■■ série livros didáticos informática ufrgs ■■

bookman

grupo a
Conhecimento que transforma.

.inf
INSTITUTO
DE INFORMÁTICA
UFRGS

Linguagens Formais e Autômatos

Paulo Blauth Menezes

Linguagens Formais e Autômatos

P. Blauth Menezes

3 – Linguagens Regulares

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 subsequentes

- minimização de autômatos finitos
- propriedades da classe
- algumas importantes aplicações

3 – Linguagens Regulares

3.1 Sistema de estados finitos
3.2 Composição sequencial, 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 explicitados 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 ser 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

- estados dos processadores e memórias
  - sistema de estados finitos
- entretanto, podem existir memórias adicionais
  - discos, fitas, memórias auxiliares, etc.
  - assim, o número de estados não necessariamente é predefinido
  - não satisfaz aos princípios dos autômatos finitos

- de fato, o estudo da computabilidade
  - exige uma memória *sem limite predefinido*
- Máquina de Turing
  - mais adequado ao estudo da computabilidade
- computabilidade e solucionabilidade de problemas
  - apenas introduzidos
  - questões tratadas na teoria da computação

3 – Linguagens Regulares

3.1 Sistema de estados finitos
3.2 Composição sequencial, 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 Sequencial, 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

- Sequencial
- Concorrente
- Não determinista

Sequencial

- 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

Sequencial

- 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

- sequencial 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
- se pelo menos um caminho alternativo reconhece/soluciona
  - a máquina como um todo é considerada capaz de reconhecer/solucionar

Semântica do não determinismo adotada

- difere da adotada no estudo dos modelos para concorrência
  - ∗ exemplo: sistemas operacionais
  - ∗ não confundir com a semântica da concorrência

3 – Linguagens Regulares

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
  - para o estado corrente e o símbolo lido da entrada
  - assume um *único* estado
- não determinístico
  - para o estado corrente e o símbolo lido da entrada
  - assume um estado pertencente a um *conjunto* de estados alternativos

- com movimentos vazios
  - para o estado corrente e, independentemente de ler um símbolo ou não da entrada,
  - assume um estado pertencente a um *conjunto* de estados alternativos
  - portanto é não determinístico
  - é dito movimento vazio se muda de estado sem uma leitura de símbolo

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

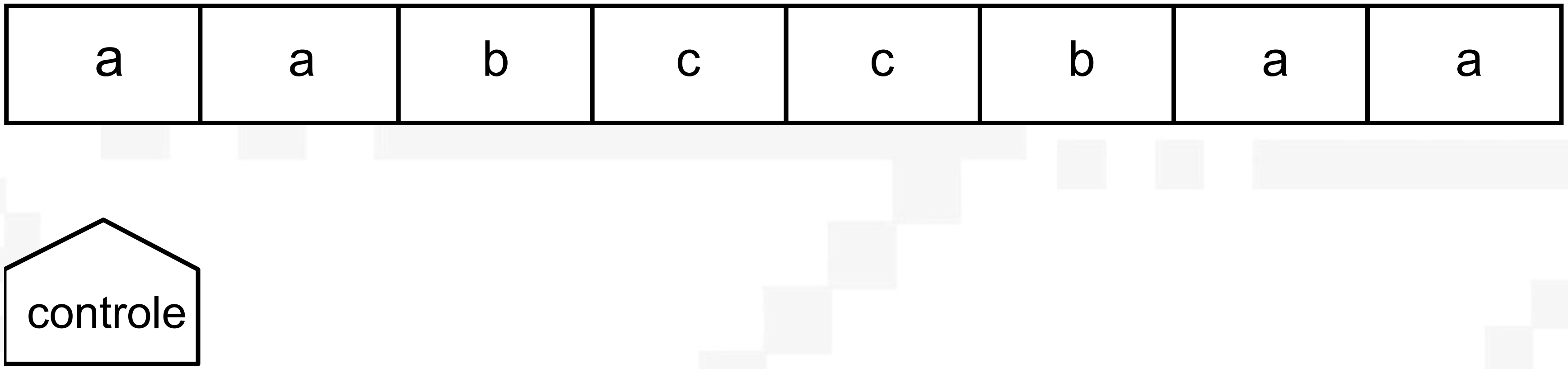
- Fita: dispositivo de entrada
  - contém informação a ser processada
- Unidade de controle: reflete o estado corrente da máquina
  - possui unidade de leitura (cabeça da fita)
  - acessa uma célula da fita de cada vez
  - movimenta-se exclusivamente para a direita
- Programa, função programa ou função 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 um símbolo da fita 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

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

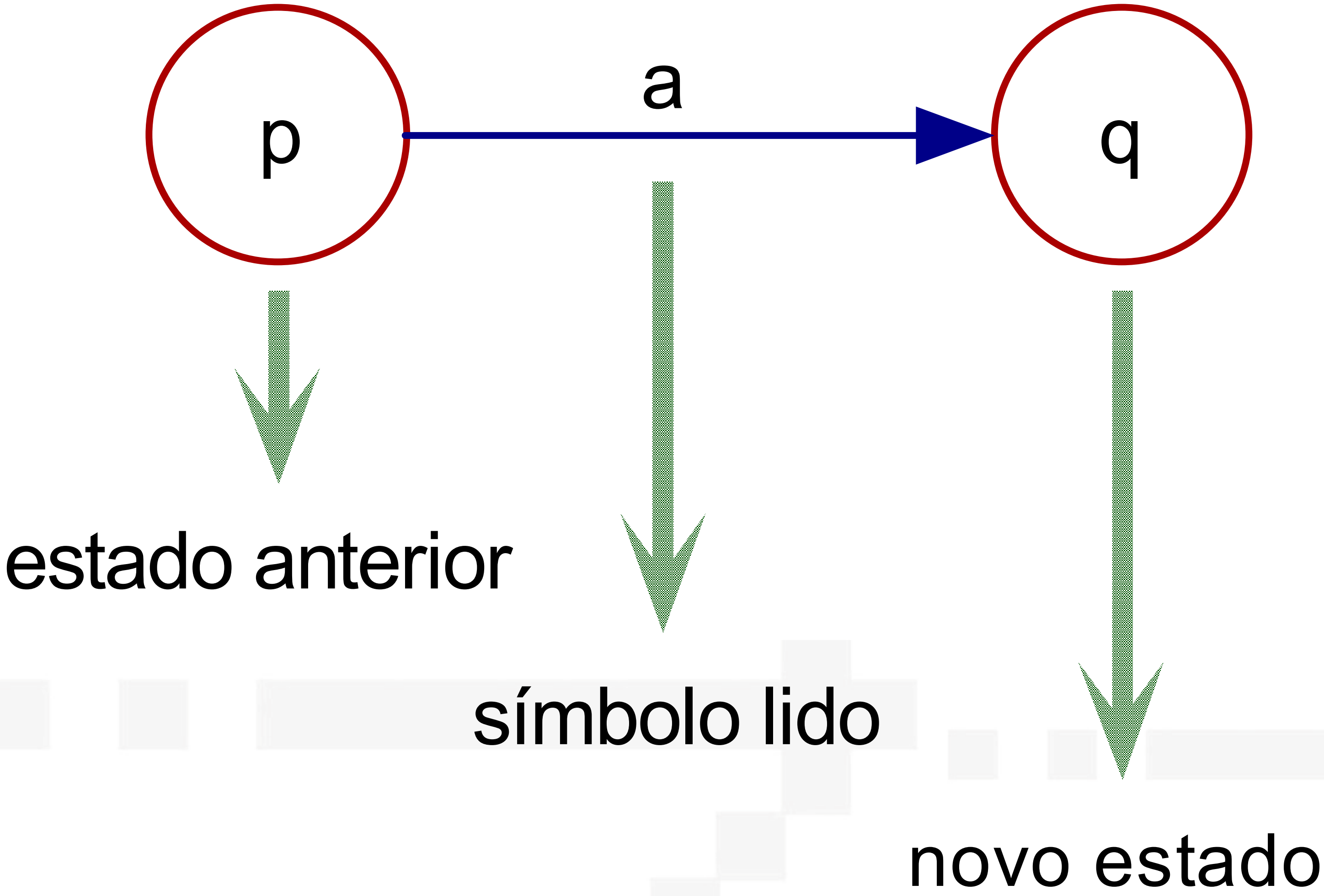
- Σ é um alfabeto (de símbolos) de entrada
- Q é um conjunto de estados possíveis do autômato (finito)
- δ é uma (função) programa ou função de transição (função parcial)

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

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

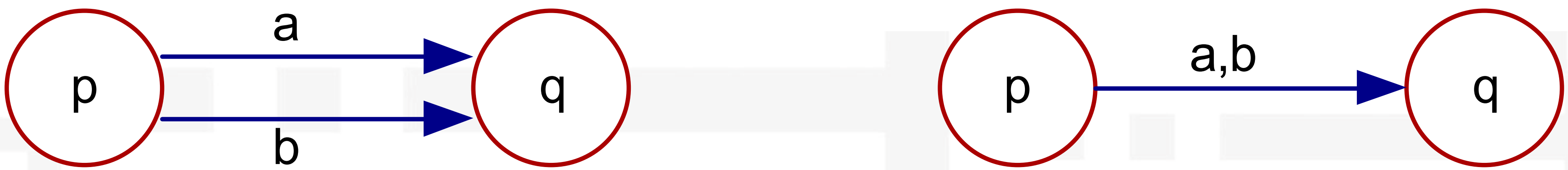
- q_0 é um elemento distinguido de Q: estado inicial
- F é um subconjunto de Q: conjunto de estados finais

