

Linguagens Formais e Autômatos

P. Blauth Menezes

blauth@inf.ufrgs.br

**Departamento de Informática Teórica
Instituto de Informática / UFRGS**



Linguagens Formais e Autômatos

P. Blauth Menezes

- 1 **Introdução e Conceitos Básicos**
- 2 **Linguagens e Gramáticas**
- 3 **Linguagens Regulares**
- 4 **Propriedades das Linguagens Regulares**
- 5 **Autômato Finito com Saída**
- 6 **Linguagens Livres do Contexto**
- 7 **Propriedades e Reconhecimento das Linguagens Livres do Contexto**
- 8 **Linguagens Recursivamente Enumeráveis e Sensíveis ao Contexto**
- 9 **Hierarquia de Classes e Linguagens e Conclusões**

3 – Linguagens Regulares

- 3.1 Sistema de Estados Finitos**
- 3.2 Composição Seqüencial, Concorrente e Não-Determinista**
- 3.3 Autômato Finito**
- 3.4 Autômato Finito Não-Determinístico**
- 3.5 Autômato Finito com Movimentos Vazios**
- 3.6 Expressão Regular**
- 3.7 Gramática Regular**



3 – Linguagens Regulares

3 Linguagens Regulares

◆ Linguagens Regulares ou Tipo 3 - formalismos

- Autômato Finito
 - * formalismo operacional ou reconhecedor
 - * basicamente, um sistema de estados finitos
- Expressão Regular
 - * formalismo denotacional ou gerador
 - * conjuntos (linguagens) básicos + concatenação e união
- Gramática Regular
 - * formalismo axiomático ou gerador
 - * gramática com restrições da forma das regras de produção

◆ Hierarquia de Chomsky

- classe de linguagens mais simples
- algoritmos de reconhecimento, geração ou conversão entre formalismos
 - * pouca complexidade
 - * grande eficiência
 - * fácil implementação

◆ Fortes limitações de expressividade

- exemplo: duplo balanceamento não é regular.
- linguagens de programação em geral: não-regulares

◆ Complexidade de algoritmos - autômatos finitos

- classe de **algoritmos mais eficientes** (tempo de processamento)
 - * supondo determinada condição
- **qualquer** autômato finito é **igualmente eficiente**
- *qualquer solução é ótima*
 - * a menos de eventual redundância de estados
- **redundância de estados**
 - * (não influi no tempo)
 - * pode ser facilmente eliminada: **Autômato Finito Mínimo**

◆ **Importantes propriedades - podem ser usadas para**

- **construir** novas **linguagens** regulares
 - * a partir de **linguagens** regulares **conhecidas**
 - * (definindo uma **álgebra**)
- **provar** propriedades
- **construir** algoritmos

◆ **Se um problema tiver uma solução regular**

- considerar **preferencialmente** a qualquer **outra não-regular**
 - * **propriedades** da Classe
 - * **eficiência** e **simplicidade** dos algoritmos

◆ Universo de aplicações das linguagens regulares

- muito grande
- constantemente ampliado

◆ Exemplo típico e simples

- análise léxica

◆ Exemplos mais recentes

- sistemas de animação
- hipertextos
- hipermídias

◆ Capítulos subseqüentes

- **minimização** de autômatos finitos
- **propriedades** da Classe
- algumas importantes **aplicações**

3 – Linguagens Regulares

3.1 Sistema de Estados Finitos

3.2 Composição Seqüencial, Concorrente e Não-Determinista

3.3 Autômato Finito

3.4 Autômato Finito Não-Determinístico

3.5 Autômato Finito com Movimentos Vazios

3.6 Expressão Regular

3.7 Gramática Regular

3.1 Sistema de Estados Finitos

◆ Sistema de Estados Finitos

- modelo matemático de sistema com entradas e saídas discretas
- número *finito* e *predefinido* de estados
 - * podem ser definidos antes de iniciar o processamento

◆ Estado

- somente informações do passado
- necessárias para determinar as ações para a próxima entrada

◆ Motivacional

- associados a diversos tipos de sistemas naturais e construídos

Exp: Elevador

Não memoriza as requisições anteriores

- Estado: sumaria "andar corrente" e "direção de movimento"
- Entrada: requisições pendentes

Exp: Analisador Léxico, Processador de Texto

- Estado: memoriza a estrutura do prefixo da palavra em análise
- Entrada: texto

◆ Restrição

- nem todos os sistemas de estados finitos
- são adequados para serem estudados por esta abordagem

Exp: Cérebro humano

- Neurônio: número finito de bits
- Cérebro: cerca de 2^{35} células
 - * abordagem pouco eficiente
 - * explosão de estados

Exp: Computador

- processadores e memórias: sistema de estados finitos
- estudo da computabilidade
 - * exige uma memória *sem limite predefinido*
- Máquina de Turing
 - * mais adequado ao estudo da computabilidade
- computabilidade e solucionabilidade de problemas
 - * apenas introduzido
 - * questões tratadas na Teoria da Computação

3 – Linguagens Regulares

3.1 Sistema de Estados Finitos

3.2 Composição Seqüencial, Concorrente e Não-Determinista

3.3 Autômato Finito

3.4 Autômato Finito Não-Determinístico

3.5 Autômato Finito com Movimentos Vazios

3.6 Expressão Regular

3.7 Gramática Regular

3.2 Composição Seqüencial, Concorrente e Não-Determinista

◆ Construção composicional de sistema

- construído a partir de sistemas conhecidos
 - * e assim sucessivamente
 - * até chegar ao nível mais elementar (como uma ação atômica)

◆ Composição

- Seqüencial
- Concorrente
- Não-Determinista

◆ Seqüencial

- execução da próxima componente
- depende da terminação da componente anterior

◆ Concorrente

- componentes independentes
 - * ordem em que são executadas não é importante
- portanto, podem ser processadas ao mesmo tempo

◆ Não-Determinista

- próxima componente: **escolha** entre diversas **alternativas**
- em oposição à determinista
 - * para as mesmas condições
 - * próxima componente é sempre a mesma
- não-determinismo pode ser
 - * **interno**: **sistema escolhe** aleatoriamente
 - * **externo**: escolha externa ao sistema

◆ Sistemas reais

- as **três formas** de composição são **comuns**

Exp: Banco

Seqüencial

- **fila**: próximo cliente depende do atendimento do anterior
- **pagamento** de uma conta depende do fornecimento de um valor

Concorrente

- diversos **caixas atendem independentemente** diversos clientes
- **clientes nos caixas**: ações **independentemente** dos **clientes** na **fila**

Não-determinista

- **dois ou mais caixas disponíveis** ao mesmo tempo
 - * próximo cliente pode escolher o caixa
- **caminhar** de um indivíduo: perna esquerda ou direita

◆ Linguagens Formais

- seqüencial e não-determinismo: especialmente importantes

◆ Semântica do não-determinismo adotada

- a usual para Linguagens Formais, Teoria da Computação...
 - * não-determinismo interno
 - * objetivo: determinar a capacidade de reconhecer linguagens e de solucionar problemas
- difere da adotada no estudo dos Modelos para Concorrência
 - * exemplo: Sistemas Operacionais
 - * pode causar confusão com a semântica da concorrência

3 – Linguagens Regulares

- 3.1 Sistema de Estados Finitos
- 3.2 Composição Seqüencial, Concorrente e Não-Determinista
- 3.3 Autômato Finito**
- 3.4 Autômato Finito Não-Determinístico
- 3.5 Autômato Finito com Movimentos Vazios
- 3.6 Expressão Regular
- 3.7 Gramática Regular

3.3 Autômato Finito

◆ Autômato Finito: sistema de estados finitos

- número finito e *predefinido* de estados
- modelo computacional comum em diversos estudos teórico-formais
 - * Linguagens Formais
 - * Compiladores
 - * Semântica Formal
 - * Modelos para Concorrência

◆ Formalismo operacional/reconhecedor - pode ser

- **determinístico**
 - * dependendo do estado corrente e do símbolo lido
 - * pode assumir um *único* estado
- **não-determinístico**
 - * dependendo do estado corrente e do símbolo lido
 - * pode assumir um *conjunto* de estados alternativos
- **com movimentos vazios**
 - * dependendo do estado corrente e *sem ler* qualquer símbolo
 - * pode assumir um *conjunto* de estados (portanto é não-determinístico)

◆ Movimento vazio

- pode ser visto como **transições encapsuladas**
 - * excetuando-se por uma eventual mudança de estado
 - * nada mais pode ser observado
- análogo à **encapsulação** das **linguagens orientadas a objetos**

◆ Três tipos de autômatos: equivalentes

- em termos de poder computacional

◆ Autômato finito (determinístico): máquina constituída por

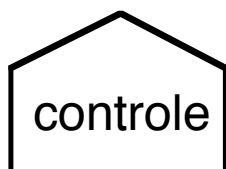
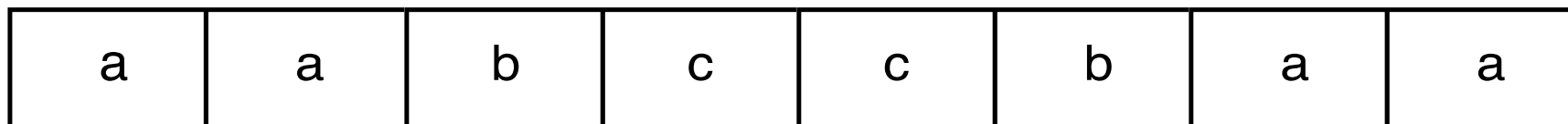
- **Fita**: dispositivo de entrada
 - * contém informação a ser processada
- **Unidade de Controle**: reflete o estado corrente da máquina
 - * possui unidade de leitura (**cabeça da fita**)
 - * acessa uma célula da fita de cada vez
 - * **movimenta-se** exclusivamente para a direita
- **Programa, Função Programa** ou **Função de Transição**
 - * **comanda** as leituras
 - * **define** o estado da máquina

◆ Fita é finita

- dividida em células
- cada célula armazena um símbolo
- símbolos pertencem a um alfabeto de entrada
- não é possível gravar sobre a fita (não existe memória auxiliar)
- palavra a ser processada ocupa toda a fita

◆ Unidade de controle

- número finito e predefinido de estados
 - * origem do termo **controle finito**
- leitura
 - * lê o símbolo de uma célula de cada vez
 - * move a cabeça da fita uma célula para a direita
 - * posição inicial da cabeça célula mais à esquerda da fita



◆ Programa: função parcial

- dependendo do estado corrente e do símbolo lido
- determina o novo estado do autômato

Def: Autômato Finito (Determinístico) ou AFD

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

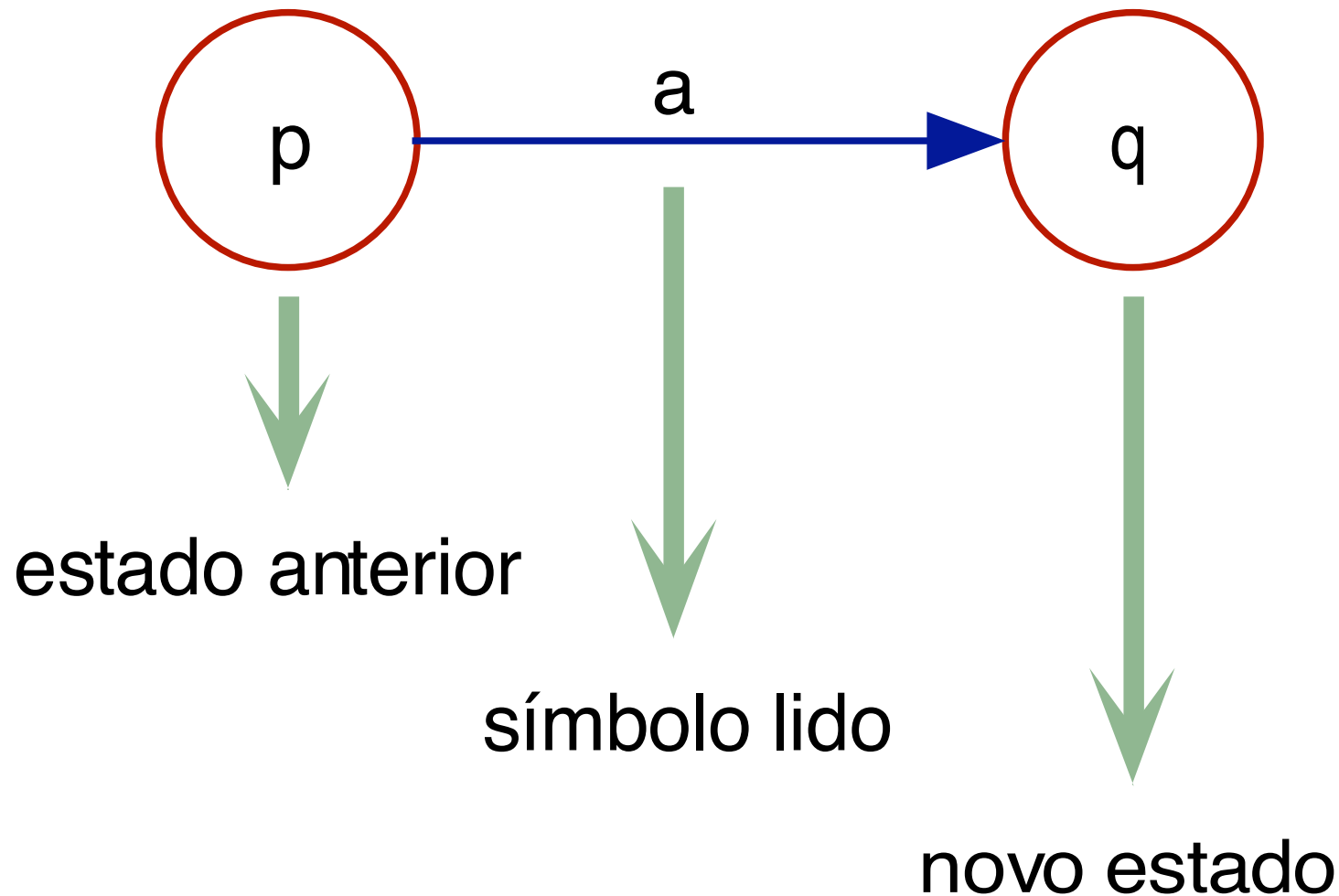
- Σ é um alfabeto (de símbolos) de entrada
- Q é um conjunto de estados possíveis do autômato (finito)
- δ é uma (função) programa ou função de transição (função parcial)

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

* transição do autômato: $\delta(p, a) = q$

- q_0 é um elemento distinguido de Q : estado inicial
- F é um subconjunto de Q : conjunto de estados finais

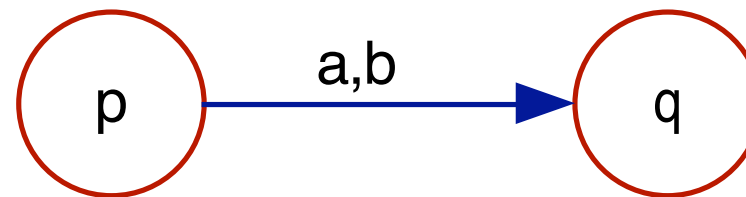
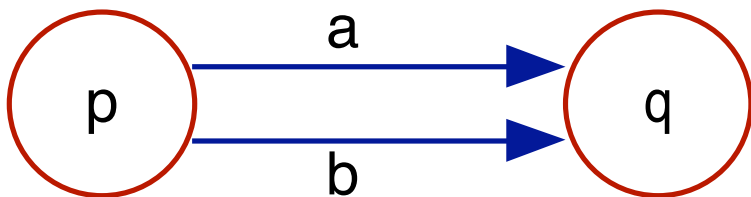
◆ **Autômato finito como um diagrama:** $\delta(p, a) = q$



◆ Estados iniciais e finais



◆ Transições paralelas: $\delta(q, a) = p$ e $\delta(q, b) = p$



◆ Função programa como uma tabela de dupla entrada

$$\delta(p, a) = q$$

δ	a	...
p	q	...
q

◆ **Computação de um autômato finito**

- sucessiva aplicação da função programa
- para cada símbolo da entrada (da esquerda para a direita)
- até ocorrer uma condição de parada

◆ **Lembre-se que um autômato finito**

- não possui memória de trabalho
- para armazenar as informações passadas
- deve-se usar o conceito de estado

