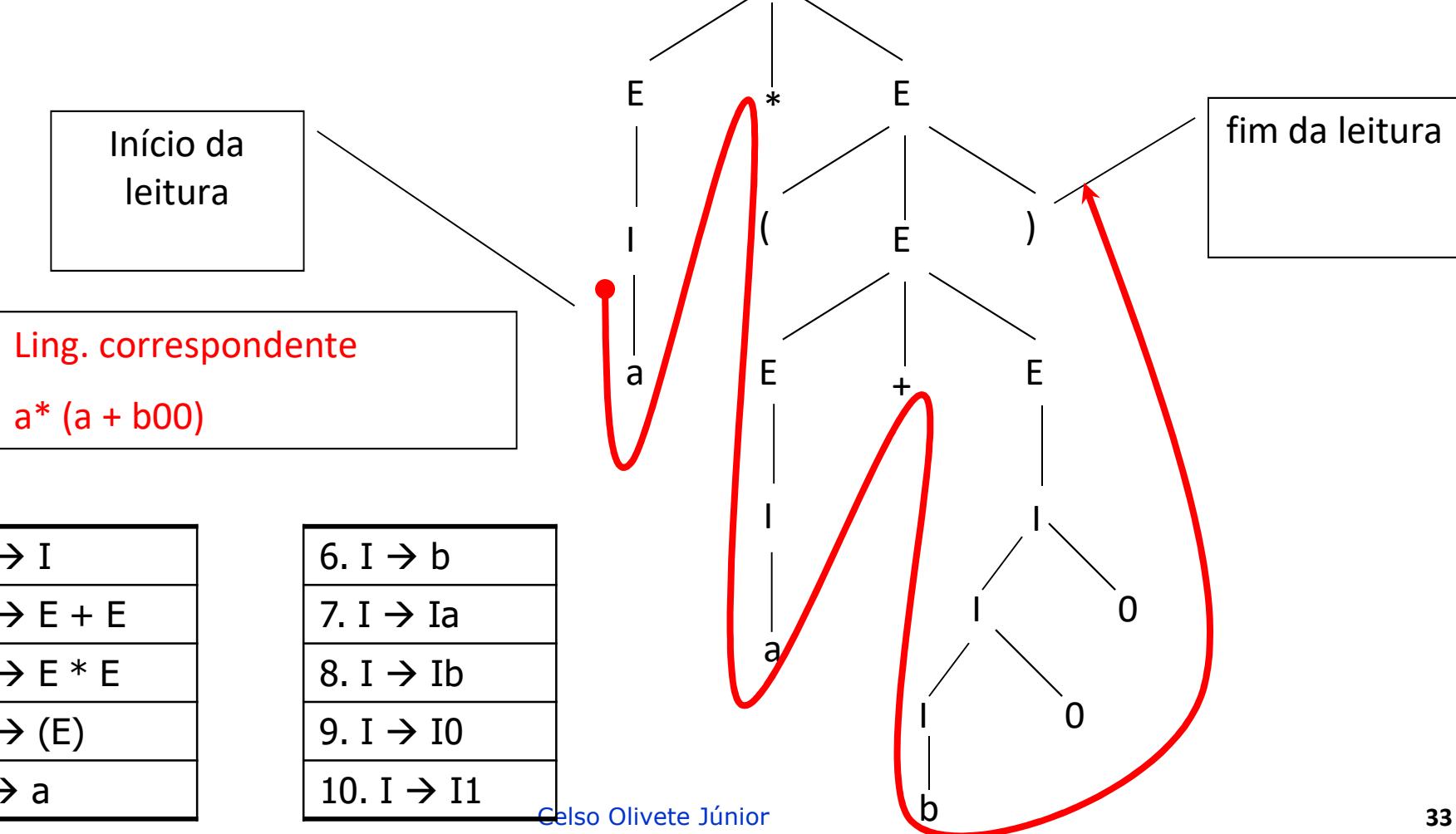


O resultado de uma árvore de análise sintática

- A leitura de uma árvore é feita a partir da concatenação das folhas a partir da esquerda. O resultado dessa leitura é conhecido por **resultado da árvore** (que é um string derivado a partir da variável raiz)

O resultado de uma árvore de análise sintática

- Exemplo de leitura





Exercícios

- 5) Exercício - resolver o exercício 5.2.1 (pág. 205)
- 6) Construa a árvore de análise sintática para a gramática do exercício 3 e 4.
- 7) Considere a gramática $G = (\{S\}, \{a, b\}, \{S \rightarrow ab; S \rightarrow ba; S \rightarrow SS; S \rightarrow aSb; S \rightarrow bSa\}, S)$.
 - a) Determine, justificando, a linguagem gerada pela gramática G .
 - b) Determine uma derivação à esquerda, caso exista, da seguinte sentença: **aababbba**
- 8) Considere a gramática $G = (\{S\}, \{0, 1\}, \{S \rightarrow 0S0; S \rightarrow 1X; X \rightarrow 1X; X \rightarrow 1\}, S)$. Determine, justificando, a linguagem gerada pela gramática G .

Exercícios

9) Considere a gramática $G = (S, T, P, A)$ que representa o cabeçalho de métodos na linguagem Java (sem os modificadores de acesso), onde

$$\Sigma = \{ S, \text{Type}, \text{Param}, \text{Exception}, \text{ParamList} \}$$

$$T = \{ \text{id}, \text{int}, \text{boolean}, (,), .., \text{throws}, \text{ArithmeticException} \}$$

P é formado pelas seguintes produções :

$$S \rightarrow \text{Type id (Param) Exception}$$

$$\text{Type} \rightarrow \text{int} \mid \text{boolean}$$

$$\text{Param} \rightarrow \epsilon \mid \text{ParamList}$$

$$\text{ParamList} \rightarrow \text{Type id} \mid \text{ParamList , Type id}$$

$$\text{Exception} \rightarrow \epsilon \mid \text{throws ArithmeticException}$$

a) Encontre a derivação mais à esquerda de
int id (boolean id) throws ArithmeticException

Exercícios

10) Considere a seguinte gramática $G = (S, T, P, A)$ que descreve uma versão simplificada de escrita de documentos na linguagem MathML (*Mathematical Markup Language*), onde

$$\Sigma = \{ D, A, C \}$$

$$T = \{ <, >, /, ci, cn, apply, =, id, string, plus, sin, number \}$$

P é formado pelas seguintes produções :

$$D \rightarrow < ci \ A > C < / ci > \mid < cn > number < / cn > \mid < apply > C < / apply >$$

$$A \rightarrow \epsilon \mid A \ id = string$$

$$C \rightarrow id \mid < sin / > D \mid < plus / > D \ D$$

a) Encontre a derivação mais à esquerda para

$< apply > < plus / > < ci > id < / ci > < cn >$
 $number < / cn > < / apply >$