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 de que um autômato finito

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

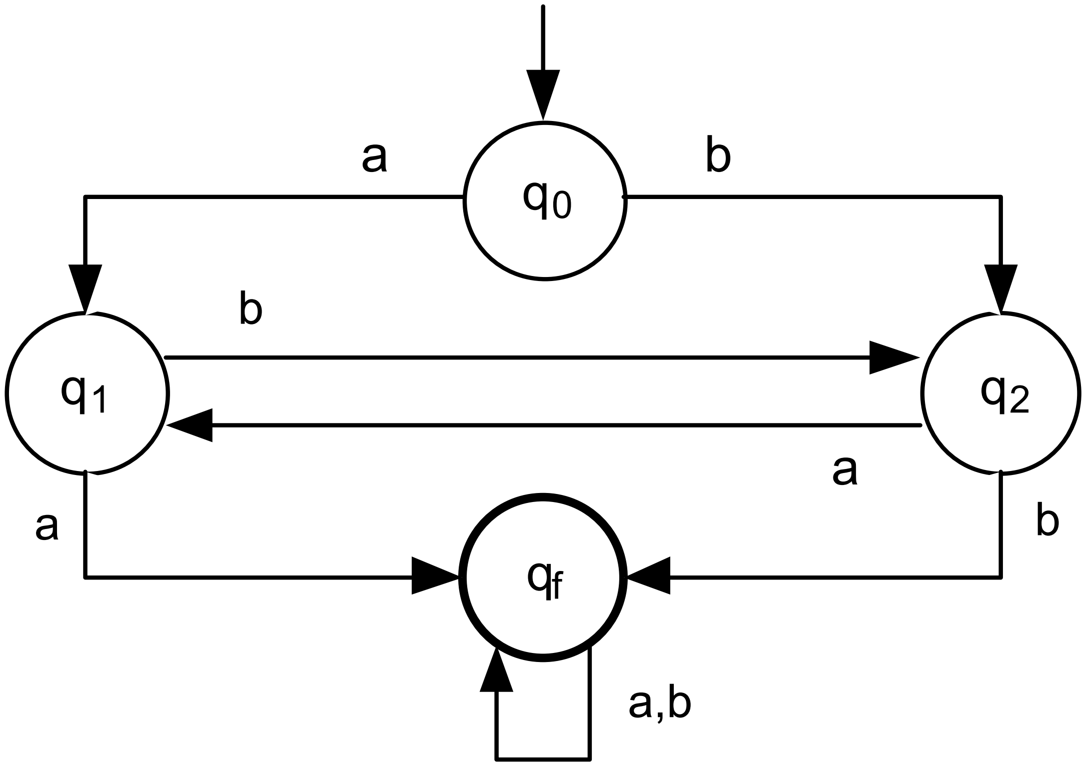
Exp: Autômato finito: aa ou bb como subpalavra

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

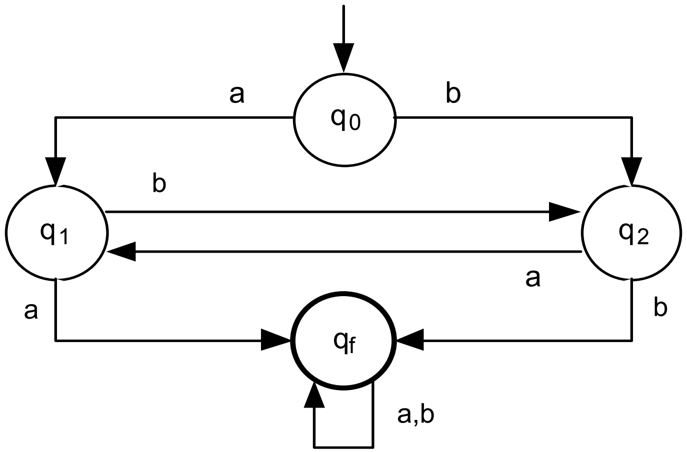
Autômato finito

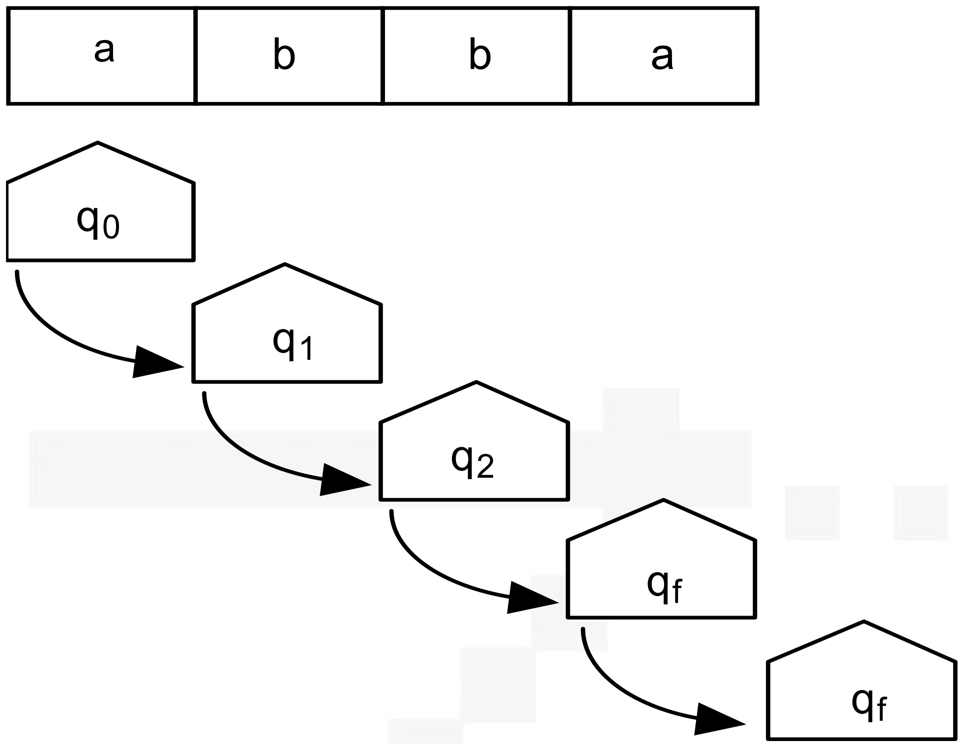
$M_1 = (\{ a, b \}, \{ q_0, q_1, q_2, q_f \}, \delta_1, q_0, \{ q_f \})$

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


a
q0
b
b
q1
q2
a
a
b
qf
a,b


a
b
b
a
q0
q1
q2
qf
qf

Obs: Autômato finito sempre para

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 $\times$ 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 onde

- 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

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

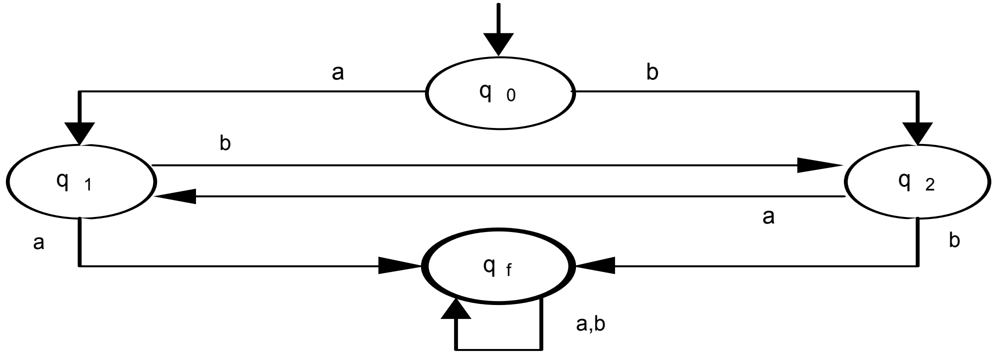
é $\delta: Q \times \Sigma \to Q$ estendida para palavras - indutivamente definida

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

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 *a*baa
- $\underline{\delta}^*(q_1, baa) =$ função estendida sobre baa
- $\underline{\delta}^*(\delta(q_1, b), aa) =$ processa *b*aa
- $\underline{\delta}^*(q_2, aa) =$ função estendida sobre aa
- $\underline{\delta}^*(\delta(q_2, a), a) =$ processa *a*a
- $\underline{\delta}^*(q_1, a) =$ função estendida sobre a
- $\underline{\delta}^*(\delta(q_1, a), \varepsilon) =$ processa *a*
- $\underline{\delta}^*(q_f, \varepsilon) = q_f$ 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

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

Linguagem rejeitada por M:

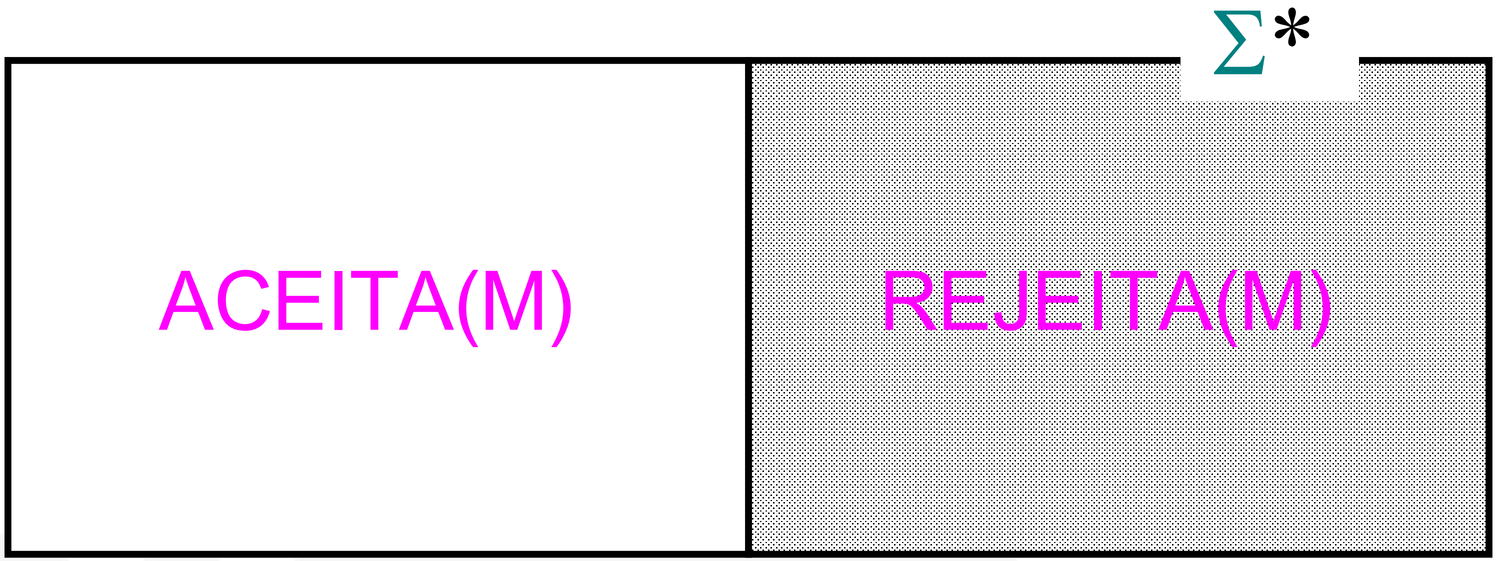
$$REJEITA(M) = \{ w \mid \delta^*(q_0, w) \notin F \text{ ou } \delta^*(q_0, w) \text{ é indefinida} \}$$

Supondo que Σ^* é o conjunto universo

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

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,

$$\text{ACEITA}(M_1) = \text{ACEITA}(M_2)$$

Def: Linguagem regular, linguagem tipo 3

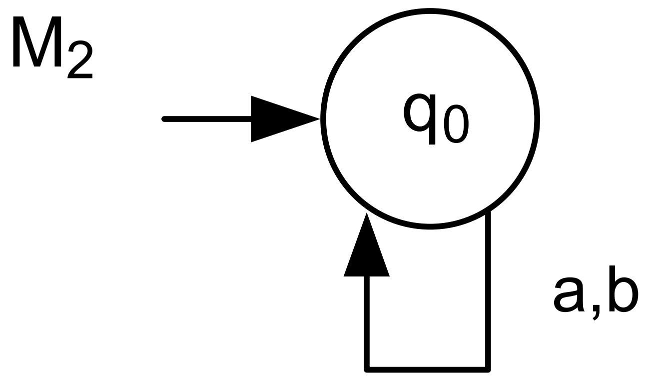
L é uma linguagem regular ou linguagem tipo 3