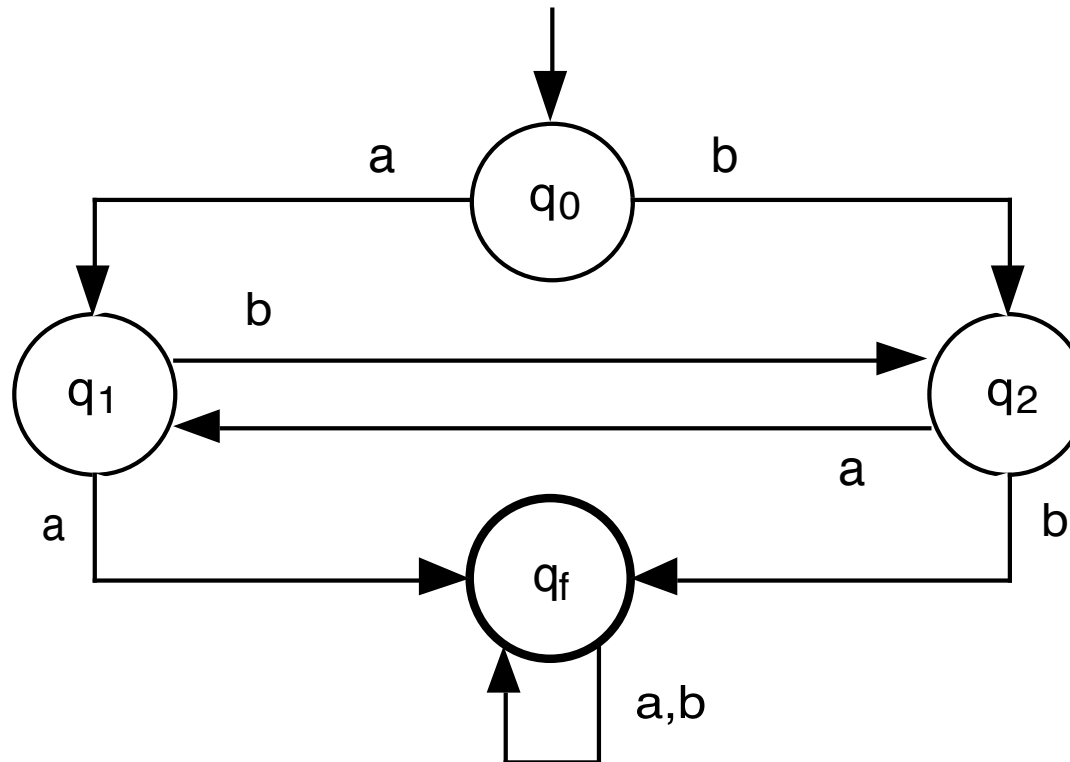
Exp: Autômato Finito: aa ou bb como subpalavra

$$L_1 = \{ w \mid w \text{ possui aa ou bb como subpalavra} \}$$

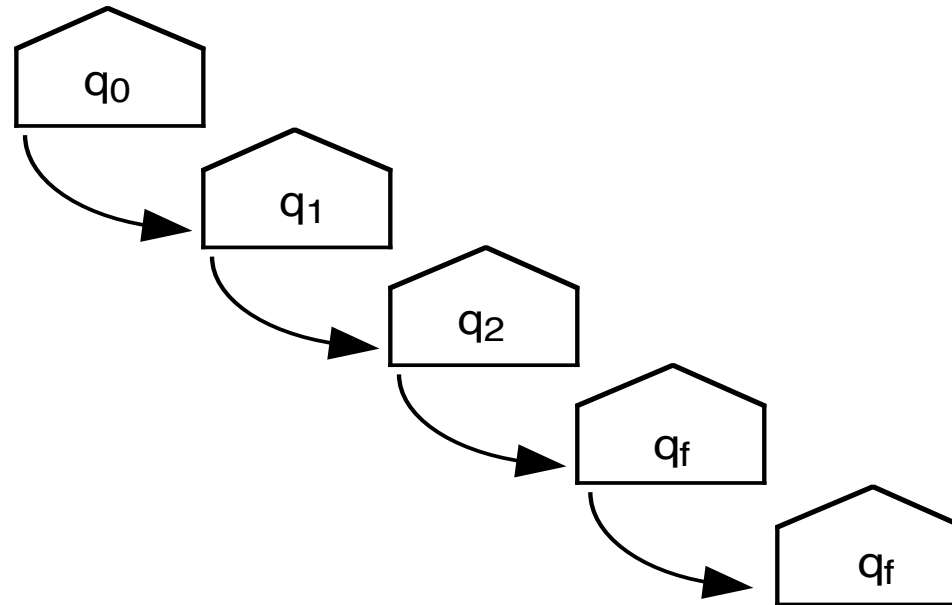
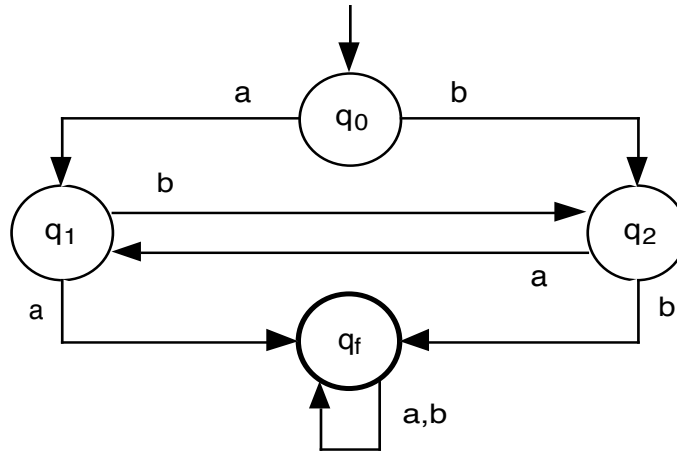
Autômato finito

$$M_1 = (\{ a, b \}, \{ q_0, q_1, q_2, q_f \}, \delta_1, q_0, \{ q_f \})$$

δ_1	a	b
q ₀	q ₁	q ₂
q ₁	q _f	q ₂
q ₂	q ₁	q _f
q _f	q _f	q _f



- q_1 : "símbolo anterior é a"
- q_2 : "símbolo anterior é b"
- qual a informação memorizada por q_0 e q_f
- após identificar aa ou bb
 - * q_f (final): varre o sufixo da entrada - terminar o processamento



Obs: Autômato Finito Sempre Pára

Como

- qualquer **palavra** é **finita**
- novo **símbolo** é **lido** a **cada aplicação** da função **programa**

não existe a possibilidade de **ciclo** (*loop*) **infinito**

◆ Parada do processamento

- **Aceita** a entrada
 - * após processar o **último símbolo**, assume um **estado final**
- **Rejeita** a entrada. Duas possibilidades
 - * após processar o **último símbolo**, assume um **estado não-final**
 - * programa indefinido para argumento (estado e símbolo)

Obs: Autômato Finito × Grafo Finito Direto

Qual a diferença entre um autômato finito e um grafo finito direto?

Qualquer autômato finito pode ser visto como um grafo finito direto

- podem existir arcos paralelos (mesmos nodos origem e destino)
- dois ou mais arcos podem ser identificados com a mesma etiqueta (símbolo do alfabeto)
- existe um nodo distinguido: estado inicial
- existe um conjunto de nodos distinguidos: estados finais

Usual considerar um autômato finito como grafo finito direto especial

- herda resultados da Teoria dos Grafos

◆ Definição formal do comportamento de um autômato finito

- dar **semântica** à sintaxe
- necessário **estender** a função **programa**
- **argumento**: **estado** e *palavra*

Def: Função Programa Estendida, Computação

$M = (\Sigma, Q, \delta, q_0, F)$ autômato finito determinístico

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

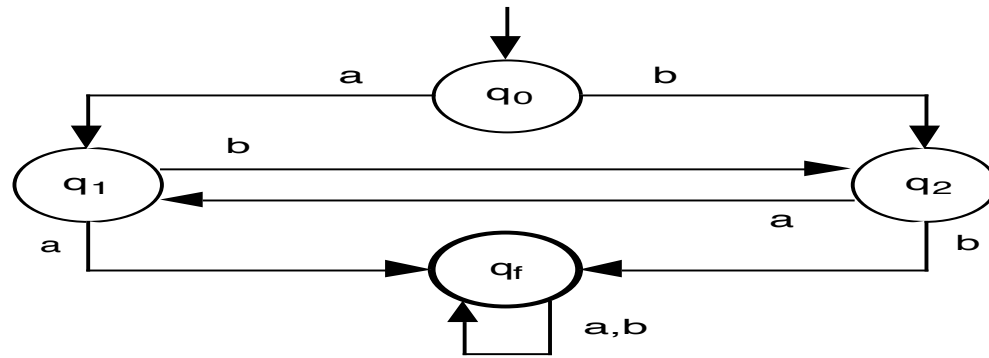
é $\delta: Q \times \Sigma \rightarrow Q$ estendida para palavras - indutivamente definida

- $\delta^*(q, \varepsilon) = q$
- $\delta^*(q, aw) = \delta^*(\delta(q, a), w)$

◆ Observe

- **sucessiva aplicação** da função **programa**
 - * para cada símbolo da palavra
 - * a partir de um dado estado
- se a **entrada** for **vazia**, fica **parado**
- **aceita/rejeita**: **função programa estendida** a partir do **estado inicial**

Exp: Função Programa Estendida



- $\underline{\delta}^*(q_0, abaa) =$ função estendida sobre **abaa**
- $\underline{\delta}^*(\delta(q_0, a), baa) =$ processa **abaa**
- $\underline{\delta}^*(q_1, baa) =$ função estendida sobre **baa**
- $\underline{\delta}^*(\delta(q_1, b), aa) =$ processa **baa**
- $\underline{\delta}^*(q_2, aa) =$ função estendida sobre **aa**
- $\underline{\delta}^*(\delta(q_2, a), a) =$ processa **aa**
- $\underline{\delta}^*(q_1, a) =$ função estendida sobre **a**
- $\underline{\delta}^*(\delta(q_1, a), \epsilon) =$ processa **a**
- $\underline{\delta}^*(q_f, \epsilon) = q_f$ função estendida sobre ϵ : fim da indução; **ACEITA**

Def: Linguagem Aceita, Linguagem Rejeitada

$M = (\Sigma, Q, \delta, q_0, F)$ autômato finito determinístico.

Linguagem Aceita ou Linguagem Reconhecida por M

$$L(M) = \text{ACEITA}(M) = \{ w \mid \delta^*(q_0, w) \in F \}$$

Linguagem Rejeitada por M :

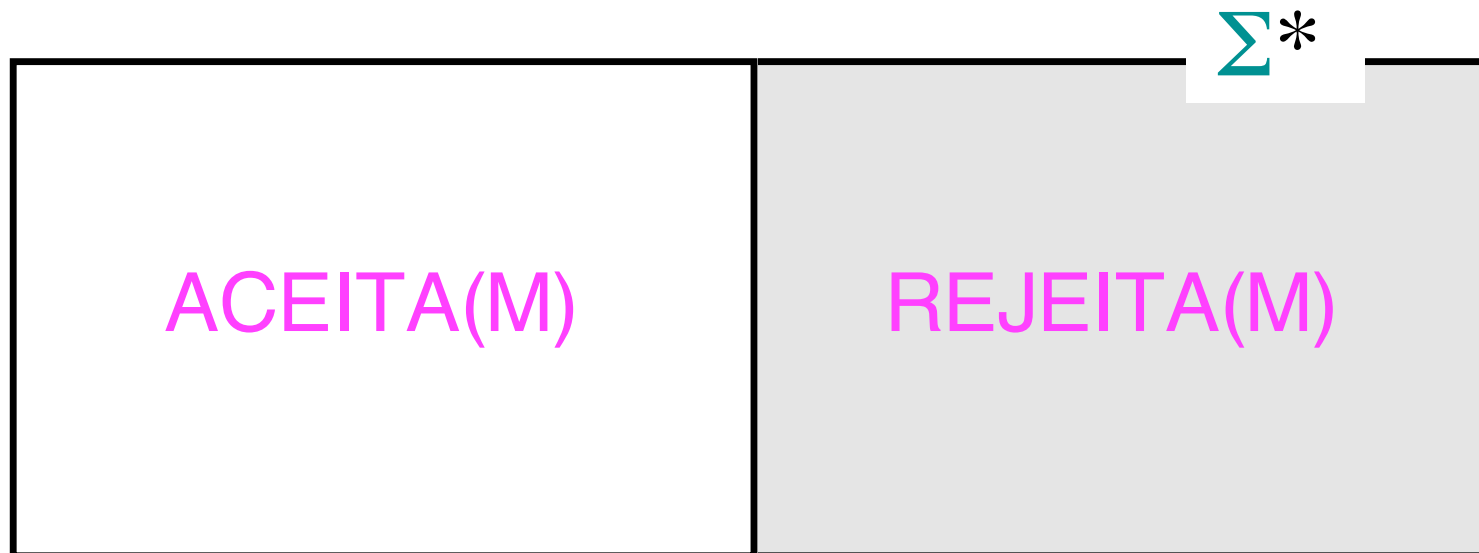
$$\text{REJEITA}(M) = \{ w \mid \delta^*(q_0, w) \notin F \text{ ou } \delta^*(q_0, w) \text{ é indefinida} \}$$

◆ Supondo que Σ^* é o conjunto universo

- $\text{ACEITA}(M) \cap \text{REJEITA}(M) = \emptyset$
- $\text{ACEITA}(M) \cup \text{REJEITA}(M) = \Sigma^*$
- $\sim \text{ACEITA}(M) = \text{REJEITA}(M)$
- $\sim \text{REJEITA}(M) = \text{ACEITA}(M)$

◆ Cada autômato finito M sobre Σ

- induz uma partição de Σ^* em duas classes de equivalência
- e se um dos dois conjuntos for vazio?



- ◆ **Diferentes autômatos finitos podem aceitar uma mesma linguagem**

Def: Autômatos Finitos Equivalentes

M_1 e M_2 são **Autômatos Finitos Equivalentes** se e somente se

$$ACEITA(M_1) = ACEITA(M_2)$$

Def: Linguagem Regular, Linguagem Tipo 3

L é uma **Linguagem Regular** ou **Linguagem Tipo 3**

- **existe** pelo menos um autômato finito determinístico que **aceita** L

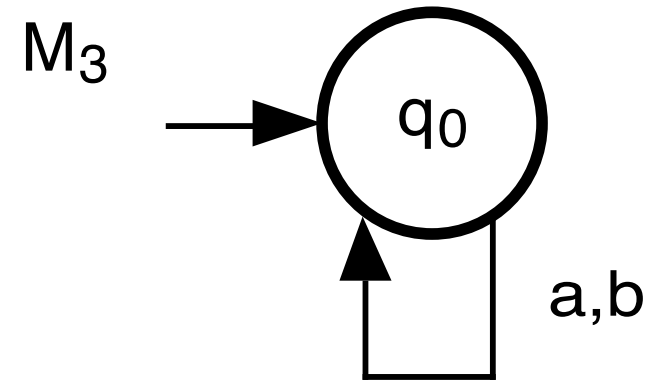
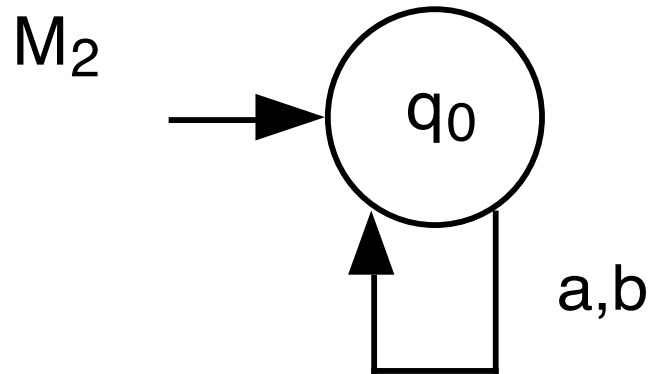
Exp: ...Autômato Finito: Vazia, Todas as Palavras

Linguagens sobre o alfabeto $\{a, b\}$

$$L_2 = \emptyset$$

ϵ

$$L_3 = \Sigma^*$$



Exp: Autômato Finito: Vazia, Todas as Palavras

$$L_2 = \emptyset \quad \text{e} \quad L_3 = \Sigma^*$$

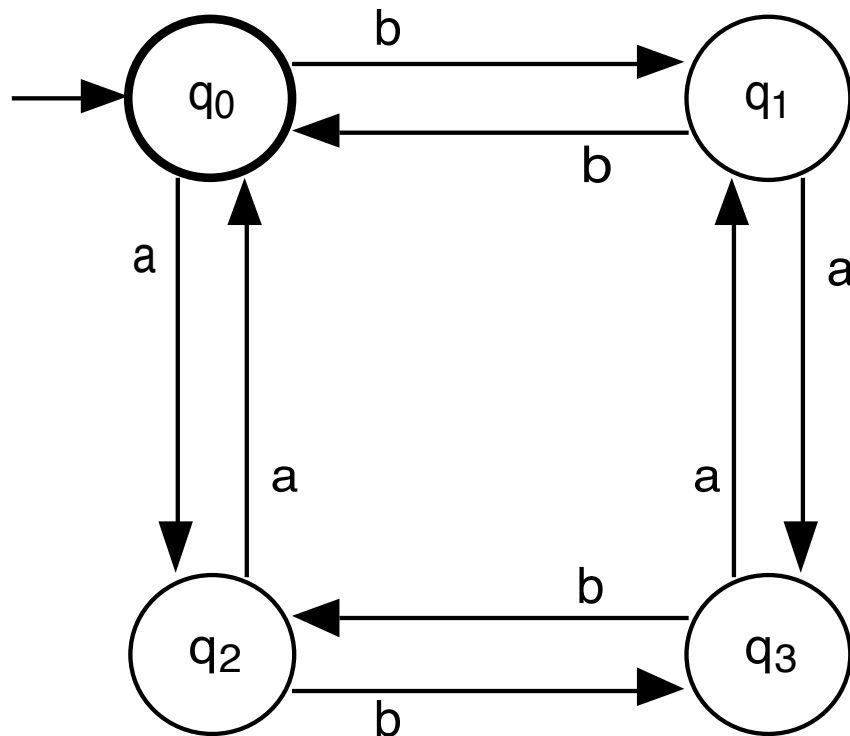
δ_2	a	b
q_0	q_0	q_0

δ_3	a	b
q_0	q_0	q_0

- diferença entre δ_2 e δ_3 ?
- o que, exatamente, diferencia M_2 de M_3 ?

Exp: Autômato Finito: número par de cada símbolo

$L_4 = \{ w \mid w \text{ possui um número par de } a \text{ e um número par de } b \}$



Como seria para aceitar um número ímpar de cada símbolo?