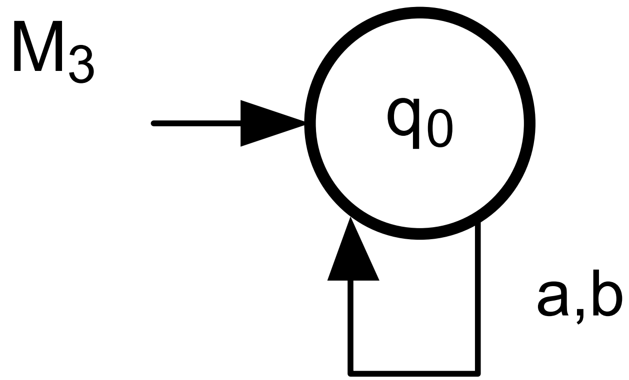
- existe pelo menos um autômato finito determinístico que aceita L

Exp: …Autômato finito: vazia, todas as palavras

Linguagens sobre o alfabeto { a, b }

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





Exp: Autômato finito: vazia, todas as palavras

$L_2 = \varnothing$ e $L_3 = \Sigma^*$

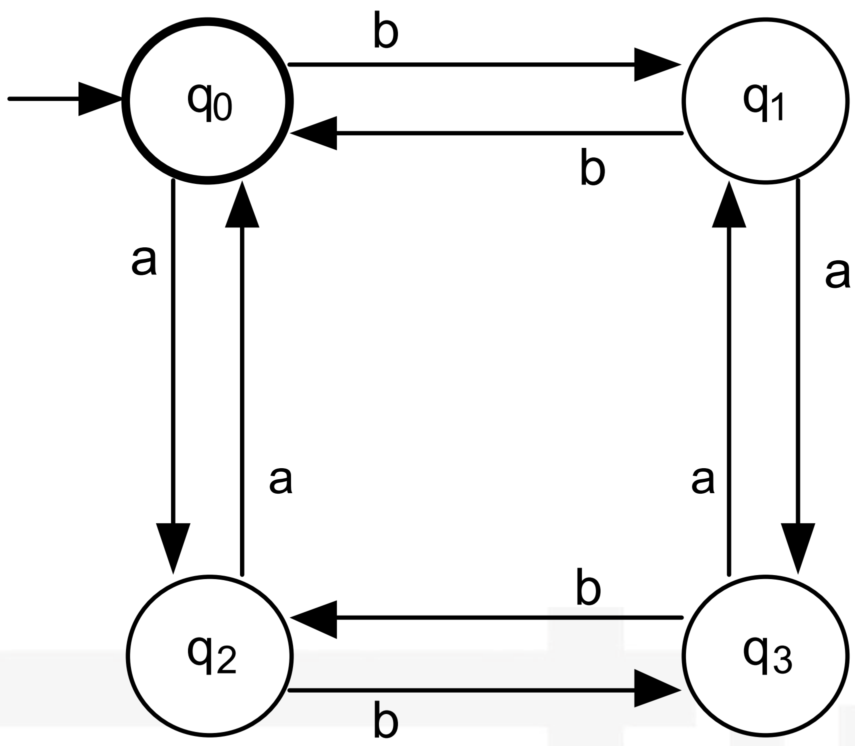
δ_2	a	b
q_0	q_0	q_0

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

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



Como seria para aceitar um número ímpar de cada símbolo?

Obs: Função programa $\times$ função programa estendida

Objetivando simplificar a notação

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

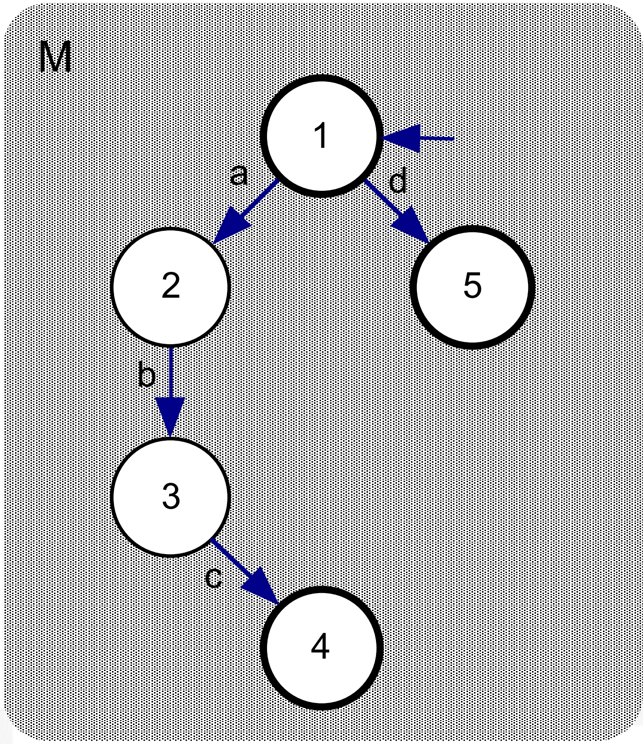
Obs: Computações $\times$ caminhos de um grafo

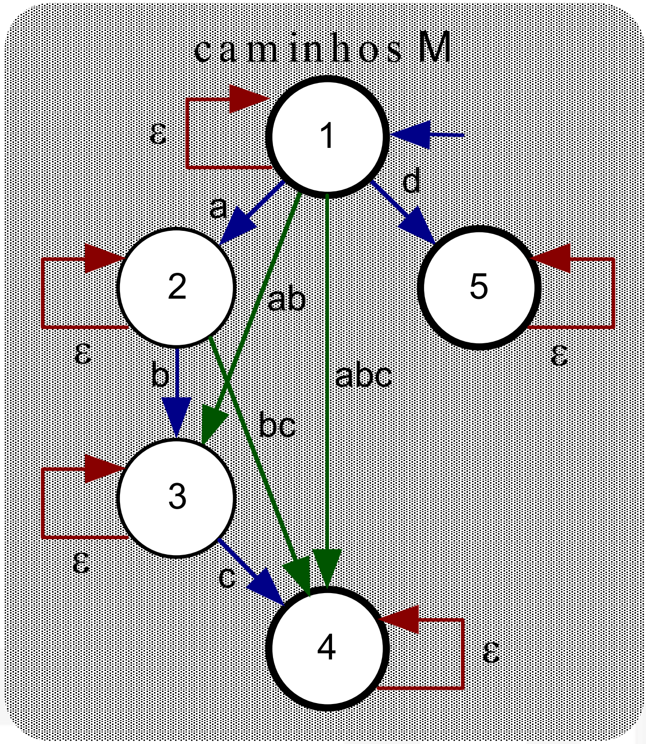
Existe uma forte relação entre as computações de um autômato finito e os caminhos do correspondente grafo finito direto

Dado um autômato, o enriquecimento do correspondente grafo com todos os caminhos (incluindo os de tamanho zero)

- conjunto de todos arcos (caminhos): computações possíveis
- linguagem aceita: subconjunto de arcos
  - ∗ com origem no estado inicial, destino em algum estado final
- linguagem rejeitada: subconjunto de arcos
  - ∗ com origem no estado inicial, destino em algum estado não final

Obs: ...Computações × caminhos de um grafo



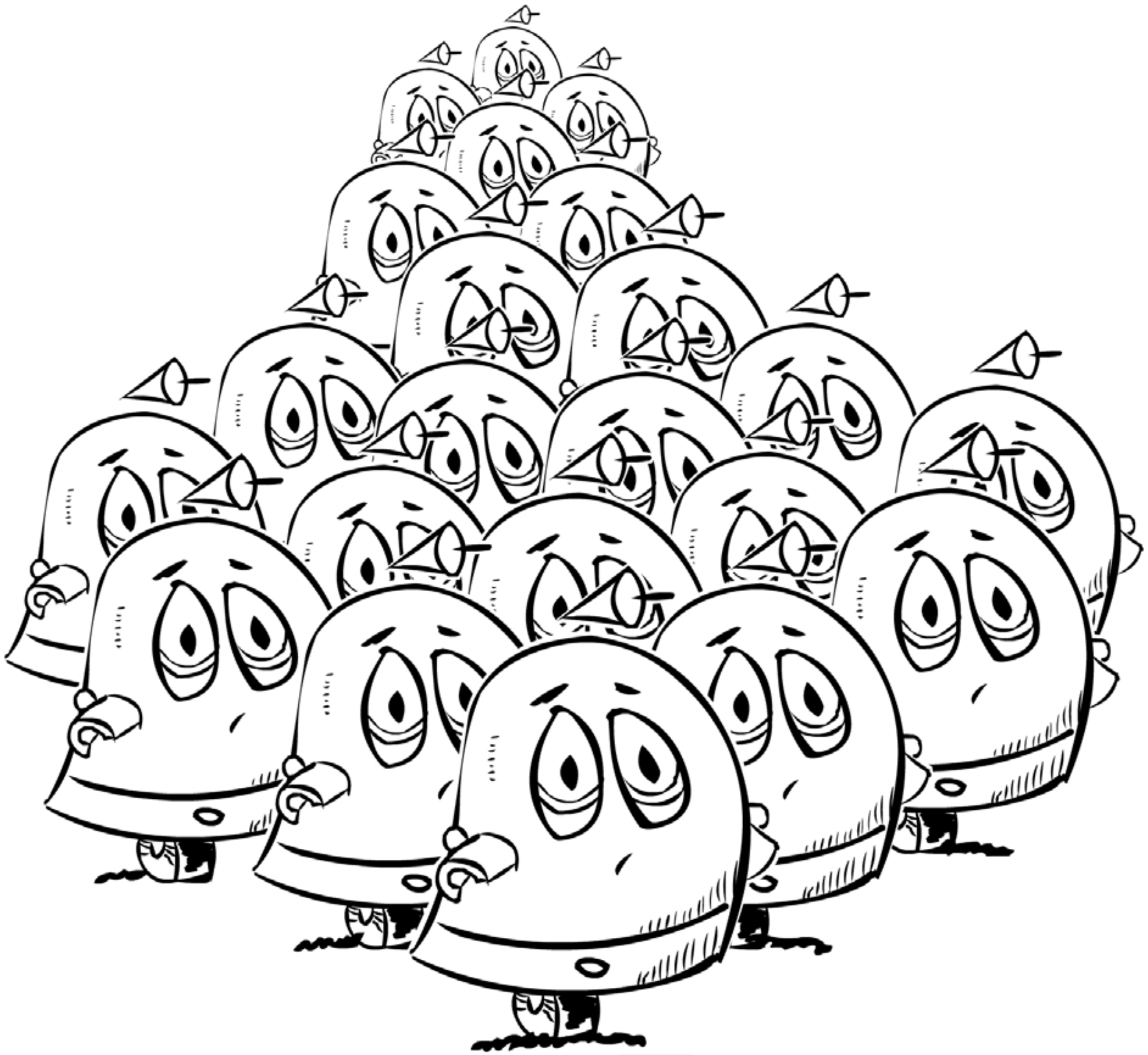


Computações(M)
=
{ε, a, b, c, d,
ab, bc, abc}

ACEITA (M)
=
{ε , d, abc}

3 – Linguagens Regulares

3.1 Sistema de Estados Finitos
3.2 Composição Sequencial, 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
- não confundir com a 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

para o estado corrente e o símbolo lido da entrada,
determina aleatoriamente um estado de um conjunto de estados alternativos.

Assim, a cada transição não determinista

- novos caminhos alternativos são possíveis
- definindo uma árvore de opções

Entrada aceita

- se pelo menos um dos caminhos alternativos aceita a entrada
- mesmo que os demais não aceitem

Semântica adotada para o não determinismo

- assume um conjunto de estados alternativos

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

- como se todos os caminhos alternativos fossem investigados simultaneamente
  - o processamento de um caminho
  - não influi no estado, símbolo lido e posição da cabeça
  - dos demais caminhos alternativos

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 total) programa ou função de transição (função total)

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

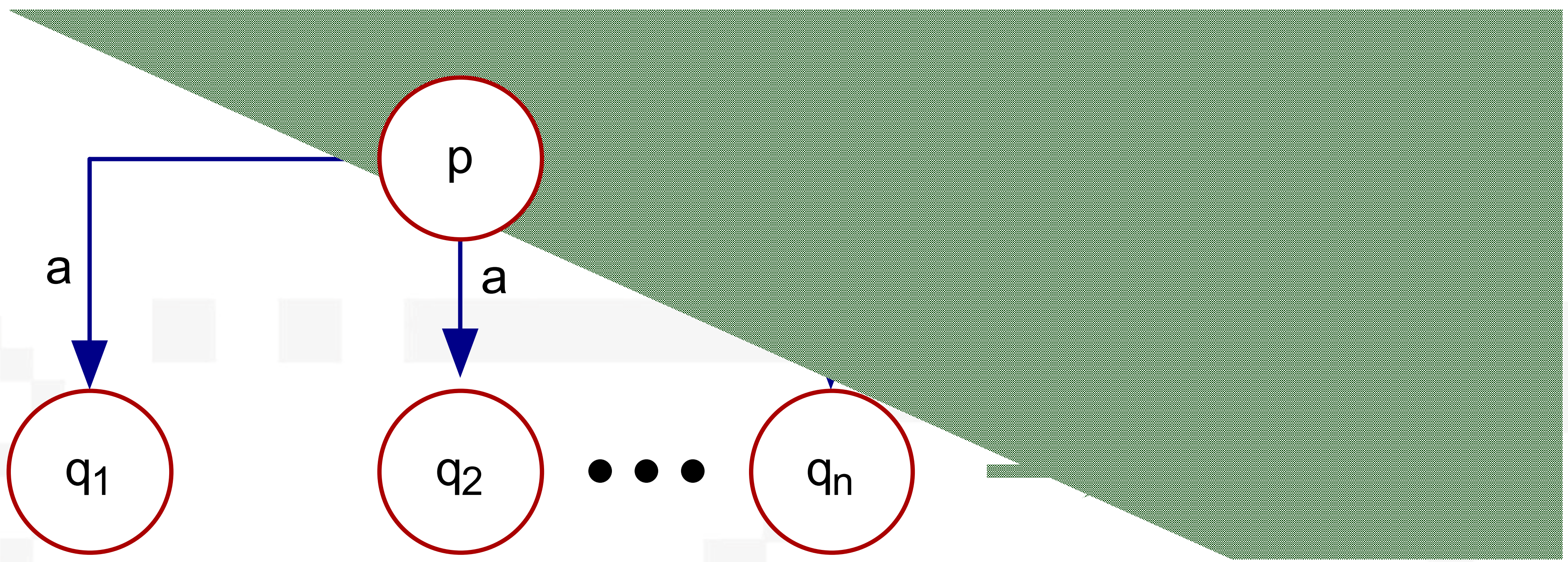
  * transição: $\delta(p, a) = \{ q_1, q_2, \ldots, q_n \}$
- q_0 é um elemento distinguido de Q: estado inicial
- F é um subconjunto de Q: conjunto de estados finais

Se $\delta(p, a) = \varnothing$

- transição é indefinida para o par (p, a)
- o autômato para, rejeitando a entrada

Autômato como diagrama

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



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

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

indutivamente definida

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

Transição estendida (a um conjunto de estados)

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

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*
  - conjunto de estados alternativos atingido é vazio: o autômato para por indefinição

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

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

Linguagem rejeitada por M

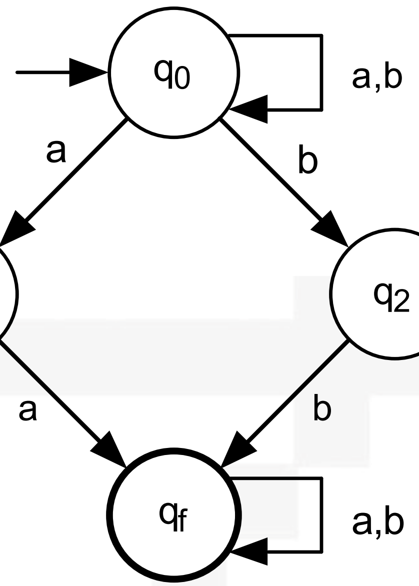
$$REJEITA(M) = \{ w \mid \delta^*(\{ q_0 \}, w) \cap F = \varnothing \text{ ou } \delta^*(\{ q_0 \}, w) \text{ é indefinida} \}$$

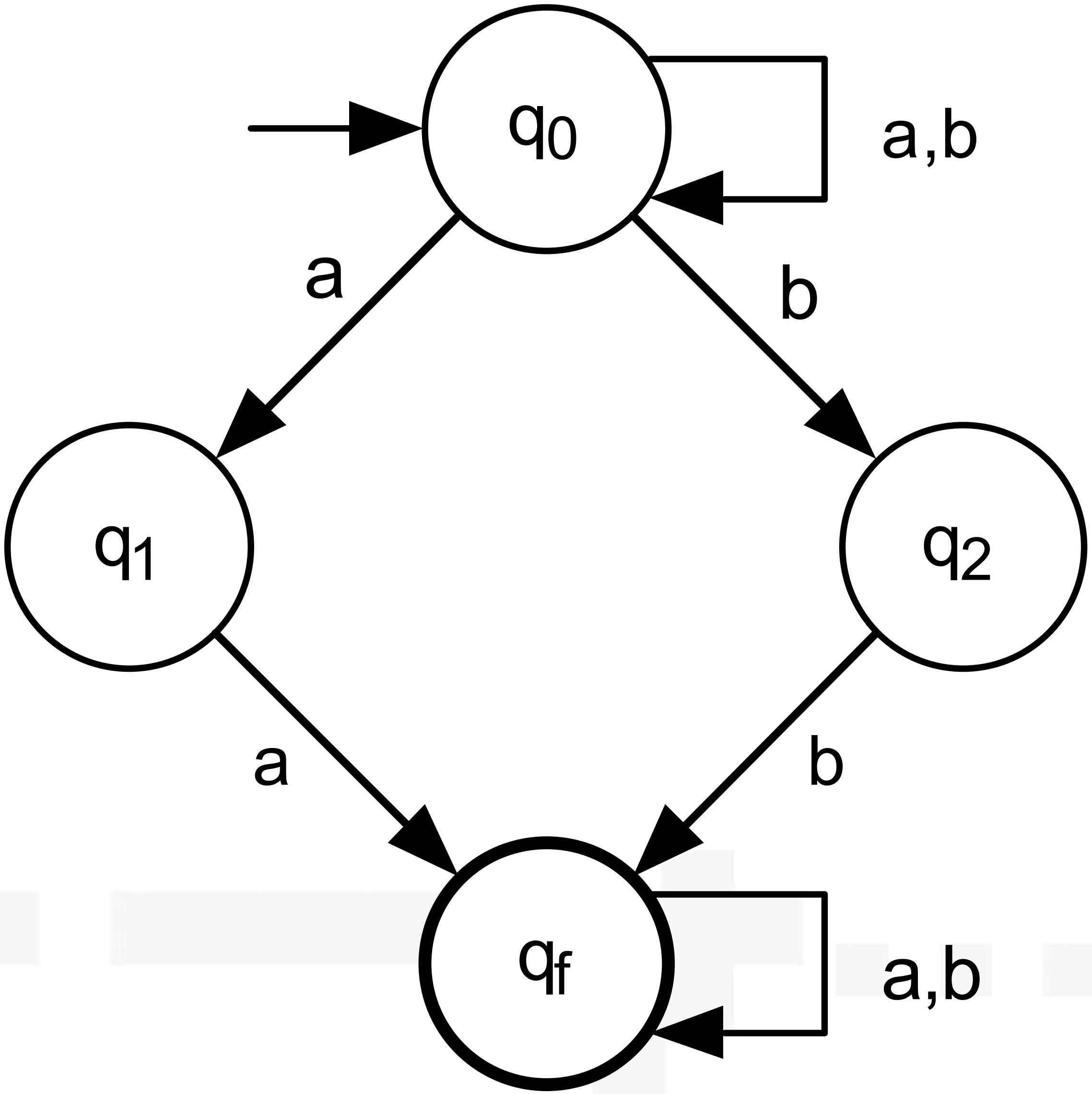
Exp: Autômato finito não determinístico: aa ou bb como subpalavra

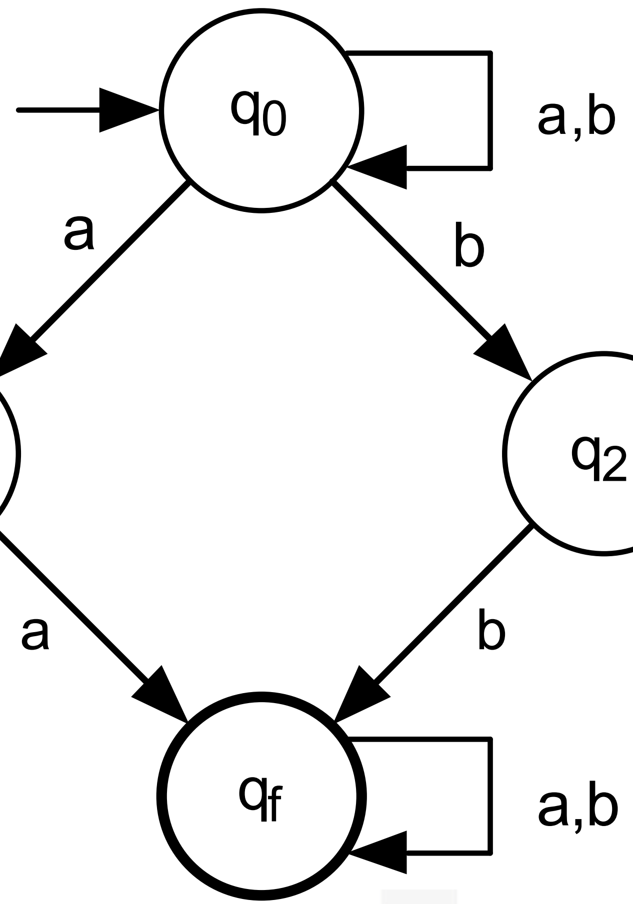
$L_5 = \{ w \mid w$ possui aa ou bb como subpalavra $\}$