Obs: Função Programa × Função Programa Estendida

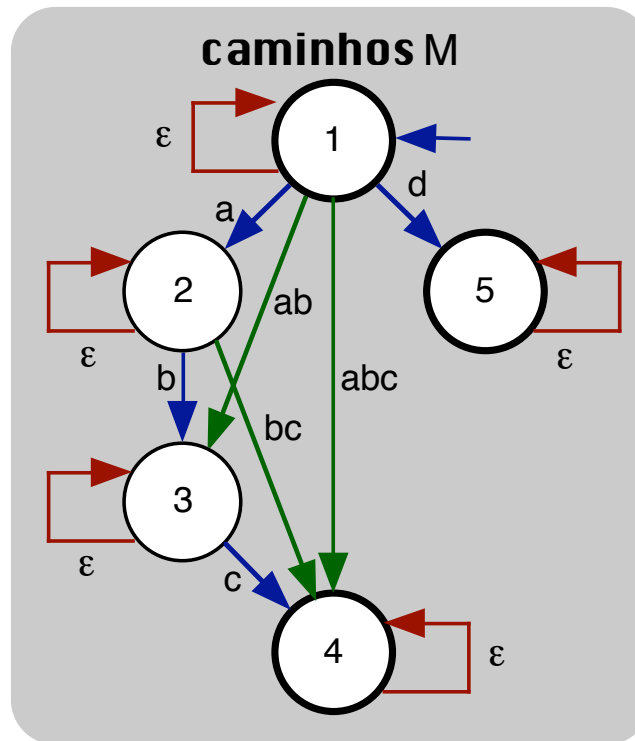
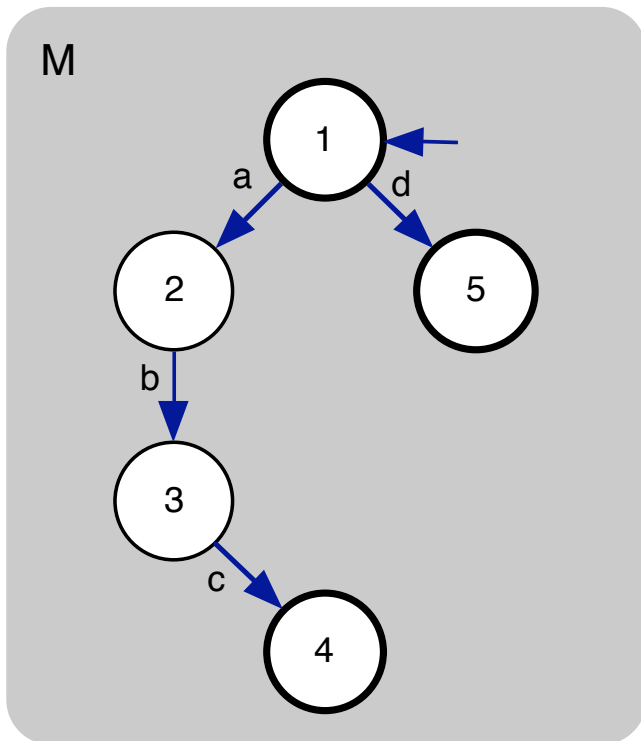
Objetivando simplificar a notação

- δ e a sua correspondente extensão $\underline{\delta}^*$
- podem ser ambas denotadas por δ

Obs: Computações × Caminhos de um Grafo

- conjunto de **arcos**: **computações** possíveis
- subconjunto de **arcos**
 - * com **origem** no estado **inicial**
 - * **destino** em algum estado **final**
 - * linguagem **aceita**

Obs: ...Computações × Caminhos de um Grafo

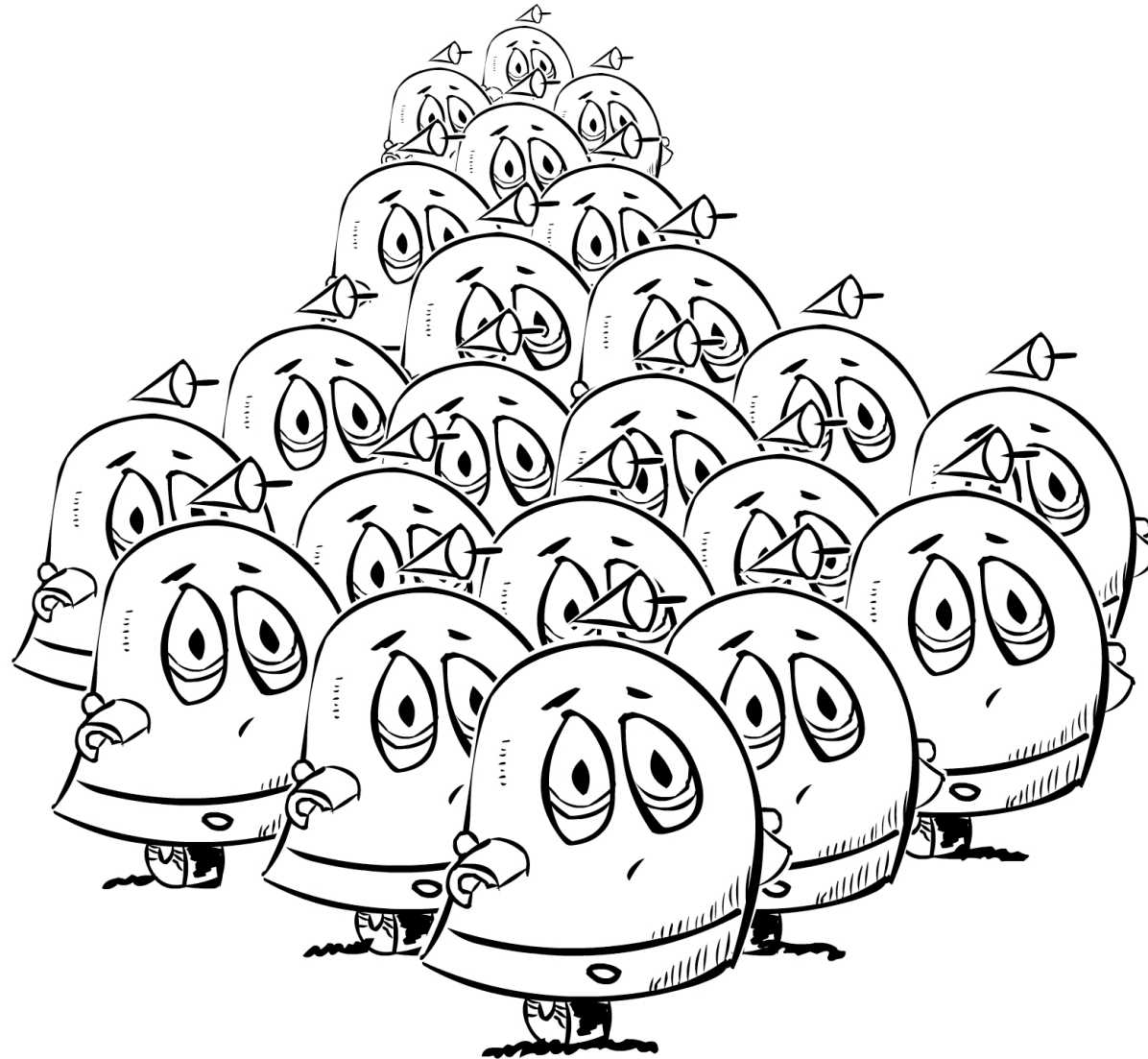


Computações(M)
=
{ ϵ , a, b, c, d,
ab, bc, abc}

ACEITA (M)
=
{ ϵ , d, abc}

3 – Linguagens Regulares

- 3.1 Sistema de Estados Finitos
- 3.2 Composição Seqüencial, Concorrente e Não-Determinista
- 3.3 Autômato Finito
- 3.4 Autômato Finito Não-Determinístico
- 3.5 Autômato Finito com Movimentos Vazios
- 3.6 Expressão Regular
- 3.7 Gramática Regular



3.4 Autômato Finito Não-Determinístico

◆ Não-determinismo

- importante generalização dos modelos de máquinas
- fundamental no estudo
 - * Modelos para Concorrência
 - * Teoria da Computação
 - * Linguagens Formais, ...

◆ Semântica de não-determinismo adotada

- usual no estudo das Linguagens Formais
- objetiva determinar a capacidade de
 - * reconhecer linguagens
 - * solucionar problemas
- pode causar alguma confusão com semântica da concorrência

◆ Nem sempre não-determinismo aumenta o poder

- reconhecimento de linguagens de uma classe de autômatos
 - * qualquer autômato finito não-determinístico pode ser simulado por um autômato finito determinístico

◆ Não-determinismo no programa, é uma função parcial

*dependendo do estado corrente e do símbolo lido,
determina um conjunto de estados do autômato.*

◆ Assume um conjunto de estados alternativos

- como uma multiplicação da unidade de controle
- uma para cada alternativa
- processando independentemente
- sem compartilhar recursos

Def: Autômato Finito Não-Determinístico (AFN)

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

- Σ - alfabeto (de símbolos) de entrada
- Q - conjunto de estados possíveis (finito)
- δ - (função) programa ou função de transição (função parcial)

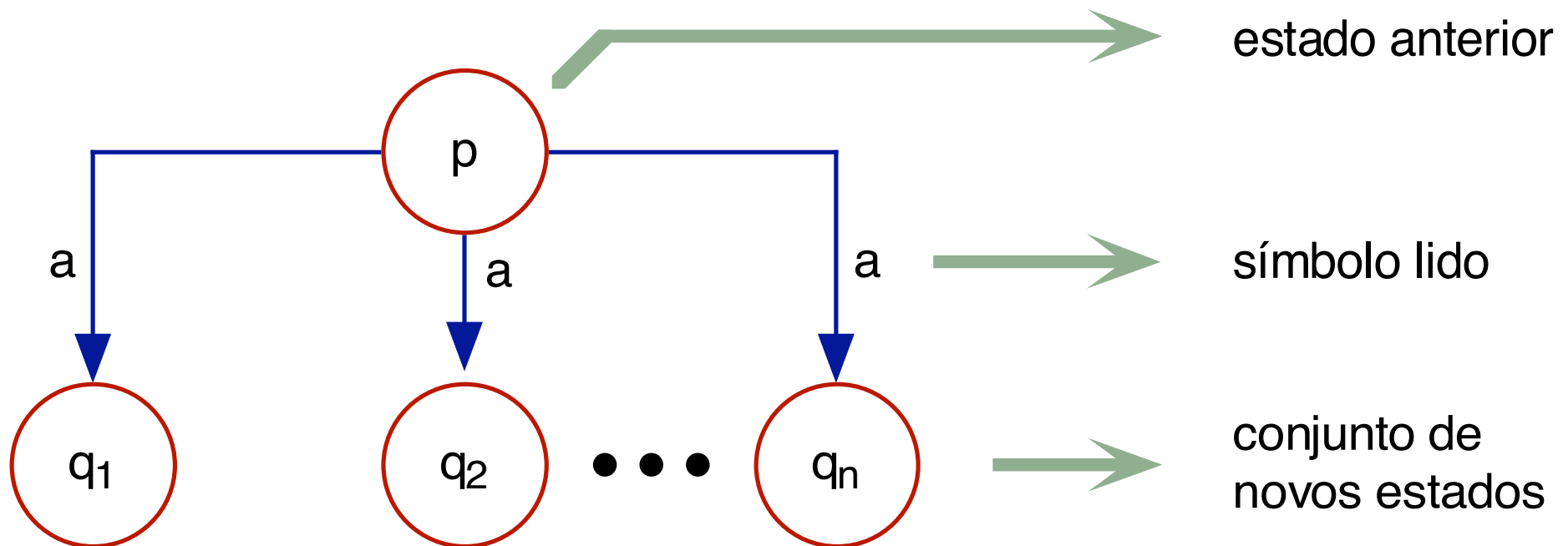
$$\delta: Q \times \Sigma \rightarrow 2^Q$$

* transição: $\delta(p, a) = \{ q_1, q_2, \dots, q_n \}$

- q_0 é um elemento distinguido de Q : estado inicial
- F é um subconjunto de Q : conjunto de estados finais

◆ Autômato como diagrama

$$\delta(p, a) = \{ q_1, q_2, \dots, q_n \}$$



◆ **Computação de um autômato finito não-determinístico**

- sucessiva aplicação da função programa
- para cada símbolo da entrada (da esquerda para a direita)
- até ocorrer uma condição de parada

◆ **Argumentos: computação/função programa estendida**

- conjunto finito de estados e uma palavra

Def: Função Programa Estendida, Computação

$M = (\Sigma, Q, \delta, q_0, F)$ autômato finito não-determinístico

$$\underline{\delta}^*: 2^Q \times \Sigma^* \rightarrow 2^Q$$

indutivamente definida

- $\underline{\delta}^*(P, \varepsilon) = P$
- $\underline{\delta}^*(P, aw) = \underline{\delta}^*(\bigcup_{q \in P} \delta(q, a), w)$

◆ Transição estendida (a um conjunto de estados)

$$\underline{\delta}^*({q_1, q_2, \dots, q_n}, a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \dots \cup \delta(q_n, a)$$

◆ Parada do processamento

- **Aceita** a entrada
 - * após **processar** o **último símbolo** da fita, **existe** pelo menos um **estado final** pertencente ao conjunto de **estados alternativos** atingidos
- **Rejeita** a entrada. Duas possibilidades
 - * após processar o **último símbolo** da fita, **todos** os **estados alternativos** atingidos são **não-finais**
 - * **programa indefinido** para o argumento (conjunto de estados e símbolo)

Def: Linguagem Aceita, Linguagem Rejeitada

Seja $M = (\Sigma, Q, \delta, q_0, F)$ um autômato finito não-determinístico

Linguagem Aceita ou Linguagem Reconhecida por M

$$L(M) = \text{ACEITA}(M) = \{ w \mid \delta^*(\{ q_0 \}, w) \cap F \neq \emptyset \}$$

Linguagem Rejeitada por M

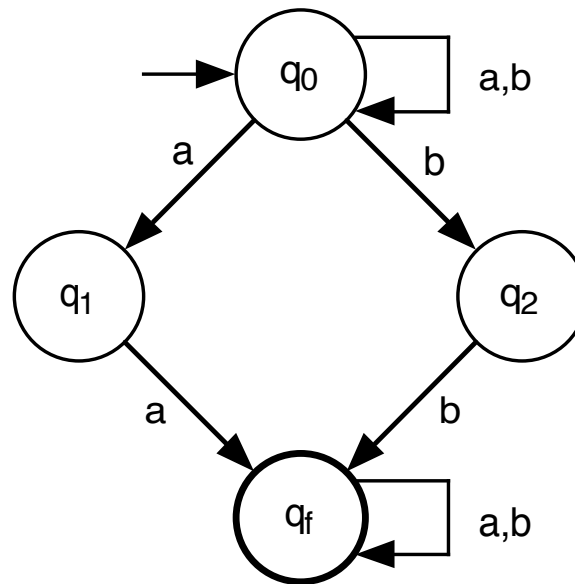
$$\text{REJEITA}(M) = \{ w \mid \delta^*(\{ q_0 \}, w) \cap F = \emptyset \text{ ou } \delta^*(\{ q_0 \}, w) \text{ é indefinida} \}$$

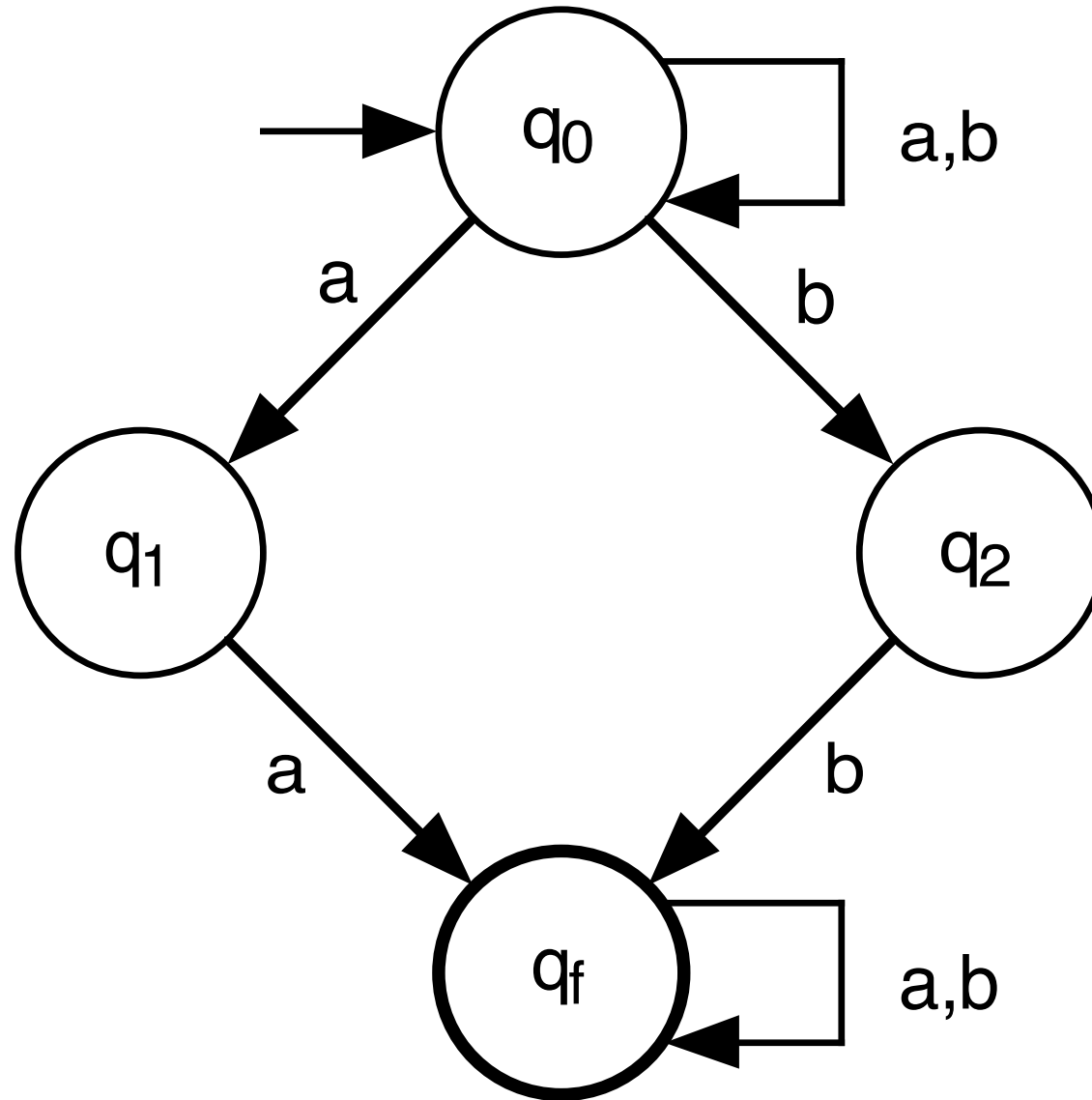
Exp: Autômato Finito Não-Determinístico: aa ou bb como subpalavra