Autômato finito não determinístico:

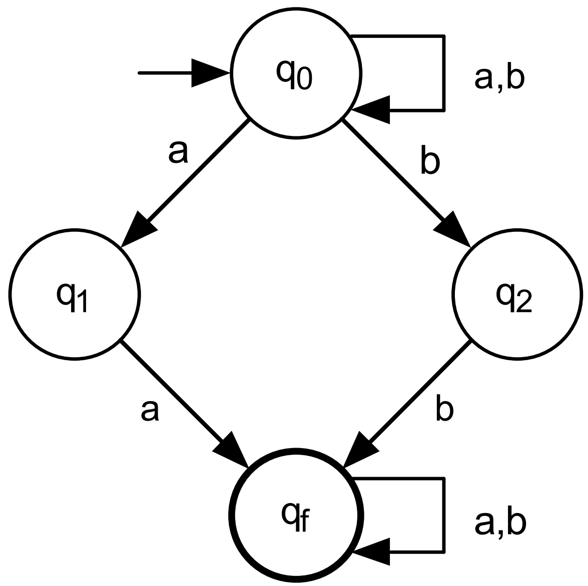
$$M_5 = (\{ a, b \}, \{ q_0, q_1, q_2, q_f \}, \delta_5, q_0, \{ q_f \})$$




q0
a,b
a
b
q1
q2
a
b
qf
a,b



- 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


q0
a,b
a
b
q1
q2
a
b
qf
a,b

δ_5	a	b
q_0	$\{ q_0, q_1 \}$	$\{ q_0, q_2 \}$
q_1	$\{ q_f \}$	-
q_2	-	$\{ q_f \}$
q_f	$\{ q_f \}$	$\{ q_f \}$

- $\underline{\delta}^*(\{ q_0 \}, abaa) =$ função estendida sobre abaa
- $\underline{\delta}^*(\delta(q_0, a), baa) =$ processa *abaa*
- $\underline{\delta}^*(\{ q_0, q_1 \}, baa) =$ função estendida sobre baa
- $\underline{\delta}^*(\delta(q_0, b) \cup \delta(q_1, b), aa) =$ processa *baa*
- $\underline{\delta}^*(\{ q_0, q_2 \} \cup \varnothing, aa) =$
- $\underline{\delta}^*(\{ q_0, q_2 \}, aa) =$ função estendida sobre aa

- $\underline{\delta}^*(\delta(q_0, a) \cup \delta(q_2, a), a) =$ processa *aa*
- $\underline{\delta}^*(\{ q_0, q_1 \} \cup \varnothing, a) =$
- $\underline{\delta}^*(\{ q_0, q_1 \}, a) =$ função estendida sobre *a*
- $\underline{\delta}^*(\delta(q_0, a) \cup \delta(q_1, a), \varepsilon) =$ processa *a*
- $\underline{\delta}^*(\{ q_0, q_1 \} \cup \{ q_f \}, \varepsilon) =$
- $\underline{\delta}^*(\{ q_0, q_1, q_f \}, \varepsilon) = \{ q_0, q_1, q_f \}$ função estendida sobre ε: fim da indução

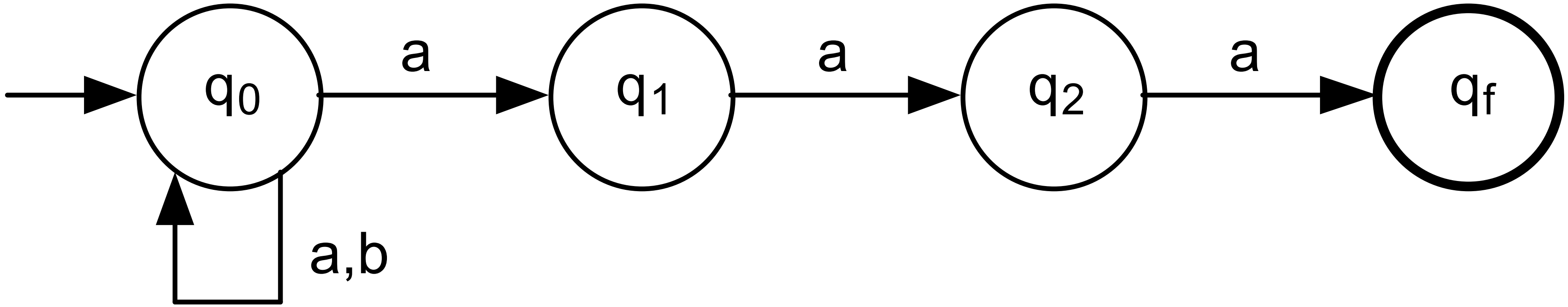
e, portanto, a palavra é aceita, pois $\{ q_0, q_1, q_f \} \cap F = \{ q_f \} \neq \varnothing$

Exp: AFN: aaa como sufixo

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

Autômato finito não determinístico:

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


a
a
a
q0
q1
q2
qf
a,b

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

e, portanto, a palavra é rejeitada, pois $\{ q_0, q_1, q_2 \} \cap F = \varnothing$

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 realize 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 $\langle q_1 q_2 \ldots q_n \rangle$
  - ∗ ordem não distingue combinações: $\langle q_u q_v \rangle = \langle q_v q_u \rangle$
  - ∗ imagem de todos os estados alternativos de M

- $\delta_D: Q_D \times \Sigma \to Q_D$

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

- ∗ em particular:

$\delta_D(\langle q_1 \ldots q_n \rangle, a)$ é indefinida sse $\delta^*(\{ q_1, \ldots, q_n \}, a) = \varnothing$

- $\langle q_0 \rangle$ – estado inicial

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

AFD M_D simula as computações do AFN M ???

- indução no tamanho da palavra
- mostrar que

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

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:

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

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

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

- equivale (hipótese de indução)

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

- 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

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

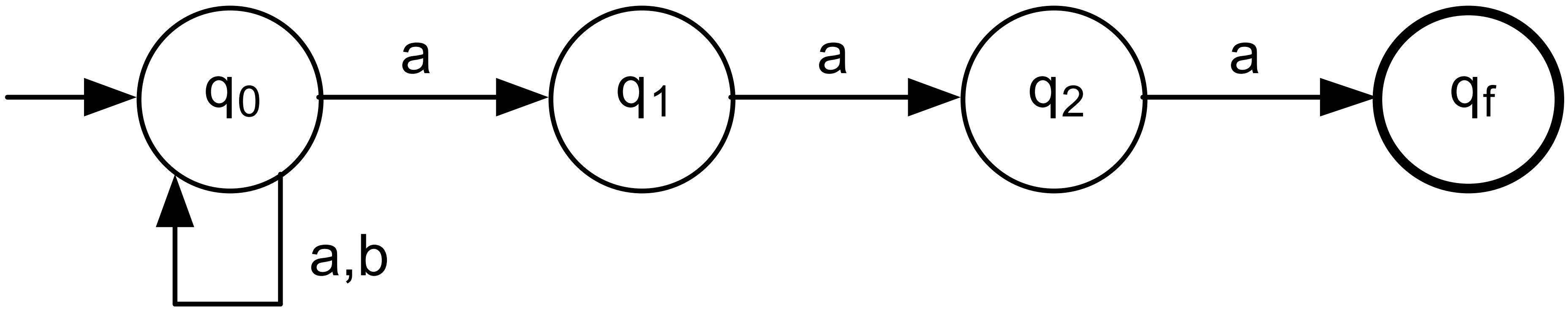
- solução determinista: não é trivial; número grande de estados
- solução não determinista: bem simples; poucos estados

Alternativa para construir um AFD

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

Exp: AFN $\rightarrow$ AFD

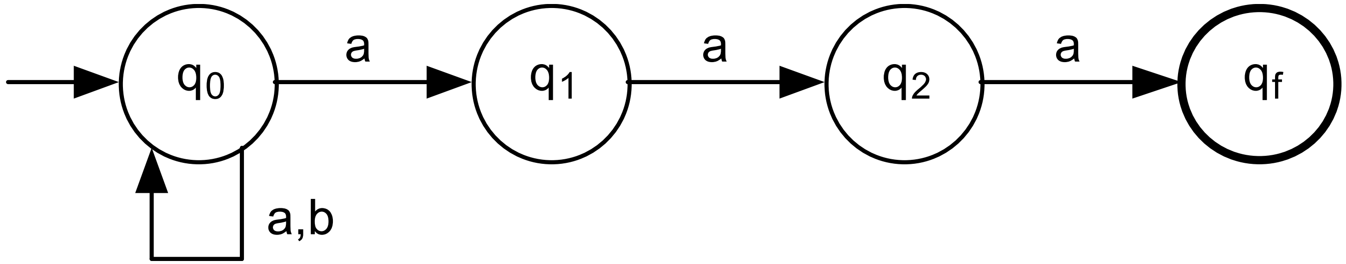
$$M_6 = (\{ a, b \}, \{ q_0, q_1, q_2, q_f \}, \delta_6, q_0, \{ q_f \})$$