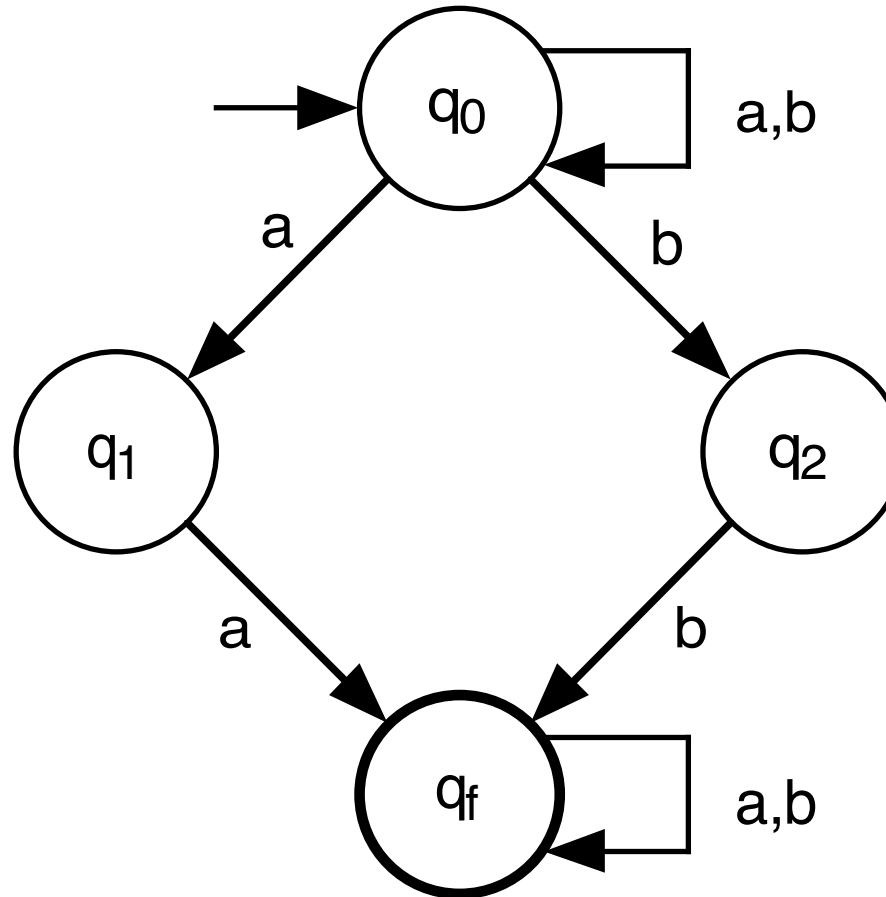
$$L_5 = \{ w \mid w \text{ possui aa ou bb como subpalavra} \}$$

Autômato finito não-determinístico:

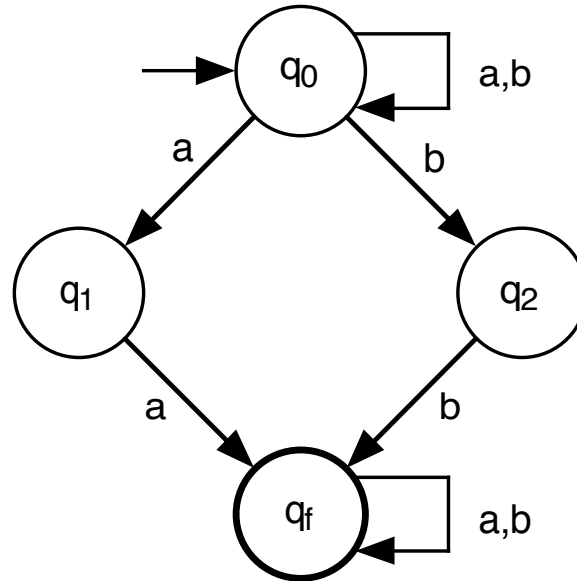
$$M_5 = (\{ a, b \}, \{ q_0, q_1, q_2, q_f \}, \delta_5, q_0, \{ q_f \})$$







- o ciclo em q_0 realiza uma **varredura** em toda a entrada
- o caminho $q_0/q_1/q_f$ **garante** a ocorrência de **aa**
- o caminho $q_0/q_2/q_f$ **garante** a ocorrência de **bb**



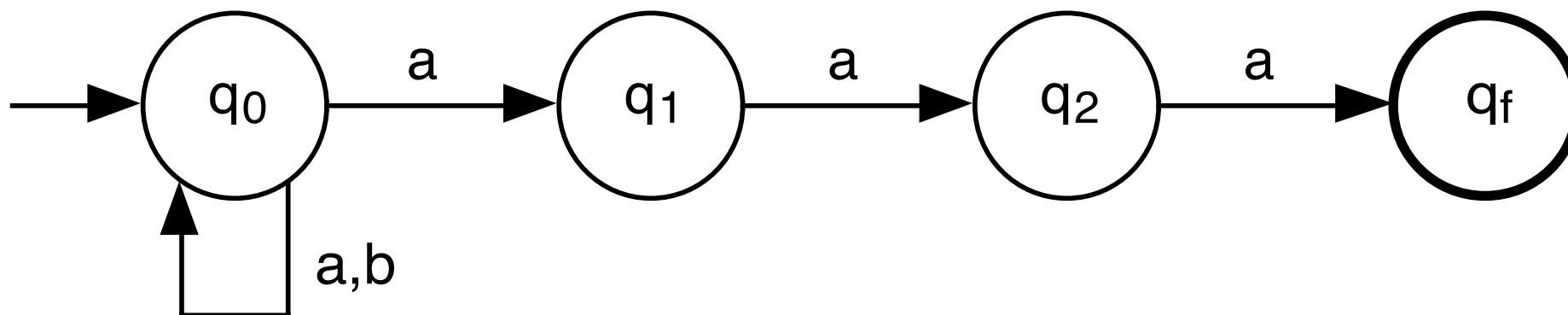
δ_5	a	b
q0	{ q0, q1 }	{ q0, q2 }
q1	{ qf }	-
q2	-	{ qf }
qf	{ qf }	{ qf }

Exp: AFN: aaa como sufixo

$$L_6 = \{ w \mid w \text{ possui } aaa \text{ como sufixo} \}$$

Autômato finito não-determinístico:

$$M_6 = (\{ a, b \}, \{ q_0, q_1, q_2, q_f \}, \delta_6, q_0, \{ q_f \})$$



◆ Não-determinismo

- **aparentemente**, um **significativo acréscimo** ao poder computacional autômato finito
- **na realidade** *não* aumenta seu poder computacional

Teorema: Equivalência entre AFD e AFN

Classe dos Autômatos Finitos **Determinísticos** é **equivalente** à

Classe dos Autômatos Finitos **Não-Determinísticos**

Prova: (por indução)

Mostrar que

- a partir de um AFN M qualquer
- construir um AFD M_D que realiza as mesmas computações
- M_D simula M

AFN \rightarrow AFD

- estados de M_D simulam combinações de estados alternativos de M
- prova da simulação: por indução

AFD \rightarrow AFN

- não necessita ser mostrado: decorre trivialmente das definições

$M = (\Sigma, Q, \delta, q_0, F)$ um AFN qualquer. AFD construído

$$M_D = (\Sigma, Q_D, \delta_D, \langle q_0 \rangle, F_D)$$

- Q_D - todas as combinações, sem repetições, de estados de Q
 - * notação $\langle q_1 q_2 \dots q_n \rangle$
 - * ordem não distingue combinações: $\langle q_u q_v \rangle = \langle q_v q_u \rangle$
 - * imagem de todos os estados alternativos de M
- $\delta_D: Q_D \times \Sigma \rightarrow Q_D$

$$\delta_D(\langle q_1 \dots q_n \rangle, a) = \langle p_1 \dots p_m \rangle \quad \text{sse} \quad \delta^*(\{q_1, \dots, q_n\}, a) = \{p_1, \dots, p_m\}$$

- $\langle q_0 \rangle$ - estado inicial
- F_D - conjunto de estados $\langle q_1 q_2 \dots q_n \rangle$ pertencentes a Q_D
 - * alguma componente q_i pertence a F , para i em $\{1, 2, \dots, n\}$

AFD M_D simula as computações do AFN M ???

- indução no tamanho da palavra
- mostrar que

$$\delta_D^*(\langle q_0 \rangle, w) = \langle q_1 \dots q_u \rangle \quad \text{sse} \quad \delta^*(\{ q_0 \}, w) = \{ q_1, \dots, q_u \}$$

Base de indução. $|w| = 0$. Portanto $w = \varepsilon$:

$$\delta_D^*(\langle q_0 \rangle, \varepsilon) = \langle q_0 \rangle \quad \text{se e somente se} \quad \delta^*(\{ q_0 \}, \varepsilon) = \{ q_0 \}$$

- verdadeiro, por definição de computação

Hipótese de indução. $|w| = n$ e $n \geq 1$. Suponha que:

$$\delta_D^*(\langle q_0 \rangle, w) = \langle q_1 \dots q_u \rangle \quad \text{sse} \quad \delta^*(\{q_0\}, w) = \{q_1, \dots, q_u\}$$

Passo de Indução. $|wa| = n + 1$ e $n \geq 1$

$$\delta_D^*(\langle q_0 \rangle, wa) = \langle p_1 \dots p_v \rangle \quad \text{sse} \quad \delta^*(\{q_0\}, wa) = \{p_1, \dots, p_v\}$$

- equivale (hipótese de indução)

$$\delta_D(\langle q_1 \dots q_u \rangle, a) = \langle p_1 \dots p_v \rangle \quad \text{sse} \quad \delta^*(\{q_1, \dots, q_u\}, a) = \{p_1, \dots, p_v\}$$

- verdadeiro, por definição de δ_D

Logo, M_D simula M para qualquer entrada w pertencente a Σ^*

◆ Portanto, linguagem aceita por AFN

- é Linguagem Regular ou Tipo 3

Obs: Determinismo × Não-Determinismo

Muitas vezes é mais fácil desenvolver um AFN do que um AFD

- exemplo

$\{ w \mid \text{o quinto símbolo da direita para a esquerda de } w \text{ é } a \}$

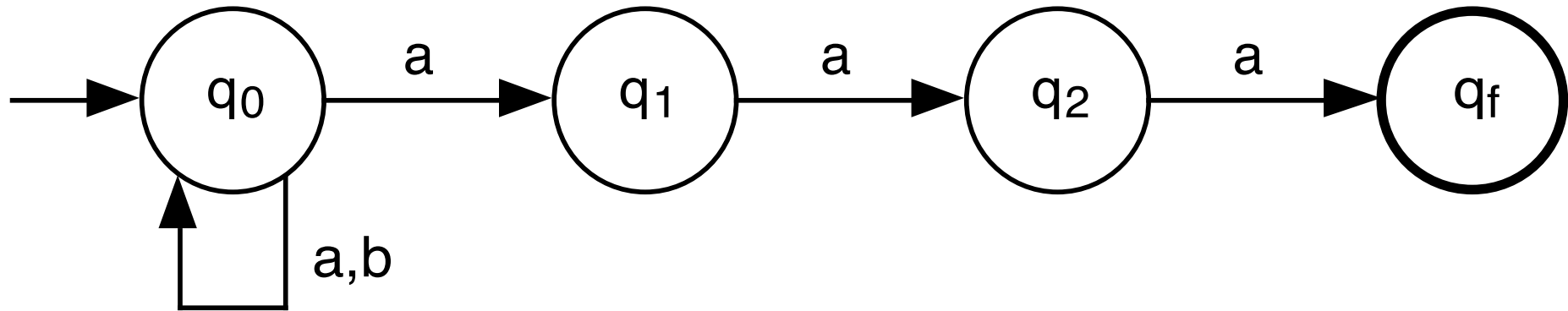
- solução determinista: não é trivial; número grande de estados
- solução não-determinista: bem simples; poucos estados

Alternativa para construir um AFD

- desenvolver inicialmente AFN
- aplicar o algoritmo apresentado na prova do teorema

Exp: AFN \rightarrow AFD

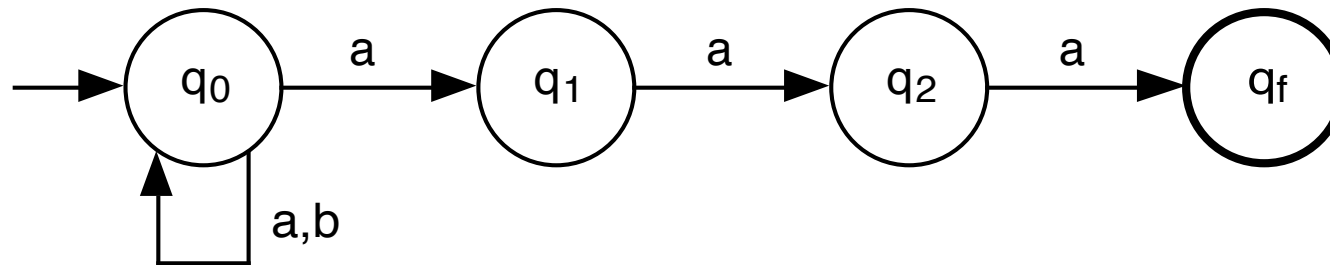
$$M_6 = (\{ a, b \}, \{ q_0, q_1, q_2, q_f \}, \delta_6, q_0, \{ q_f \})$$



$$M_{6D} = (\{ a, b \}, Q_D, \delta_{6D}, \langle q_0 \rangle, F_D)$$

- $Q_D = \{ \langle q_0 \rangle, \langle q_1 \rangle, \langle q_2 \rangle, \langle q_f \rangle, \langle q_0q_1 \rangle, \langle q_0q_2 \rangle, \dots, \langle q_0q_1q_2q_f \rangle \}$
- $F_D = \{ \langle q_f \rangle, \langle q_0q_f \rangle, \langle q_1q_f \rangle, \dots, \langle q_0q_1q_2q_f \rangle \}$

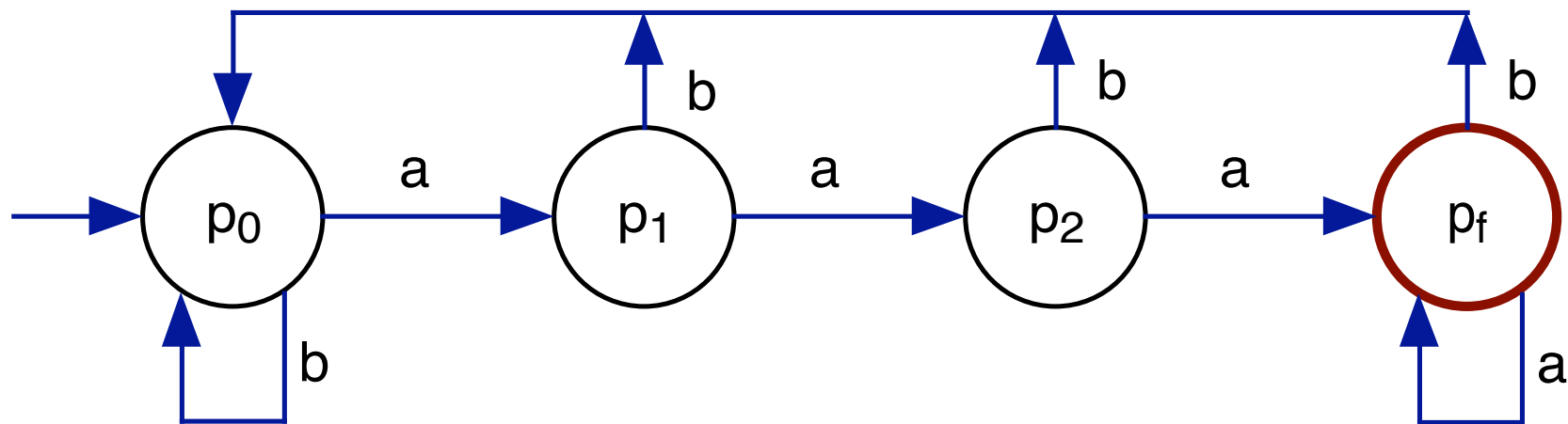
AFN



AFD

δ_{6D}	a	b
$\langle q_0 \rangle$	$\langle q_0 q_1 \rangle$	$\langle q_0 \rangle$
$\langle q_0 q_1 \rangle$	$\langle q_0 q_1 q_2 \rangle$	$\langle q_0 \rangle$
$\langle q_0 q_1 q_2 \rangle$	$\langle q_0 q_1 q_2 q_f \rangle$	$\langle q_0 \rangle$
$\langle q_0 q_1 q_2 q_f \rangle$	$\langle q_0 q_1 q_2 q_f \rangle$	$\langle q_0 \rangle$

δ_{6D}	a	b
$p_0 = \langle q_0 \rangle$	$\langle q_0 q_1 \rangle$	$\langle q_0 \rangle$
$p_1 = \langle q_0 q_1 \rangle$	$\langle q_0 q_1 q_2 \rangle$	$\langle q_0 \rangle$
$p_2 = \langle q_0 q_1 q_2 \rangle$	$\langle q_0 q_1 q_2 q_f \rangle$	$\langle q_0 \rangle$
$p_f = \langle q_0 q_1 q_2 q_f \rangle$	$\langle q_0 q_1 q_2 q_f \rangle$	$\langle q_0 \rangle$



3 – Linguagens Regulares

- 3.1 Sistema de Estados Finitos
- 3.2 Composição Seqüencial, Concorrente e Não-Determinista
- 3.3 Autômato Finito
- 3.4 Autômato Finito Não-Determinístico
- 3.5 Autômato Finito com Movimentos Vazios
- 3.6 Expressão Regular
- 3.7 Gramática Regular

3.5 Autômato Finito com Movimentos Vazios

◆ Movimentos vazios

- generalizam os movimentos não-determinísticos

◆ Movimento vazio

- transição sem leitura de símbolo algum da fita
- interpretado como um não-determinismo interno ao autômato
 - * transição encapsulada
 - * excetuando-se por uma eventual mudança de estados
 - * nada mais pode ser observado