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

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

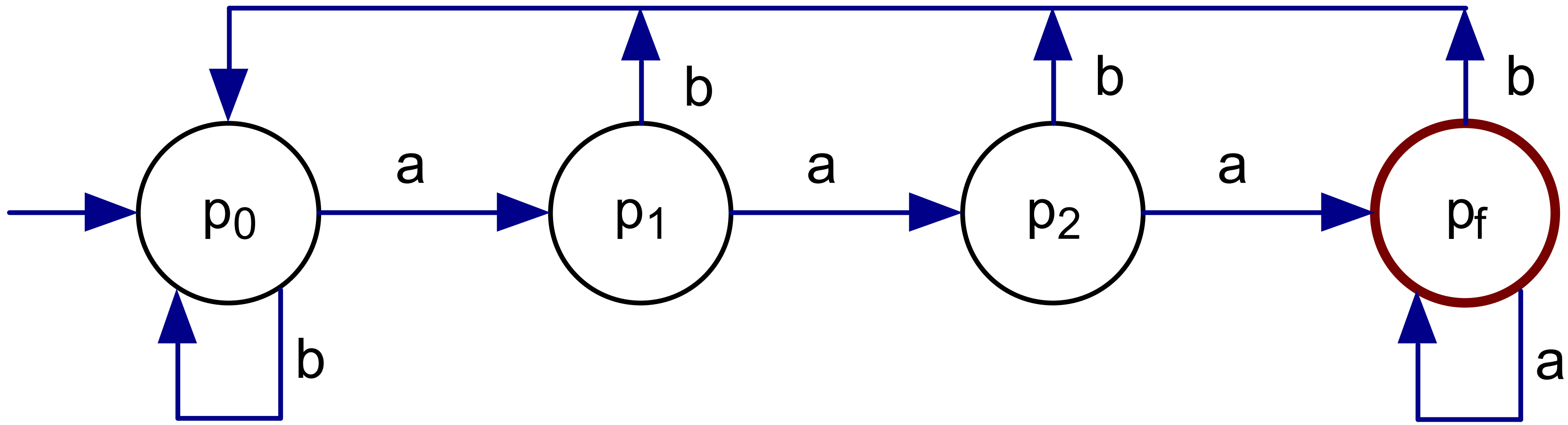
AFN



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
$p_0 = \langle q_0 \rangle$	$\langle q_0 q_1 \rangle$	$\langle q_0 \rangle$
$p_1 = \langle q_0 q_1 \rangle$	$\langle q_0 q_1 q_2 \rangle$	$\langle q_0 \rangle$
$p_2 = \langle q_0 q_1 q_2 \rangle$	$\langle q_0 q_1 q_2 q_f \rangle$	$\langle q_0 \rangle$
$p_f = \langle q_0 q_1 q_2 q_f \rangle$	$\langle q_0 q_1 q_2 q_f \rangle$	$\langle q_0 \rangle$



p_0
p_1
p_2
p_f
a
a
a
b
b
b
b
a

3 – Linguagens Regulares

3.1 Sistema de Estados Finitos
3.2 Composição Sequencial, 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 para autômatos finitos

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

Def: Autômato finito com movimentos vazios – AFNε

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

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

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

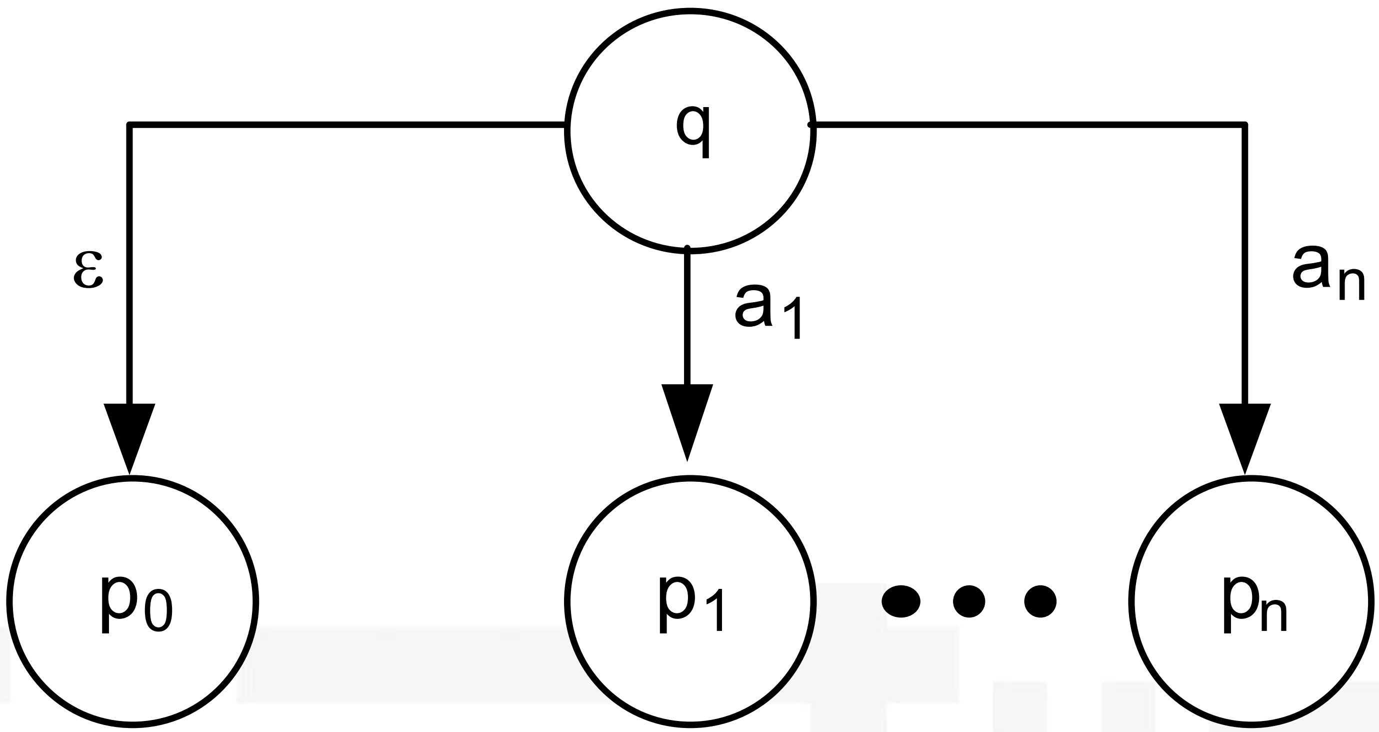
∗ movimento vazio ou transição vazia

$$\delta(p, \varepsilon) = \{ q_1, q_2, \ldots, q_n \}$$

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

Autômato como diagrama

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



Computação de um AFNε

- análoga à de um AFN

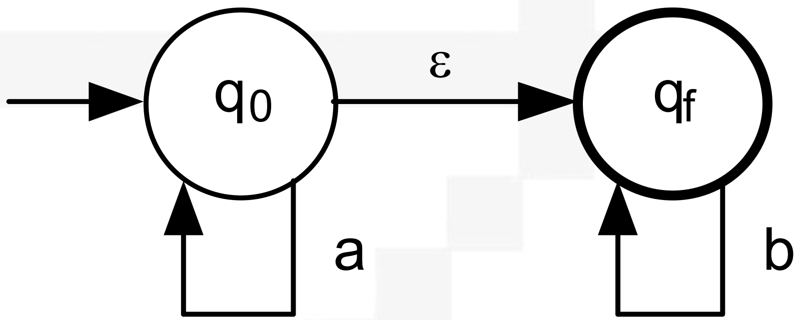
Processamento de uma transição vazia

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

Exp: AFNε: 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)

$$\delta\varepsilon: Q \to 2^Q$$

- 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)

$$\delta\varepsilon^*: 2^Q \to 2^Q$$

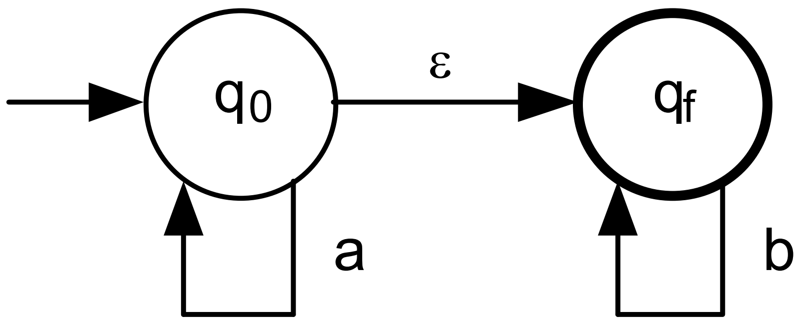
- tal que

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

Por simplicidade, $\delta\varepsilon$ e $\delta\varepsilon^*$

- ambas denotadas por $\delta\varepsilon$

Exp: Computação vazia



- $\delta\varepsilon(q_0) = \{ q_0, q_f \}$
- $\delta\varepsilon(q_f) = \{ q_f \}$
- $\delta\varepsilon(\{ q_0, q_f \}) = \{ q_0, q_f \}$

Computação de um AFNε 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

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

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

indutivamente definida

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

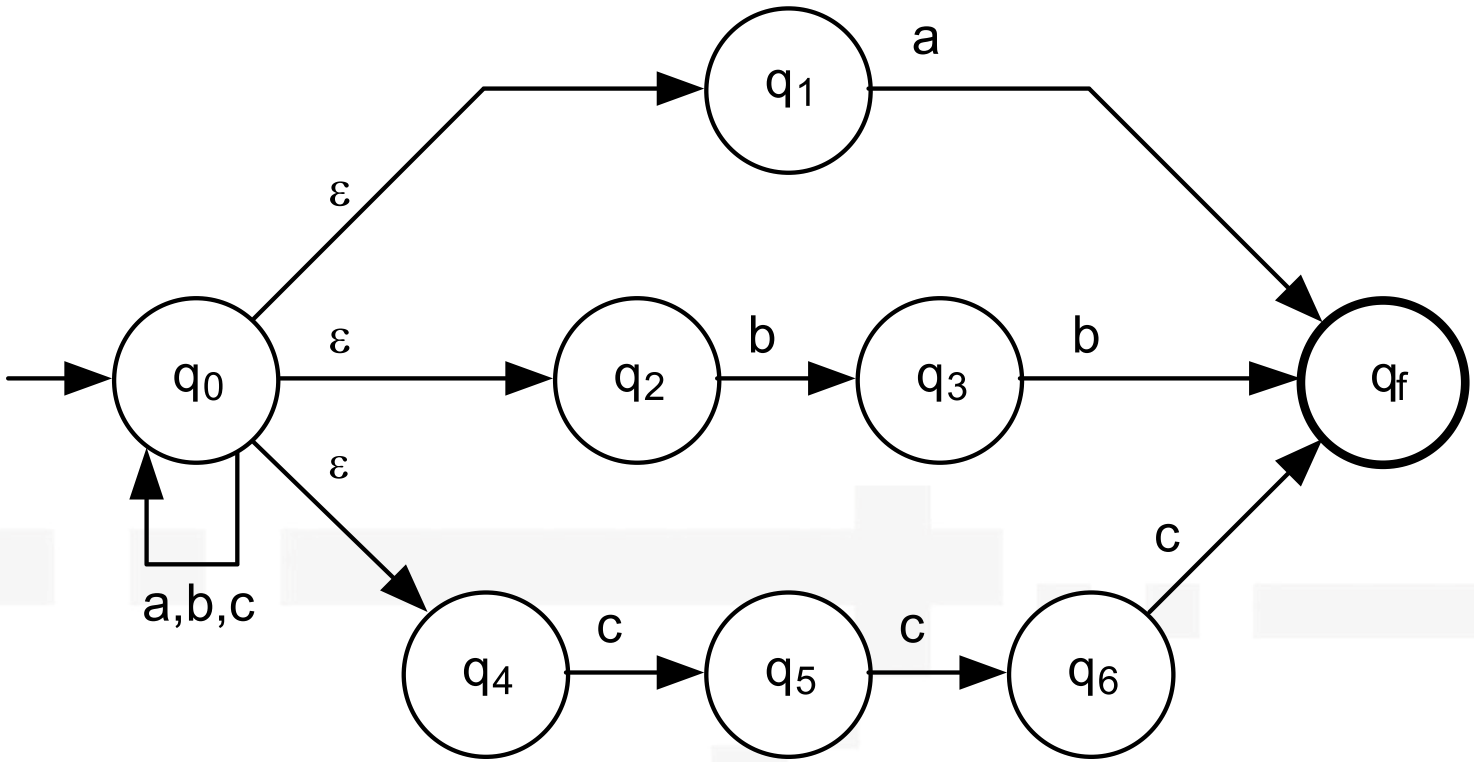
Parada do processamento, ling. aceita/rejeitada

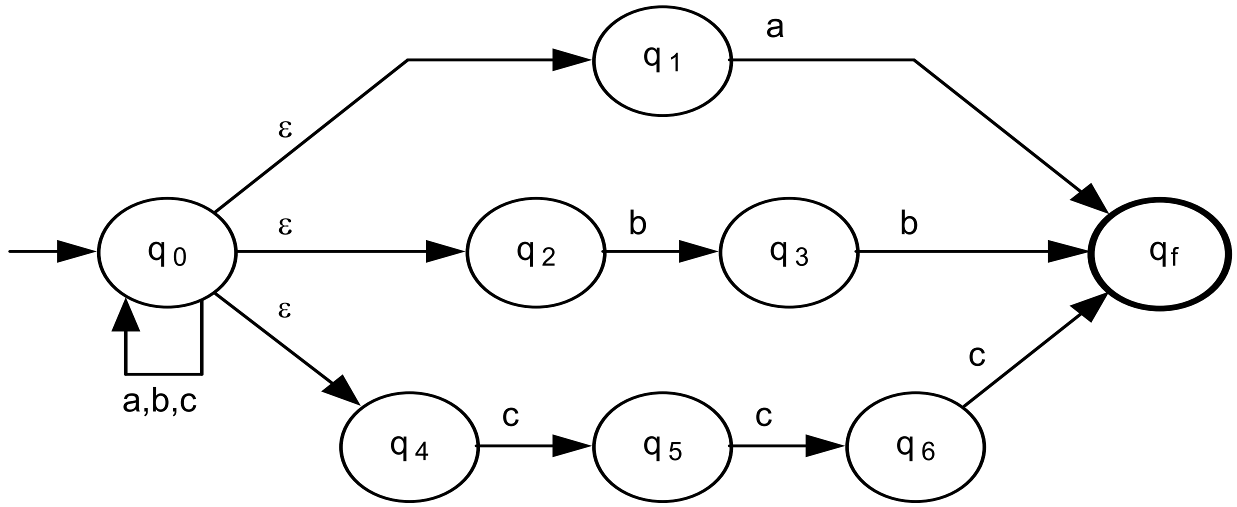
- análoga à do autômato finito não determinístico

Exp: Computação vazia, computação

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

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





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

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

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

Como:

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

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

$\delta^*(\{ q_0 \}, ab) = \{ q_0, q_1, q_2, q_3, q_4 \}$ 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 AFNε

Classe dos autômatos finitos com movimentos vazios é equivalente à Classe dos autômatos finitos não determinísticos

<u>Prova:</u> (*por indução*)

Mostrar que

- a partir de um AFNε M qualquer
- construir um AFN M_N que realize as mesmas computações
- M_N simula M

AFNε → 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 AFNε qualquer. AFN construído

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

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

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

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

$$\delta\varepsilon(q) \cap F \neq \varnothing$$

  * estados que atingem estados finais via computações vazias

Demonstração que, de fato, o AFN M_N simula o AFNε M

- indução no tamanho da palavra
- exercício

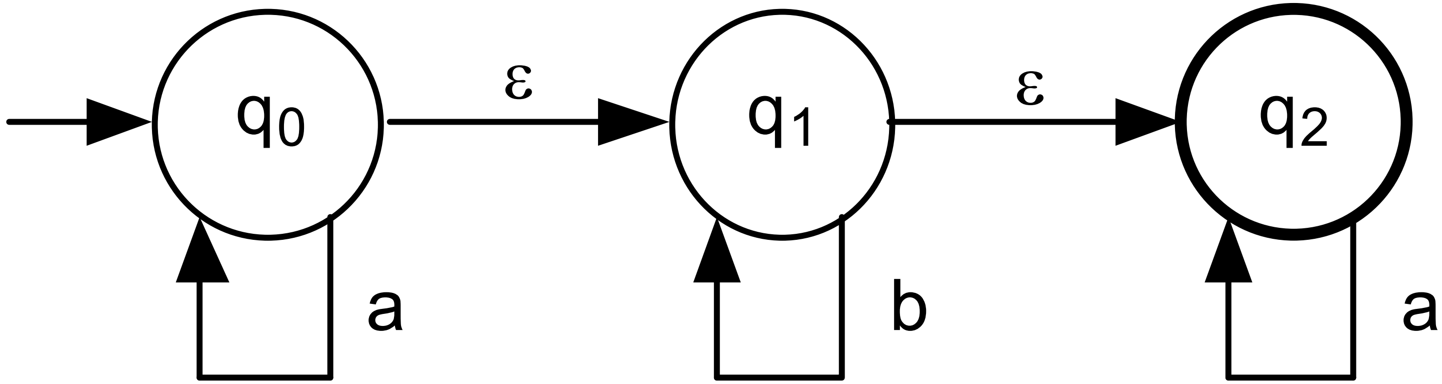
Portanto, linguagem aceita por AFNε

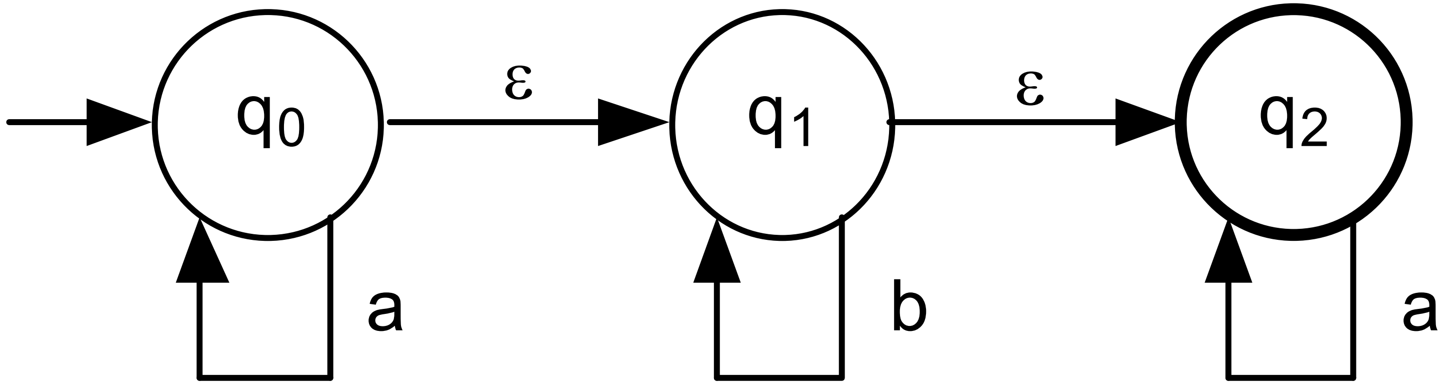
- é linguagem regular ou tipo 3

Exp: Construção de um AFN a partir de um AFNε

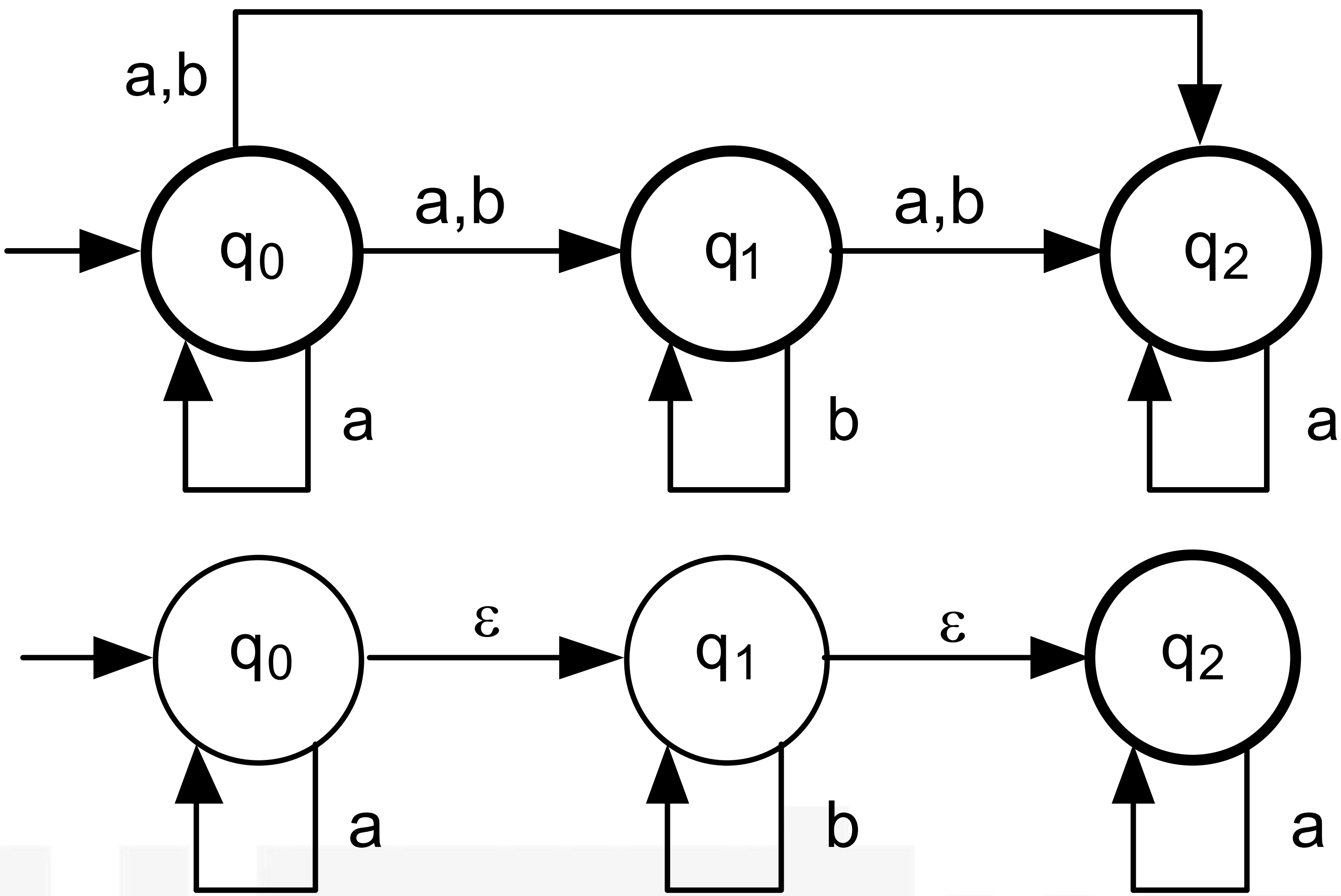
AFNε - $M_9 = (\{ a, b \}, \{ q_0, q_1, q_2 \}, \delta_9, q_0, \{ q_2 \})$