◆ Algumas vantagens

- facilita algumas construções e demonstrações

◆ Poder computacional p/ autômatos finitos

- *não* aumenta o poder de reconhecimento de linguagens
- qualquer AFN_ϵ pode ser simulado por um AFD

Def: Autômato Finito com Movimentos Vazios - AFN_ϵ

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

- Σ - alfabeto (de símbolos) de entrada
- Q - conjunto de estados possíveis
- δ - (função) programa ou função de transição (função parcial)

$$\delta: Q \times (\Sigma \cup \{ \epsilon \}) \rightarrow 2^Q$$

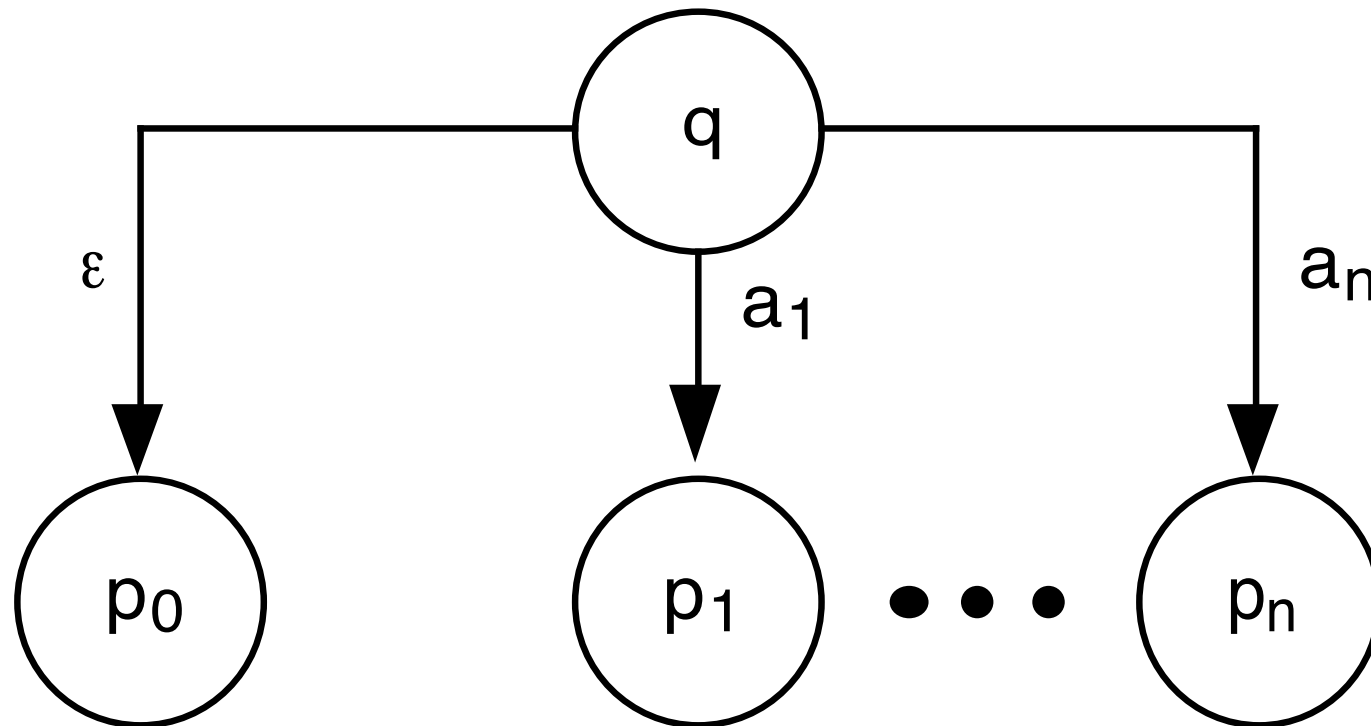
* movimento vazio ou transição vazia

$$\delta(p, \epsilon) = \{ q_1, q_2, \dots, q_n \}$$

- q_0 - elemento distinguido de Q : estado inicial
- F - subconjunto de Q : conjunto de estados finais

◆ Autômato como diagrama

$$\delta(q, \varepsilon) = \{ p_0 \} \quad \delta(q, a_1) = \{ p_1 \} \quad \dots \quad \delta(q, a_n) = \{ p_n \}$$



◆ **Computação de um AFN_ϵ**

- análoga à de um AFN

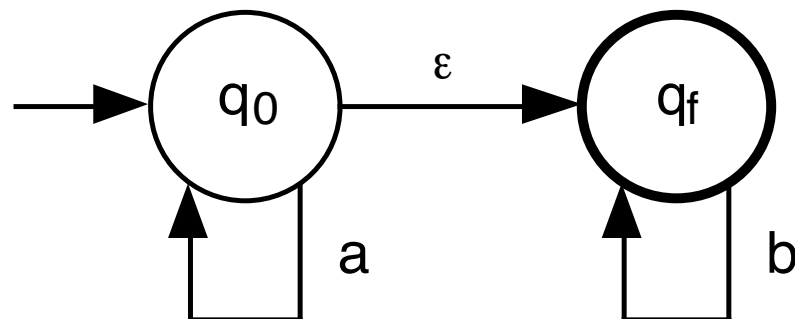
◆ **Processamento de uma transição vazia**

- não-determinístico
- assume **simultaneamente** os estados **destino** e **origem**
- origem de um movimento vazio: caminho alternativo

Exp: AFN_{ϵ} : a's antecedem b's

$$M_7 = (\{ a, b \}, \{ q_0, q_f \}, \delta_7, q_0, \{ q_f \})$$

δ_7	a	b	ϵ
q_0	{ q_0 }	-	{ q_f }
q_f	-	{ q_f }	



◆ Antes de definir computação

- computação de transições vazias a partir de
 - * um estado
 - * um conjunto finito de estados

Def: Computação Vazia

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

Computação Vazia ou Função Fecho Vazio (*um estado*)

$$\delta_\varepsilon: Q \rightarrow 2^Q$$

- indutivamente definida
 - * $\delta_\varepsilon(q) = \{q\}$, se $\delta(q, \varepsilon)$ é indefinida
 - * $\delta_\varepsilon(q) = \{q\} \cup \delta(q, \varepsilon) \cup (\bigcup_{p \in \delta(q, \varepsilon)} \delta_\varepsilon(p))$, caso contrário

Computação Vazia ou Função Fecho Vazio (*conjunto de estados*)

$$\delta_\varepsilon^*: 2^Q \rightarrow 2^Q$$

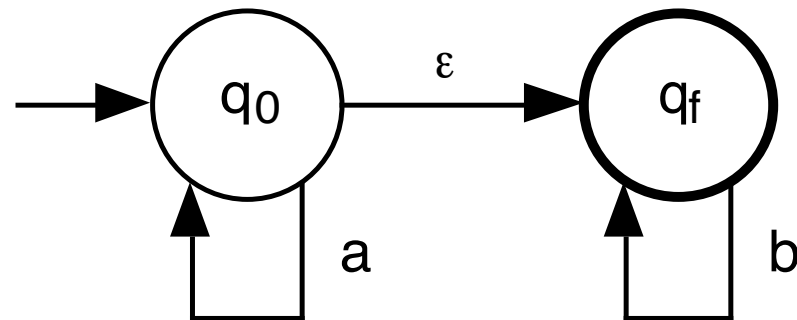
- tal que

$$\delta_\varepsilon^*(P) = \bigcup_{q \in P} \delta_\varepsilon(q)$$

◆ Por simplicidade, δ_ϵ e δ_ϵ^*

- ambas denotadas por δ_ϵ

Exp: Computação Vazia



- $\delta_\epsilon(q_0) = \{q_0, q_f\}$
- $\delta_\epsilon(q_f) = \{q_f\}$
- $\delta_\epsilon(\{q_0, q_f\}) = \{q_0, q_f\}$

◆ **Computação de um AFN_ϵ para uma entrada w**

- sucessiva aplicação da função programa
- para cada símbolo de w (da esquerda para a direita)
- cada passo de aplicação *intercalado* com computações vazias
- até ocorrer uma condição de parada

◆ **Assim, antes de processar a próxima transição**

- determinar
 - * todos os demais estados atingíveis
 - * exclusivamente por movimentos vazios

Def: Função Programa Estendida, Computação

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

$$\delta^*: 2^Q \times \Sigma^* \rightarrow 2^Q$$

indutivamente definida

- $\delta^*(P, \epsilon) = \delta_{\epsilon}(P)$
- $\delta^*(P, wa) = \delta_{\epsilon}(R)$ onde $R = \{ r \mid r \in \delta(s, a) \text{ e } s \in \delta^*(P, w) \}$

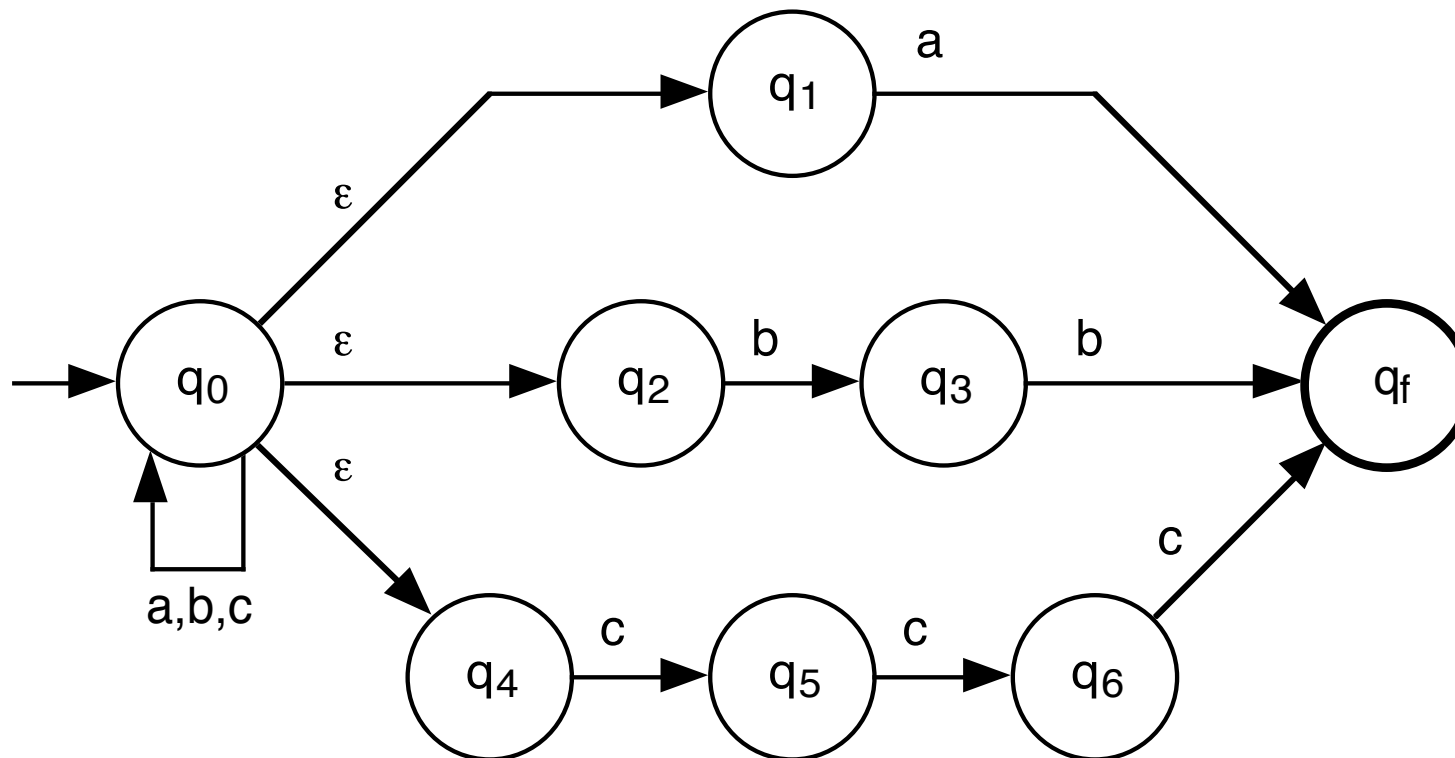
◆ Parada do processamento, Ling. Aceita/Rejeitada

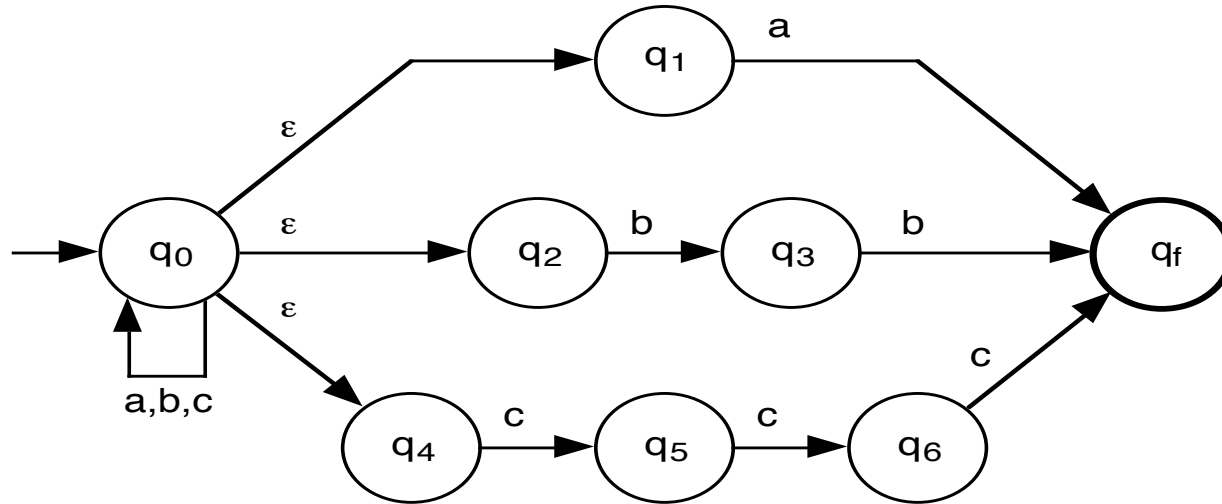
- **análoga** à do autômato finito **não-determinístico**

Exp: Computação Vazia, Computação

$L_8 = \{ w \mid w \text{ possui como sufixo } a \text{ ou } bb \text{ ou } ccc \}$

$M_8 = (\{ a, b, c \}, \{ q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_f \}, \delta_8, q_0, \{ q_f \})$





$$\delta^*({q_0}, abb) = \delta\epsilon(\{ r \mid r \in \delta(s, b) \text{ e } s \in \delta^*({q_0}, ab) \}) \quad (1)$$

$$\delta^*({q_0}, ab) = \delta\epsilon(\{ r \mid r \in \delta(s, b) \text{ e } s \in \delta^*({q_0}, a) \}) \quad (2)$$

$$\delta^*({q_0}, a) = \delta\epsilon(\{ r \mid r \in \delta(s, a) \text{ e } s \in \delta^*({q_0}, \epsilon) \}) \quad (3)$$

Como:

$$\delta^*({q_0}, \epsilon) = \delta\epsilon({q_0}) = \{ q_0, q_1, q_2, q_4 \} \quad \text{considerado em (3)}$$

$$\delta^*({q_0}, a) = \{ q_0, q_1, q_2, q_4, q_f \} \quad \text{considerado em (2)}$$

$$\delta^*({q_0}, ab) = \{ q_0, q_1, q_2, q_3, q_4 \} \quad \text{considerado em (1)}$$

Resulta na computação: $\delta^*({q_0}, abb) = \{ q_0, q_1, q_2, q_3, q_4, q_f \}$

Teorema: Equivalência entre AFN e AFN ϵ

Classe dos Autômatos Finitos com Movimentos Vazios é equivalente à Classe dos Autômatos Finitos Não-Determinísticos

Prova: (por indução)

Mostrar que

- a partir de um AFN ϵ M qualquer
- construir um AFN M_N que realiza as mesmas computações
- M_N simula M

AFN ϵ \rightarrow AFN

- construção de uma função programa sem movimentos vazios
- conjunto de estados destino de cada transição não-vazia
 - * ampliado com os demais estados possíveis de serem atingidos exclusivamente por transições vazias

$M = (\Sigma, Q, \delta, q_0, F)$ um AFN_ϵ qualquer. AFN construído

$$M_N = (\Sigma, Q, \delta_N, q_0, F_N)$$

- $\delta_N: Q \times \Sigma \rightarrow 2^Q$ é tal que

$$\delta_N(q, a) = \delta^*(\{q\}, a)$$

- F_N é o conjunto de todos os estados q pertencentes a Q

$$\delta_\epsilon(q) \cap F \neq \emptyset$$

- * estados que atingem estados finais via computações vazias

Demonstração que, de fato, o $AFN M_N$ simula o $AFN_\epsilon M$

- indução no tamanho da palavra
- **exercício**

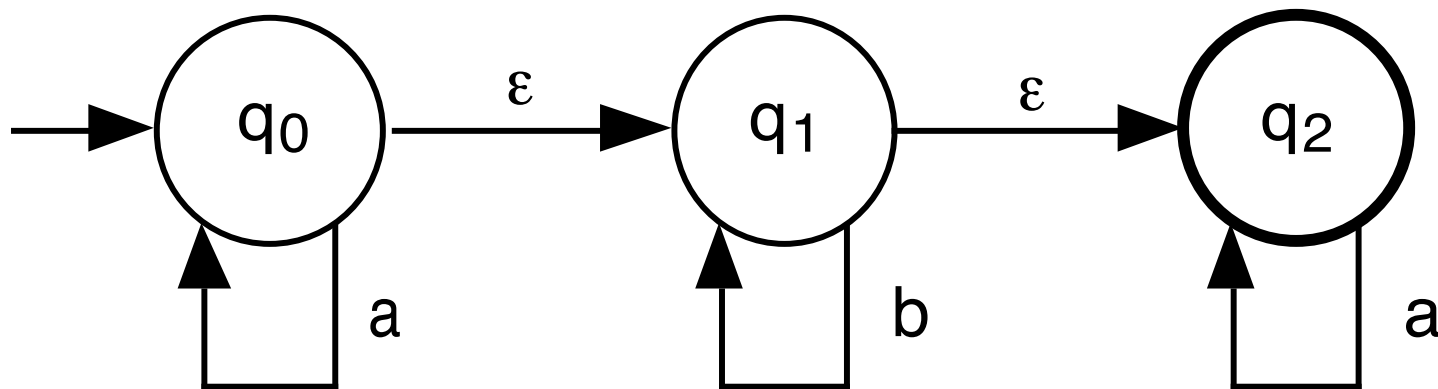
◆ **Portanto, linguagem aceita por AFN_ϵ**

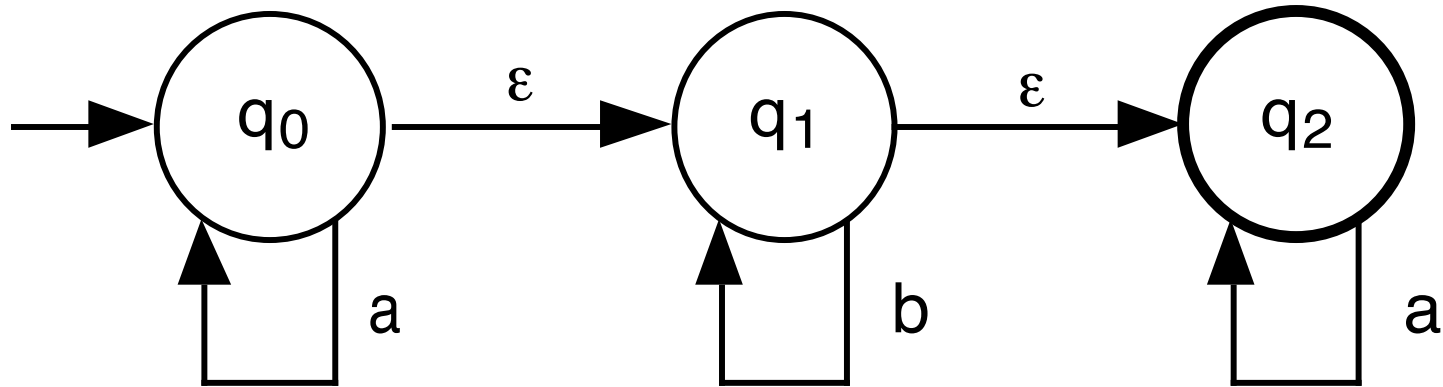
- **é Linguagem Regular ou Tipo 3**

Exp: Construção de um AFN a partir de um AFN ϵ

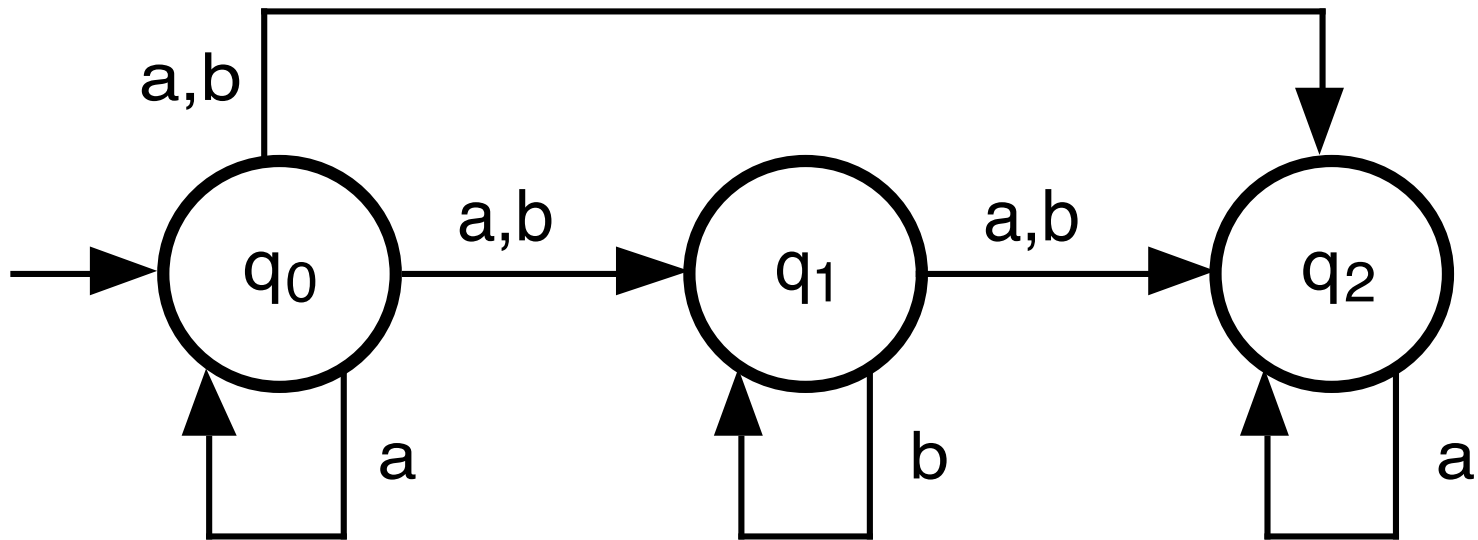
AFN ϵ - $M_g = (\{ a, b \}, \{ q_0, q_1, q_2 \}, \delta_g, q_0, \{ q_2 \})$

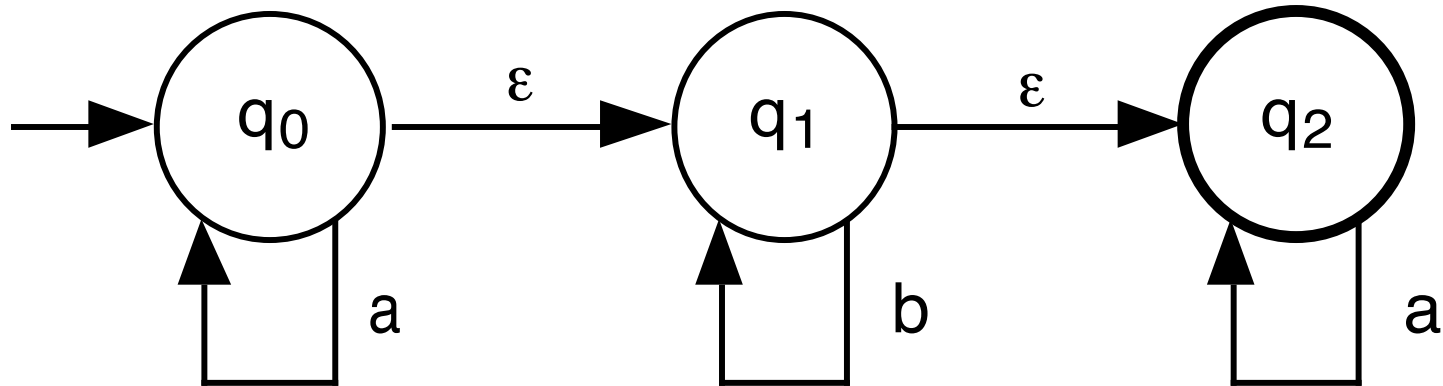
δ_g	a	b	ϵ
q ₀	{ q ₀ }	-	{ q ₁ }
q ₁	-	{ q ₁ }	{ q ₂ }
q ₂	{ q ₂ }	-	-





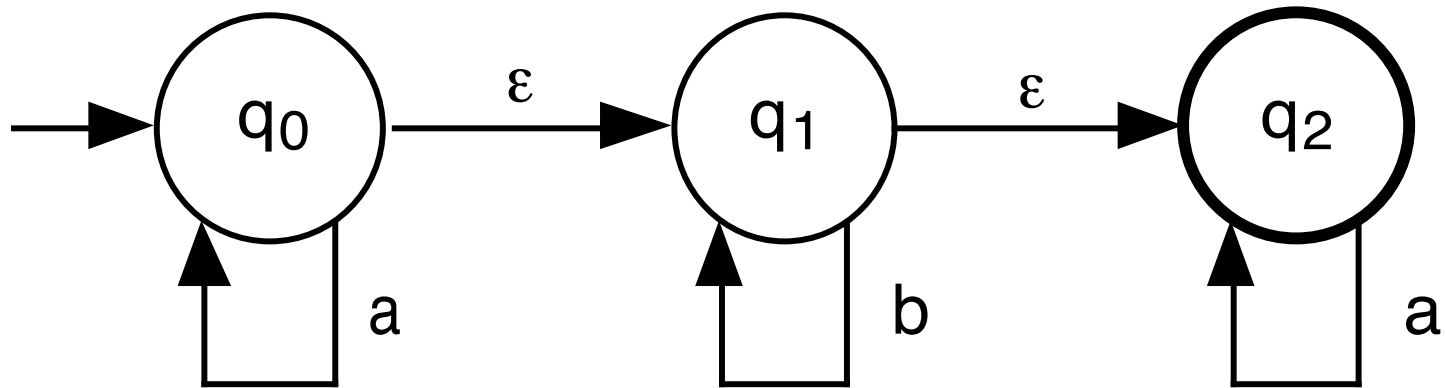
$$M_{g_N} = (\{ a, b \}, \{ q_0, q_1, q_2 \}, \delta_{g_N}, q_0, F_N)$$





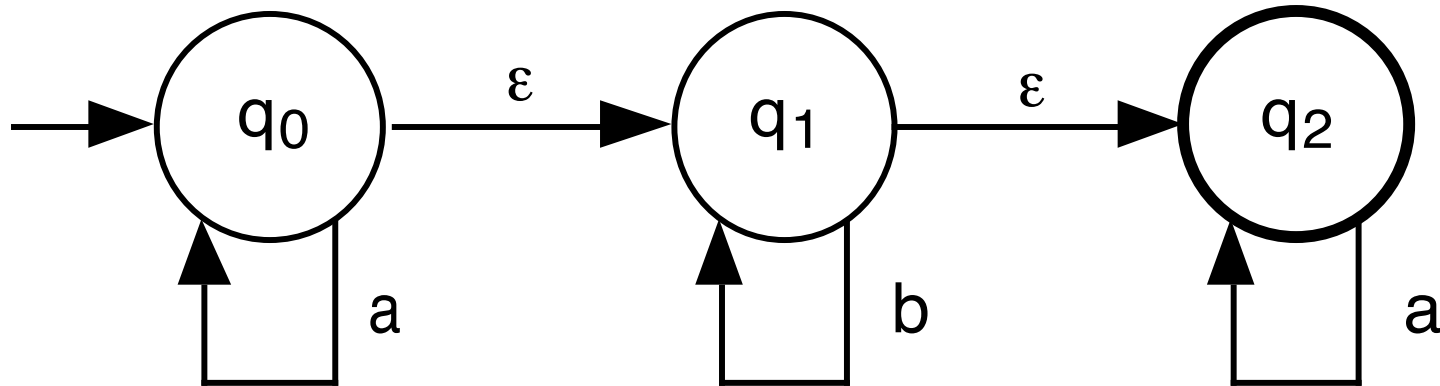
$$F_N = \{ q_0, q_1, q_2 \}$$

- $\delta_\epsilon(q_0) = \{ q_0, q_1, q_2 \}$
- $\delta_\epsilon(q_1) = \{ q_1, q_2 \}$
- $\delta_\epsilon(q_2) = \{ q_2 \}$



Na construção de δ_{g_N}

- $\delta_{g_N}^*(\{q_0\}, \epsilon) = \{q_0, q_1, q_2\}$
- $\delta_{g_N}^*(\{q_1\}, \epsilon) = \{q_1, q_2\}$
- $\delta_{g_N}^*(\{q_2\}, \epsilon) = \{q_2\}$



Assim, δ_{g_N} é tal que

$$\delta_{g_N}(q_0, a) = \delta_{g^*}(\{q_0\}, a) =$$

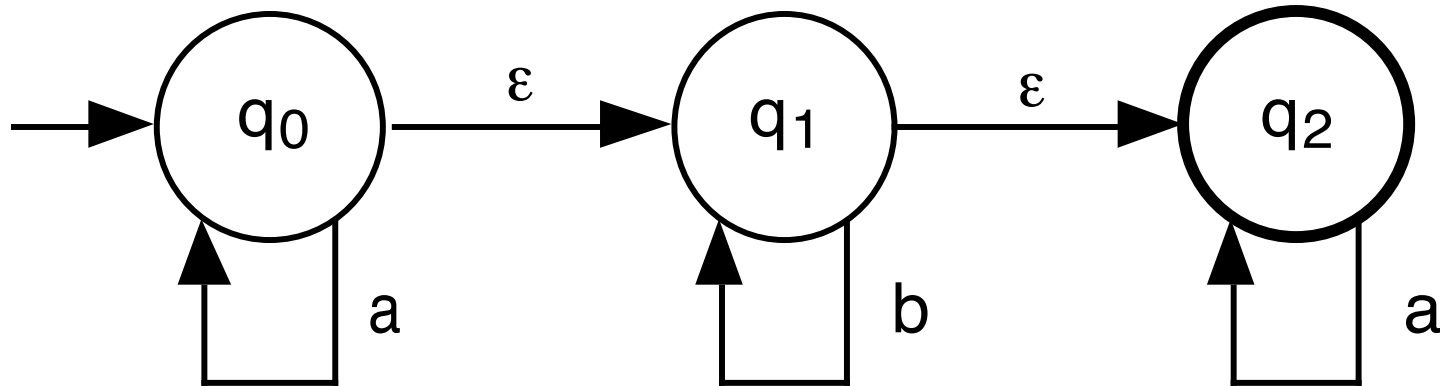
$$\delta_{\epsilon}(\{r \mid r \in \delta(s, a) \text{ e } s \in \underline{\delta^*}(\{q_0\}, \epsilon)\}) = \{q_0, q_1, q_2\}$$

$$\delta_{g_N}(q_0, b) = \delta_{g^*}(\{q_0\}, b) =$$

$$\delta_{\epsilon}(\{r \mid r \in \delta(s, b) \text{ e } s \in \underline{\delta^*}(\{q_0\}, \epsilon)\}) = \{q_1, q_2\}$$

$$\delta_{g_N}(q_1, a) = \delta_{g^*}(\{q_1\}, a) =$$

$$\delta_{\epsilon}(\{r \mid r \in \delta(s, a) \text{ e } s \in \underline{\delta^*}(\{q_1\}, \epsilon)\}) = \{q_2\}$$



- $\delta_{g_N}(q_1, b) = \underline{\delta}_{g^*}(\{q_1\}, b) = \delta_{\epsilon}(\{r \mid r \in \delta(s, b) \text{ e } s \in \underline{\delta}^*(\{q_1\}, \epsilon)\}) = \{q_1, q_2\}$
- $\delta_{g_N}(q_2, a) = \underline{\delta}_{g^*}(\{q_2\}, a) = \delta_{\epsilon}(\{r \mid r \in \delta(s, a) \text{ e } s \in \underline{\delta}^*(\{q_2\}, \epsilon)\}) = \{q_2\}$
- $\delta_{g_N}(q_2, b) = \underline{\delta}_{g^*}(\{q_2\}, b) = \delta_{\epsilon}(\{r \mid r \in \delta(s, b) \text{ e } s \in \underline{\delta}^*(\{q_2\}, \epsilon)\})$ é indefinida

3 – Linguagens Regulares

- 3.1 Sistema de Estados Finitos
- 3.2 Composição Seqüencial, Concorrente e Não-Determinista
- 3.3 Autômato Finito
- 3.4 Autômato Finito Não-Determinístico
- 3.5 Autômato Finito com Movimentos Vazios
- 3.6 Expressão Regular
- 3.7 Gramática Regular

3.6 Expressão Regular

- ◆ **Toda linguagem regular pode ser descrita por uma**
 - Expressão Regular
- ◆ **Formalismo denotacional (gerador)**
- ◆ **Definida a partir de**
 - conjuntos (linguagens) básicos
 - concatenação e união
- ◆ **Adequadas para a comunicação**
 - humano × humano
 - humano × máquina

Def: Expressão Regular (ER)

Base de Indução

- \emptyset é ER
 - * denota a linguagem vazia: \emptyset
- ϵ é ER
 - * denota a linguagem $\{ \epsilon \}$
- x é ER (para qualquer $x \in \Sigma$)
 - * denota a linguagem $\{ x \}$

Def: ...Expressão Regular (ER)

Passo de Indução: se r e s são ER e denotam as ling. R e S , então

- *União.* $(r + s)$ é ER
* denota a linguagem $R \cup S$
- *Concatenação.* (rs) é ER
* denota a linguagem $RS = \{ uv \mid u \in R \text{ e } v \in S \}$
- *Concatenação Sucessiva.* (r^*) é ER
* denota a linguagem R^*

Def: Linguagem Gerada por uma ER

Se r é ER, a correspondente **linguagem denotada** é dita

- **Linguagem Gerada** por r

$L(r)$ ou $GERA(r)$

◆ **Omissão de parênteses em uma ER é usual**

- **concatenação sucessiva: precedência sobre concatenação e união**
- **concatenação: precedência sobre união**

Exp: Expressão Regular

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

Exp: ...Expressão Regular

ER	Linguagem Gerada
aa	somente a palavra aa
ba^*	todas as palavras que iniciam por b , seguido por zero ou mais a
$(a + b)^*$	todas as palavras sobre $\{a, b\}$
$(a + b)^*aa(a + b)^*$	todas as palavras contendo aa como subpalavra
$a^*ba^*ba^*$	todas as palavras contendo exatamente dois b
$(a + b)^*(aa + bb)$	todas as palavras que terminam com aa ou bb
$(a + \epsilon)(b + ba)^*$	todas as palavras que não possuem dois a consecutivos

Exp: ...Expressão Regular

Linguagem gerada pela ER $(a + b)^*(aa + bb)$

- a e b denotam $\{ a \}$ e $\{ b \}$, respectivamente
- $a + b$ denota $\{ a \} \cup \{ b \} = \{ a, b \}$
- $(a + b)^*$ denota $\{ a, b \}^*$
- aa e bb denotam $\{ a \} \{ a \} = \{ aa \}$ e $\{ b \} \{ b \} = \{ bb \}$, respectivamente
- $(aa + bb)$ denota $\{ aa \} \cup \{ bb \} = \{ aa, bb \}$
- $(a + b)^*(aa + bb)$ denota $\{ a, b \}^* \{ aa, bb \}$

Portanto, $GERA((a + b)^*(aa + bb))$ é

$\{ aa, bb, aaa, abb, baa, bbb, \\ aaaa, aabb, abaa, abbb, baaa, babb, bbaa, bbbb, \dots \}$

Teorema: Expressão Regular \rightarrow Linguagem Regular

Se r é ER, então $GERA(r)$ é linguagem regular

Prova: (por indução)

Uma linguagem é regular sse é possível construir um

- AFD, AFN ou AFN_ϵ que reconheça a linguagem

É necessário mostrar que

- dada uma ER r qualquer
- é possível construir um autômato finito M tal que

$$ACEITA(M) = GERA(r)$$

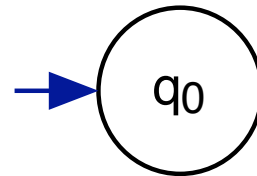
Demonstração: indução no número de operadores

Base de indução. r ER com zero operadores

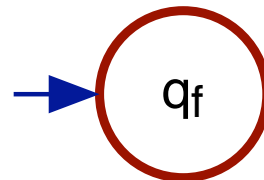
- $r = \emptyset$
 - * Autômato???
- $r = \varepsilon$
 - * Autômato???
- $r = x$ ($x \in \Sigma$)
 - * Autômato???

Base de indução. $r \in R$ com zero operadores

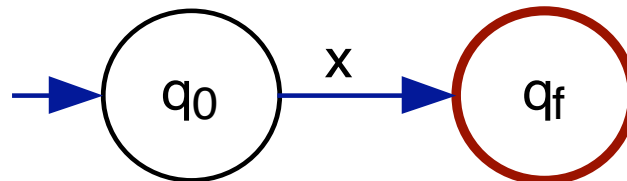
- $r = \emptyset$. Autômato: $M_1 = (\emptyset, \{q_0\}, \delta_1, q_0, \emptyset)$



- $r = \varepsilon$. Autômato: $M_2 = (\emptyset, \{q_f\}, \delta_2, q_f, \{q_f\})$



- $r = x$ ($x \in \Sigma$). Autômato: $M_3 = (\{x\}, \{q_0, q_f\}, \delta_3, q_0, \{q_f\})$



Hipótese de Indução. $r \in ER$ com até $n > 0$ operadores

- suponha que é possível definir um AF que aceita $GERA(r)$

Passo de Indução. $r \in ER$ com $n + 1$ operadores

- r pode ser representada por (r_1 e r_2 possuem conjuntamente no máximo n operadores)
 - * $r = r_1 + r_2$
 - * $r = r_1 r_2$
 - * $r = r_1^*$
- por hipótese de indução, existem

$$M_1 = (\Sigma_1, Q_1, \delta_1, q_{01}, \{ q_{f1} \}) \quad e \quad M_2 = (\Sigma_2, Q_2, \delta_2, q_{02}, \{ q_{f2} \})$$

$$ACEITA(M_1) = GERA(r_1) \quad e \quad ACEITA(M_2) = GERA(r_2)$$

$$r = r_1 + r_2$$

- Autômato???

$$r = r_1 r_2$$

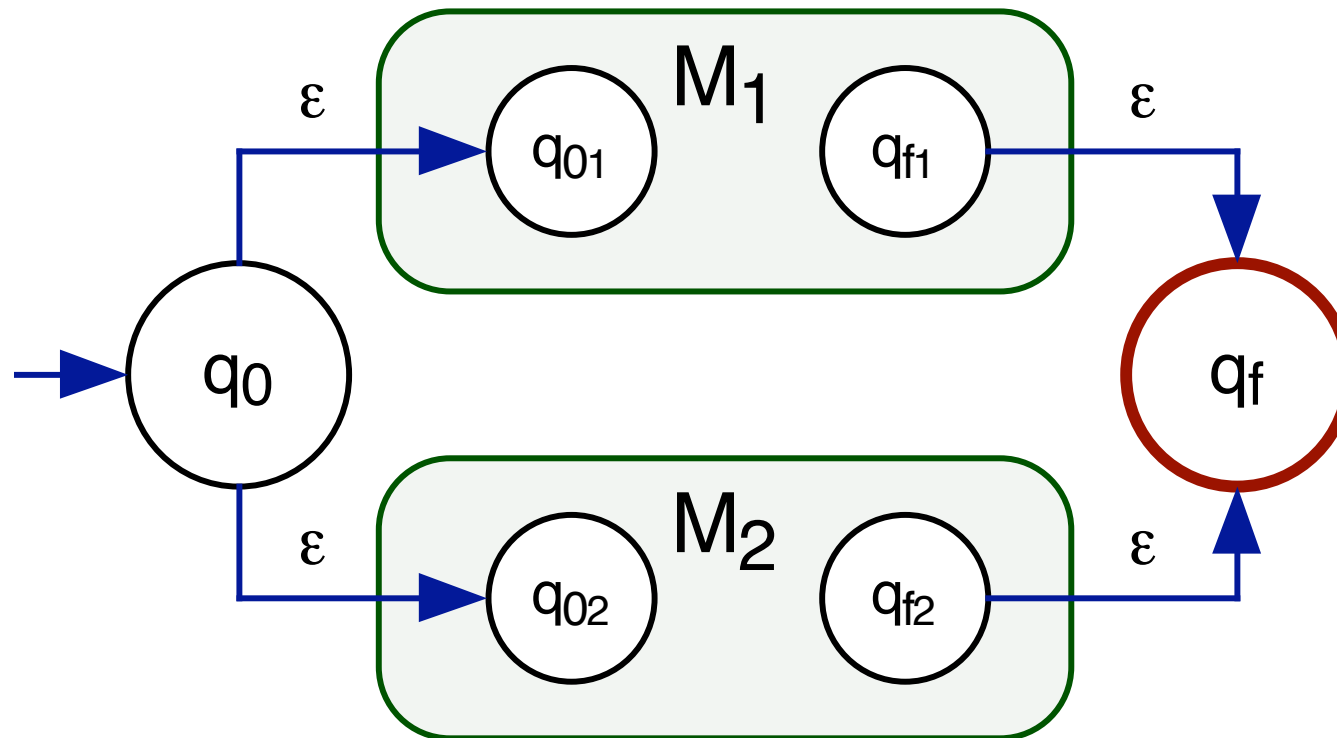
- Autômato???

$$r = r_1^*$$

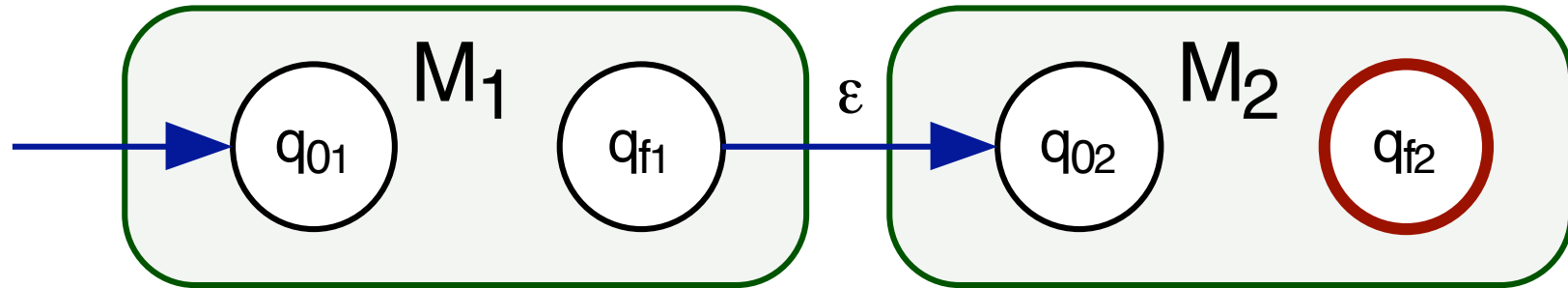
- Autômato???

- sem perda de generalidade:
 - * M_1 e M_2 possuem exatamente um estado final (**exercícios**)
 - * estados dos autômatos: conjunto disjuntos (se não forem?)

$r = r_1 + r_2$. Autômato $M = (\Sigma_1 \cup \Sigma_2, Q_1 \cup Q_2 \cup \{q_0, q_f\}, \delta, q_0, \{q_f\})$

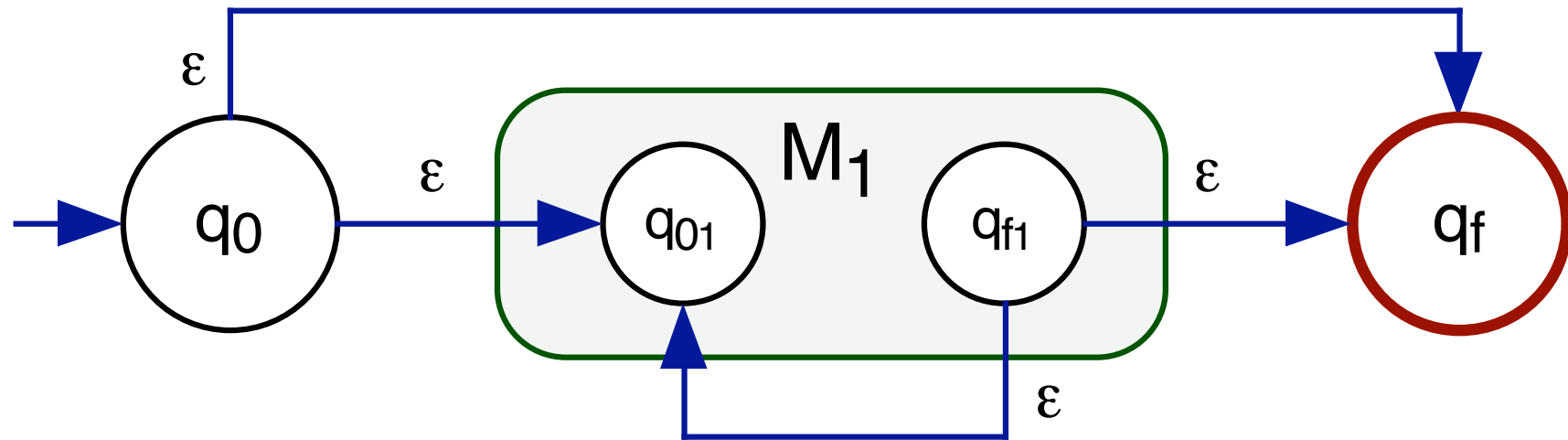


$r = r_1 r_2$. Autômato $M = (\Sigma_1 \cup \Sigma_2, Q_1 \cup Q_2, \delta, q_{01}, \{q_{f2}\})$



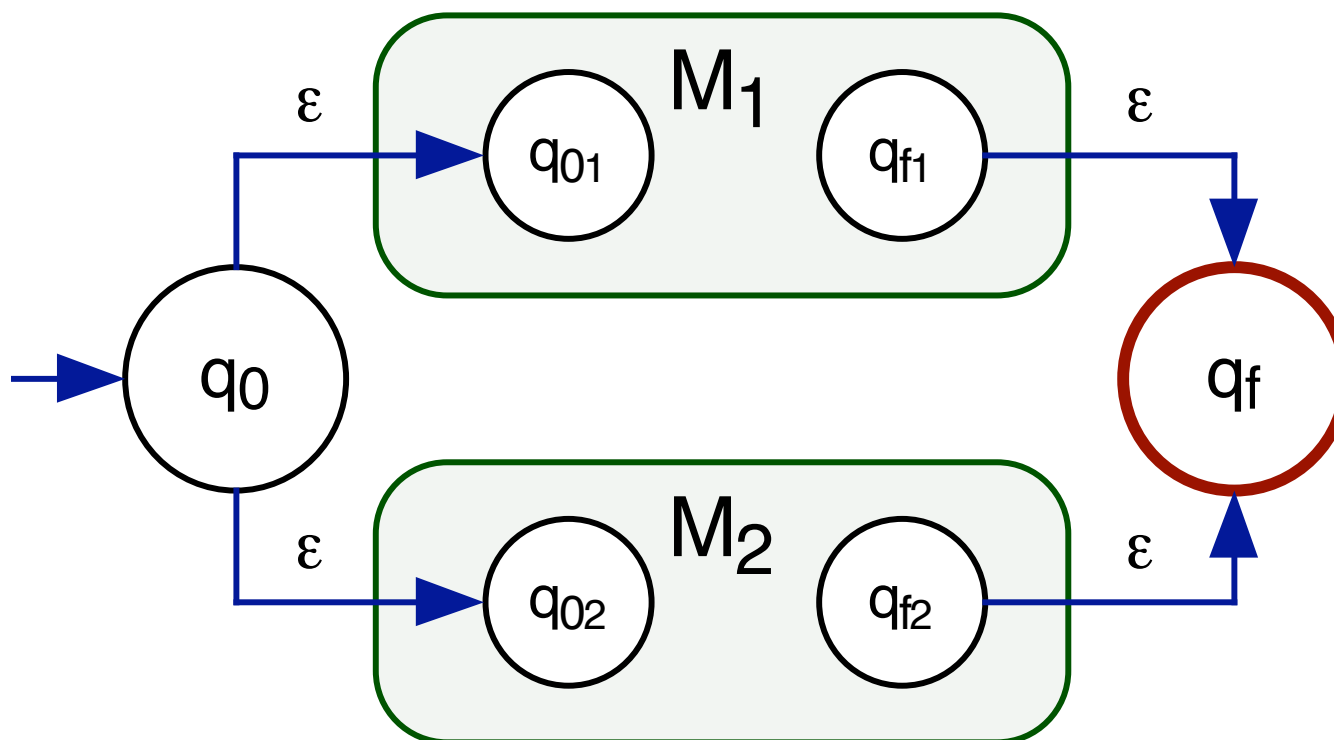
$r = r_1^*$. Autômato (suponha $q_0 \notin Q_1, q_f \notin Q_1$)

- $M = (\Sigma_1, Q_1 \cup \{q_0, q_f\}, \delta, q_0, \{q_f\})$



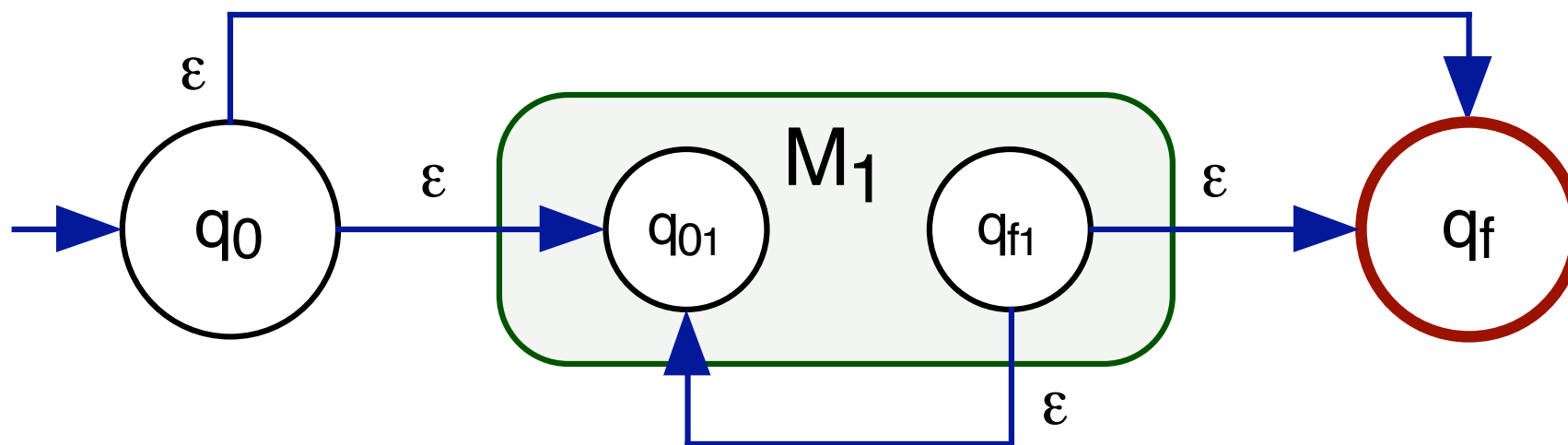
◆ **Exercício:** no caso $r = r_1 + r_2$

- não introduzir os estados q_0 e q_f
- identificar ("unificar") os estados iniciais/finais de M_1/M_2 ???



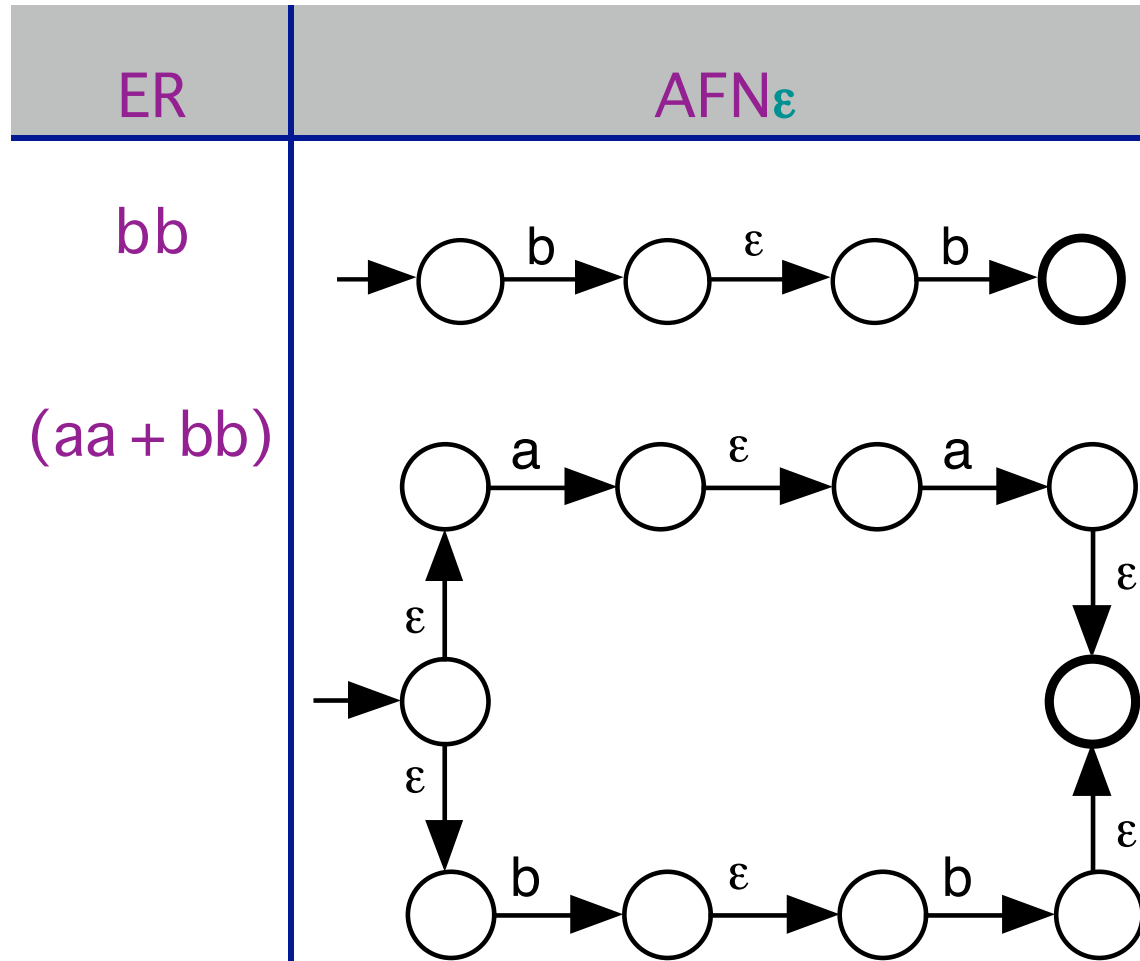
◆ **Exercício: no caso $r = r_1^*$**

- não introduzir o estado q_f
- manter q_{f_1} como o estado final
- transição vazia de q_0 para q_{f_1} ???



Exp: AFN_{ϵ} a partir de $a^*(aa + bb)$

ER	AFN_{ϵ}
a	
b	
a^*	
aa	



Teorema: Linguagem Regular \rightarrow Expressão Regular

Se L é linguagem regular, então existe uma ER r tal que

$$\text{GERA}(r) = L$$

O teorema não será provado

3 – Linguagens Regulares

- 3.1 Sistema de Estados Finitos
- 3.2 Composição Seqüencial, Concorrente e Não-Determinista
- 3.3 Autômato Finito
- 3.4 Autômato Finito Não-Determinístico
- 3.5 Autômato Finito com Movimentos Vazios
- 3.6 Expressão Regular
- 3.7 Gramática Regular

3.7 Gramática Regular

◆ Formalismo Gramáticas

- permite definir tanto **linguagens regulares** como **não-regulares**

◆ Gramática Regular

- **restrições nas regras** de produção
- existe **mais de uma forma** de **restringir** as regras de produção
 - * **Gramáticas Lineares**

Def: Gramáticas Lineares

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

Gramática Linear à Direita (GLD)

$$A \rightarrow wB \quad \text{ou} \quad A \rightarrow w$$

Gramática Linear à Esquerda (GLE)

$$A \rightarrow Bw \quad \text{ou} \quad A \rightarrow w$$

Gramática Linear Unitária à Direita (GLUD)

- como na gramática linear à direita. Adicionalmente

$$|w| \leq 1$$

Gramática Linear Unitária à Esquerda (GLUE)

- como na gramática linear à esquerda. Adicionalmente

$$|w| \leq 1$$

◆ Lado direito de uma produção

- no máximo uma variável
 - * sempre antecede (linear à esquerda)
 - * ou sucede (linear à direita)
 - * qualquer subpalavra (eventualmente vazia) de terminais

◆ Exercício

- gramática simultaneamente nas quatro formas lineares?

Teorema: Equivalência das Gramáticas Lineares

Seja L uma linguagem. Então:

- L é gerada por uma GLD sse
- L é gerada por uma GLE sse
- L é gerada por uma GLUD sse
- L é gerada por uma GLUE

◆ Diversas formas das gramáticas lineares

- formalismos equivalentes
- demonstração do teorema: exercício

Def: Gramática Regular (GR)

G é uma gramática linear

Def: Linguagem Gerada

$G = (V, T, P, S)$ gramática

$L(G)$ ou $GERA(G)$

é tal que

$$L(G) = \{ w \in T^* \mid S \Rightarrow^+ w \}$$

Exp: Gramática Regular: $a(ba)^*$

???

Exp: Gramática Regular: $a(ba)^*$

Linear à Direita. $G = (\{ S, A \}, \{ a, b \}, P, S)$

- $S \rightarrow aA$
- $A \rightarrow baA \mid \epsilon$

Linear à Esquerda. $G = (\{ S \}, \{ a, b \}, P, S)$

- $S \rightarrow Sba \mid a$

Linear Unitária à Direita. $G = (\{ S, A, B \}, \{ a, b \}, P, S)$

- $S \rightarrow aA$
- $A \rightarrow bB \mid \epsilon$
- $B \rightarrow aA$

Linear Unitária à Esquerda. $G = (\{ S, A \}, \{ a, b \}, P, S)$

- $S \rightarrow Aa \mid a$
- $A \rightarrow Sb$

Exp: Gramática Regular: $(a + b)^*(aa + bb)$

Linear à Direita. $G = (\{ S, A \}, \{ a, b \}, P, S)$, e P é tal que

- $S \rightarrow aS \mid bS \mid A$
- $A \rightarrow aa \mid bb$

Linear à Esquerda. $G = (\{ S, A \}, \{ a, b \}, P, S)$, e P é tal que

- $S \rightarrow Aaa \mid Abb$
- $A \rightarrow Aa \mid Ab \mid \varepsilon$

Obs: Gramática Linear à Esquerda e Linear à Direita

Suponha $|w| \geq 1$

Produções **simultaneamente** do tipo

- $A \rightarrow wB$ (direita) e
- $A \rightarrow Bw$ (esquerda)

correspondente **linguagem gerada**

- poderá **não** ser **regular**
- **não** é uma **gramática regular**

É possível desenvolver uma **gramática**, com produções lineares à **direita** e à **esquerda**, que gera (**exercício**)

$$\{ a^n b^n \mid n \in \mathbb{N} \}$$

Teorema: Gramática Regular \rightarrow Linguagem Regular

Se L é gerada por uma gramática regular,
então L é linguagem regular

Prova: (por indução)

Mostrar que

- dado uma GLUD G qualquer
- é possível construir um AFN_{ϵ} M tq

$$ACEITA(M) = GERA(G)$$

M simula as derivações de G

- demonstração de que $ACEITA(M) = GERA(r)$
- indução no número de derivações

Suponha $G = (V, T, P, S)$ uma GLUD. Seja o AFN_ϵ

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

- $\Sigma = T$
- $Q = V \cup \{q_f\}$
- $F = \{q_f\}$
- $q_0 = S$

(suponha $q_f \notin V$)

Tipo da Produção	Transição Gerada
$A \rightarrow \epsilon$	$\delta(A, \epsilon) = q_f$
$A \rightarrow a$	$\delta(A, a) = q_f$
$A \rightarrow B$	$\delta(A, \epsilon) = B$
$A \rightarrow aB$	$\delta(A, a) = B$

M simula as derivações de G

$$\text{ACEITA}(M) = \text{GERA}(G)$$

Base de indução. $S \Rightarrow^1 \alpha$. Quatro casos

- $\alpha = \varepsilon$ existe $S \rightarrow \varepsilon$ Logo, $\delta(S, \varepsilon) = q_f$
- $\alpha = a$ existe $S \rightarrow a$ Logo, $\delta(S, a) = q_f$
- $\alpha = A$ existe $S \rightarrow A$ Logo, $\delta(S, \varepsilon) = A$
- $\alpha = aA$ existe $S \rightarrow aA$ Logo, $\delta(S, a) = A$

Hipótese de indução. $S \Rightarrow^n \alpha$, $n > 1$. Dois casos

- $\alpha = w$ então $\underline{\delta}^*(S, w) = q_f$ (1)
- $\alpha = wA$ então $\underline{\delta}^*(S, w) = A$ (2)

Passo de Indução. $S \Rightarrow^{n+1} \alpha$. Então (2) é a única hipótese que importa

$$S \Rightarrow^n wA \Rightarrow^1 \alpha$$

Quatro casos:

- $\alpha = w\varepsilon = w$. Existe $A \rightarrow \varepsilon$. Logo

$$\underline{\delta}^*(S, w\varepsilon) = \delta(\underline{\delta}^*(S, w), \varepsilon) = \delta(A, \varepsilon) = q_f$$

- $\alpha = wb$. Existe $A \rightarrow b$. Logo

$$\underline{\delta}^*(S, wb) = \delta(\underline{\delta}^*(S, w), b) = \delta(A, b) = q_f$$

- $\alpha = wB$. Existe $A \rightarrow B$. Logo

$$\underline{\delta}^*(S, w\varepsilon) = \delta(\underline{\delta}^*(S, w), \varepsilon) = \delta(A, \varepsilon) = B$$

- $\alpha = wbB$. Existe $A \rightarrow bB$. Logo

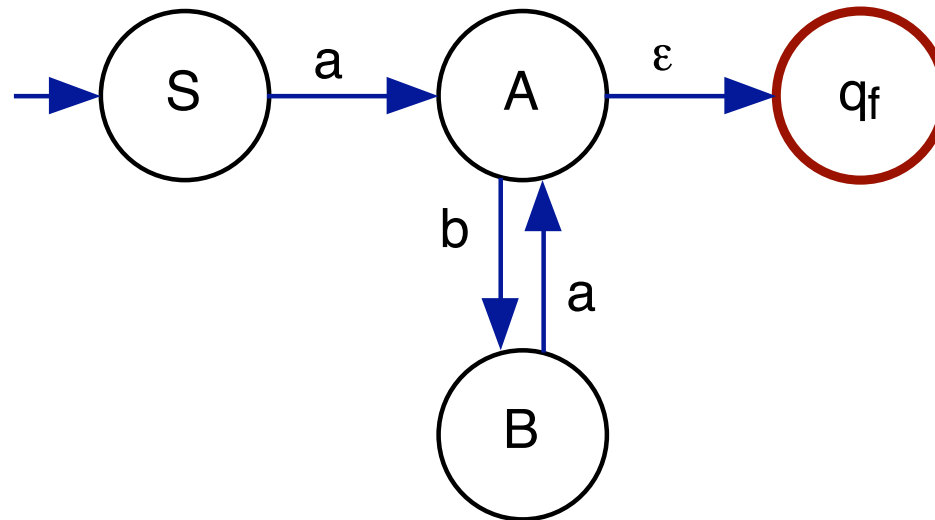
$$\underline{\delta}^*(S, wb) = \delta(\underline{\delta}^*(S, w), b) = \delta(A, b) = B$$

Exp: Construção de um AFN_ϵ a partir de uma GR

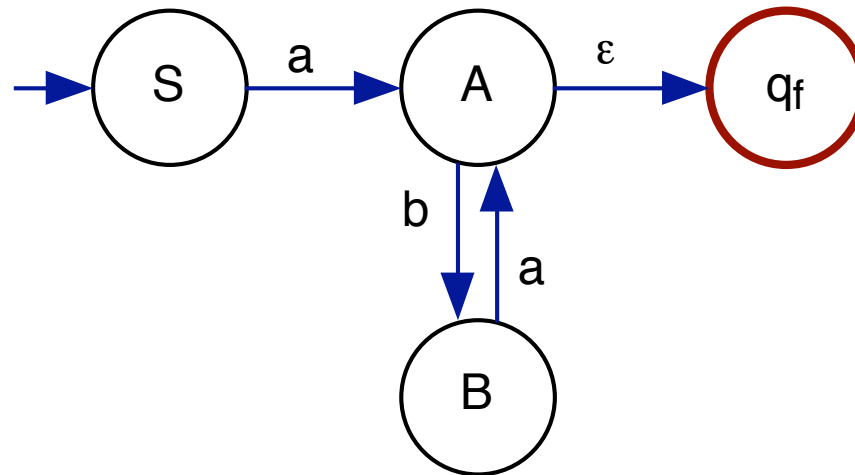
$$G = (\{ S, A, B \}, \{ a, b \}, P, S)$$

- $S \rightarrow aA$
- $A \rightarrow bB \mid \epsilon$
- $B \rightarrow aA$

$$M = (\{ a, b \}, \{ S, A, B, q_f \}, \delta, S, \{ q_f \})$$



Produção	Transição
$S \rightarrow aA$	$\delta(S, a) = A$
$A \rightarrow bB$	$\delta(A, b) = B$
$A \rightarrow \varepsilon$	$\delta(A, \varepsilon) = q_f$
$B \rightarrow aA$	$\delta(B, a) = A$



Teorema: Linguagem Regular \rightarrow Gramática Regular

Se L é linguagem regular, então existe G , gramática regular que gera L

Prova: (*por indução*)

L é linguagem regular

- existe um AFD $M = (\Sigma, Q, \delta, q_0, F)$ tal que $ACEITA(M) = L$

Construção de uma GLUD G

$$GERA(G) = ACEITA(M)$$

- derivação simula a função programa estendida

Suponha um AFD $M = (\Sigma, Q, \delta, q_0, F)$ tal que $ACEITA(M) = L$

Seja a gramática regular

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

- $V = Q \cup \{S\}$
- $T = \Sigma$
- suponha $q_i, q_k \in Q, q_f \in F$ e $a \in \Sigma$

(suponha $S \notin Q$)

Transição	Produção
-	$S \rightarrow q_0$
-	$q_f \rightarrow \epsilon$
$\delta(q_i, a) = q_k$	$q_i \rightarrow aq_k$

GERA(G) = ACEITA(M)? Indução no tamanho da palavra ($w \in \Sigma^*$)

Base de Indução. $|w| = 0$

- por definição, $S \rightarrow q_0$ é produção
- se $\varepsilon \in \text{ACEITA}(M)$, então
 - * q_0 é estado final
 - * $q_0 \rightarrow \varepsilon$ é produção

$$S \Rightarrow q_0 \Rightarrow \varepsilon$$

Hipótese de Indução. $|w| = n$ ($n \geq 1$) e $\delta^*(q_0, w) = q$. Dois casos

- q não é final. Suponha $S \Rightarrow^n wq$ (única hipótese que importa)
- q é final. Suponha $S \Rightarrow^n wq \Rightarrow w$

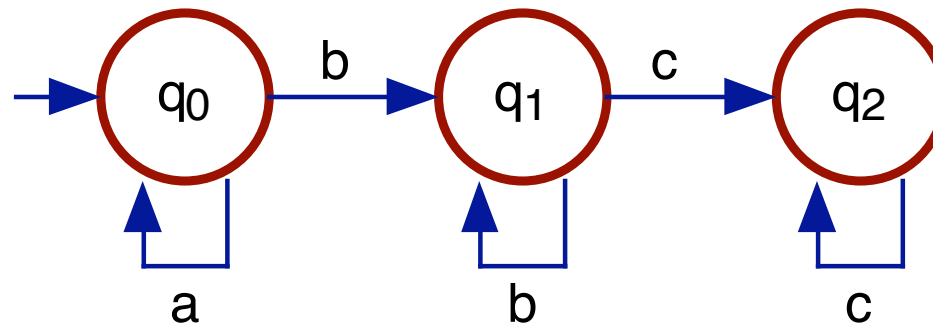
Passo de Indução. $|wa| = n + 1$ e $\delta^*(q_0, wa) = p$. Então

$$\delta(\delta^*(q_0, w), a) = \delta(q, a) = p$$

- p não é final
 - * $S \Rightarrow^n wq \Rightarrow^1 wap$
- p é final
 - * $S \Rightarrow^n wq \Rightarrow^1 wap \Rightarrow^1 wa$

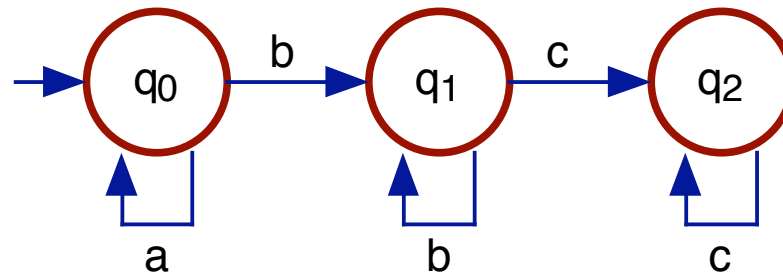
Exp: Construção de uma GR a partir de um AFD

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



$G = (\{ q_0, q_1, q_2, S \}, \{ a, b, c \}, P, S)$

Transição	Produção
-	$S \rightarrow q_0$
-	$q_0 \rightarrow \varepsilon$
-	$q_1 \rightarrow \varepsilon$
-	$q_2 \rightarrow \varepsilon$
$\delta(q_0, a) = q_0$	$q_0 \rightarrow aq_0$
$\delta(q_0, b) = q_1$	$q_0 \rightarrow bq_1$
$\delta(q_1, b) = q_1$	$q_1 \rightarrow bq_1$
$\delta(q_1, c) = q_2$	$q_1 \rightarrow cq_2$
$\delta(q_2, c) = q_2$	$q_2 \rightarrow cq_2$



Linguagens Formais e Autômatos

P. Blauth Menezes

- 1 **Introdução e Conceitos Básicos**
- 2 **Linguagens e Gramáticas**
- 3 **Linguagens Regulares**
- 4 **Propriedades das Linguagens Regulares**
- 5 **Autômato Finito com Saída**
- 6 **Linguagens Livres do Contexto**
- 7 **Propriedades e Reconhecimento das Linguagens Livres do Contexto**
- 8 **Linguagens Recursivamente Enumeráveis e Sensíveis ao Contexto**
- 9 **Hierarquia de Classes e Linguagens e Conclusões**