δ_9	a	b	ε
q_0	$\{ q_0 \}$	-	$\{ q_1 \}$
q_1	-	$\{ q_1 \}$	$\{ q_2 \}$
q_2	$\{ q_2 \}$	-	-





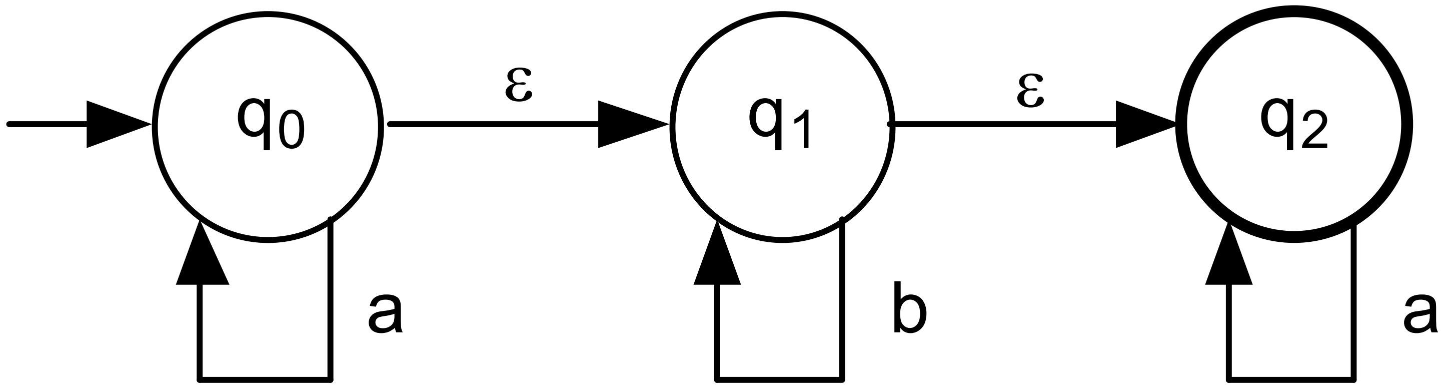
$M_{9N} = (\{ a, b \}, \{ q_0, q_1, q_2 \}, \delta_{9N}\ q_0, F_N)$



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

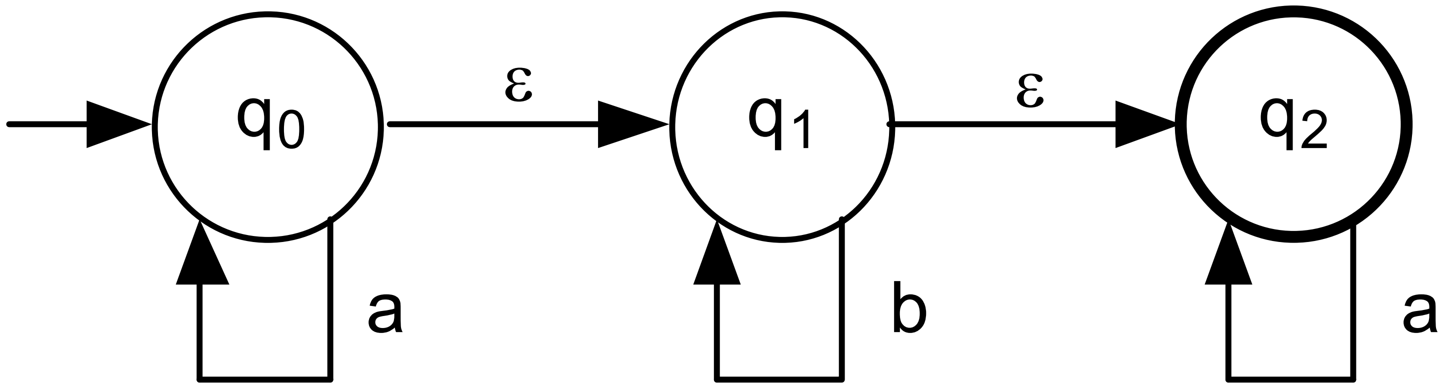
- $\delta\varepsilon(q_0) = \{ q_0, q_1, q_2 \}$

- $\delta\varepsilon(q_1) = \{ q_1, q_2 \}$
- $\delta\varepsilon(q_2) = \{ q_2 \}$



Na construção de δ_{9_N}

- $\underline{\delta}_9{}^*(\{ q_0 \}, \varepsilon) = \{ q_0, q_1, q_2 \}$
- $\underline{\delta}_9{}^*(\{ q_1 \}, \varepsilon) = \{ q_1, q_2 \}$
- $\underline{\delta}_9{}^*(\{ q_2 \}, \varepsilon) = \{ q_2 \}$



Assim, δ_{9N} é tal que

$$\delta_{9N}(q_0, a) = \delta_9^*(\{ q_0 \}, a) =$$

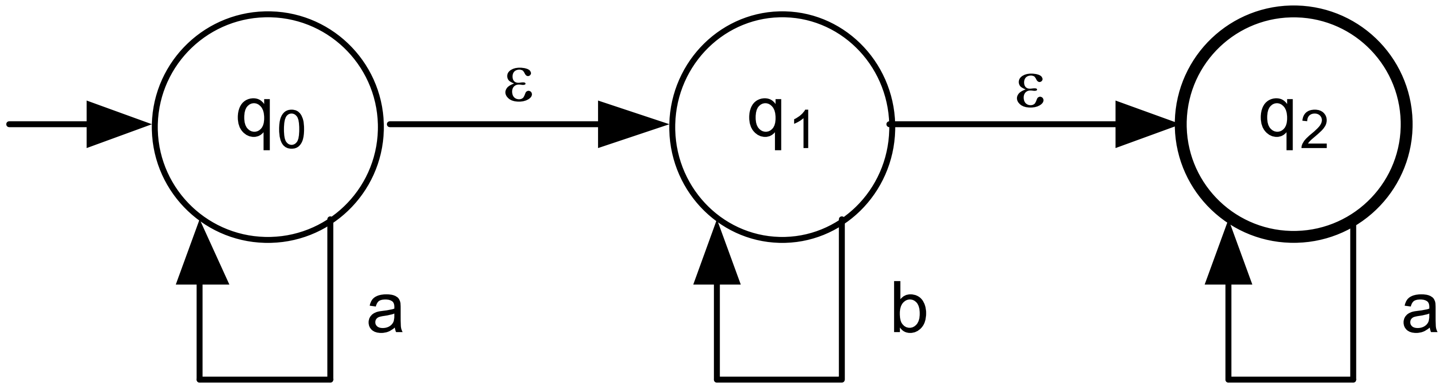
$$\delta\varepsilon(\{ r \mid r \in \delta(s, a) \text{ e } s \in \underline{\delta}^*(\{ q_0 \}, \varepsilon) \}) = \{ q_0, q_1, q_2 \}$$

$$\delta_{9N}(q_0, b) = \delta_9^*(\{ q_0 \}, b) =$$

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

$$\delta_{9N}(q_1, a) = \delta_9^*(\{ q_1 \}, a) =$$

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



- $\delta_{9N}(q_1, b) = \underline{\delta_9}^*(\{ q_1 \}, b) = \delta\varepsilon(\{ r \mid r \in \delta(s, b) \text{ e } s \in \underline{\delta}^*(\{ q_1 \}, \varepsilon) \}) = \{ q_1, q_2 \}$
- $\delta_{9N}(q_2, a) = \underline{\delta_9}^*(\{ q_2 \}, a) = \delta\varepsilon(\{ r \mid r \in \delta(s, a) \text{ e } s \in \underline{\delta}^*(\{ q_2 \}, \varepsilon) \}) = \{ q_2 \}$
- $\delta_{9N}(q_2, b) = \underline{\delta_9}^*(\{ q_2 \}, b) = \delta\varepsilon(\{ r \mid r \in \delta(s, b) \text{ e } s \in \underline{\delta}^*(\{ q_2 \}, \varepsilon) \})$ é indefinida

3 – Linguagens Regulares

3.1 Sistema de Estados Finitos
3.2 Composição Sequencial, 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 $\{ \varepsilon \}$
- 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 $R\,S = \{ uv \mid u \in R \text{ e } v \in S \}$
- *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
$(a + \varepsilon)(b + ba)^*$	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 se for possível construir um

- AFD, AFN ou AFNε que reconheça a linguagem

É necessário mostrar que

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

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

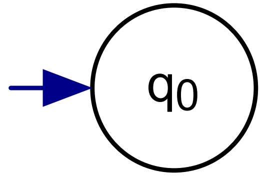
Demonstração: indução no número de operadores

Base de indução. r ER com zero operadores

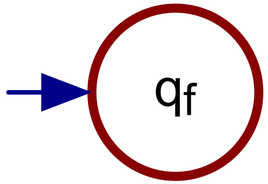
- $r = \emptyset$
  - Autômato???
- $r = \varepsilon$
  - Autômato???
- $r = x \; (x \in \Sigma)$
  - Autômato???

Base de indução. r ER com zero operadores

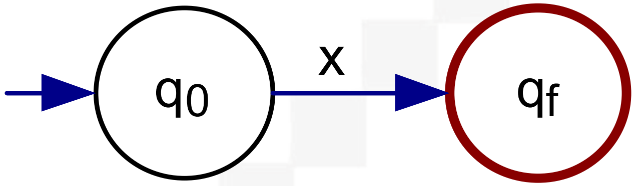
- $r = \varnothing$. Autômato: $M_1 = (\varnothing, \{ q_0 \}, \delta_1, q_0, \varnothing)$



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



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



Hipótese de indução. r ER com até $n > 0$ operadores

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

Passo de indução. r 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

$M_1 = (\Sigma_1, Q_1, \delta_1, q_{0_1}, \{ q_{f_1} \})$ e $M_2 = (\Sigma_2, Q_2, \delta_2, q_{0_2}, \{ q_{f_2} \})$

$\text{ACEITA}(M_1) = \text{GERA}(r_1)$ e $\text{ACEITA}(M_2) = \text{GERA}(r_2)$

$r = r_1 + r_2$

- Autômato???

$r = r_1 r_2$

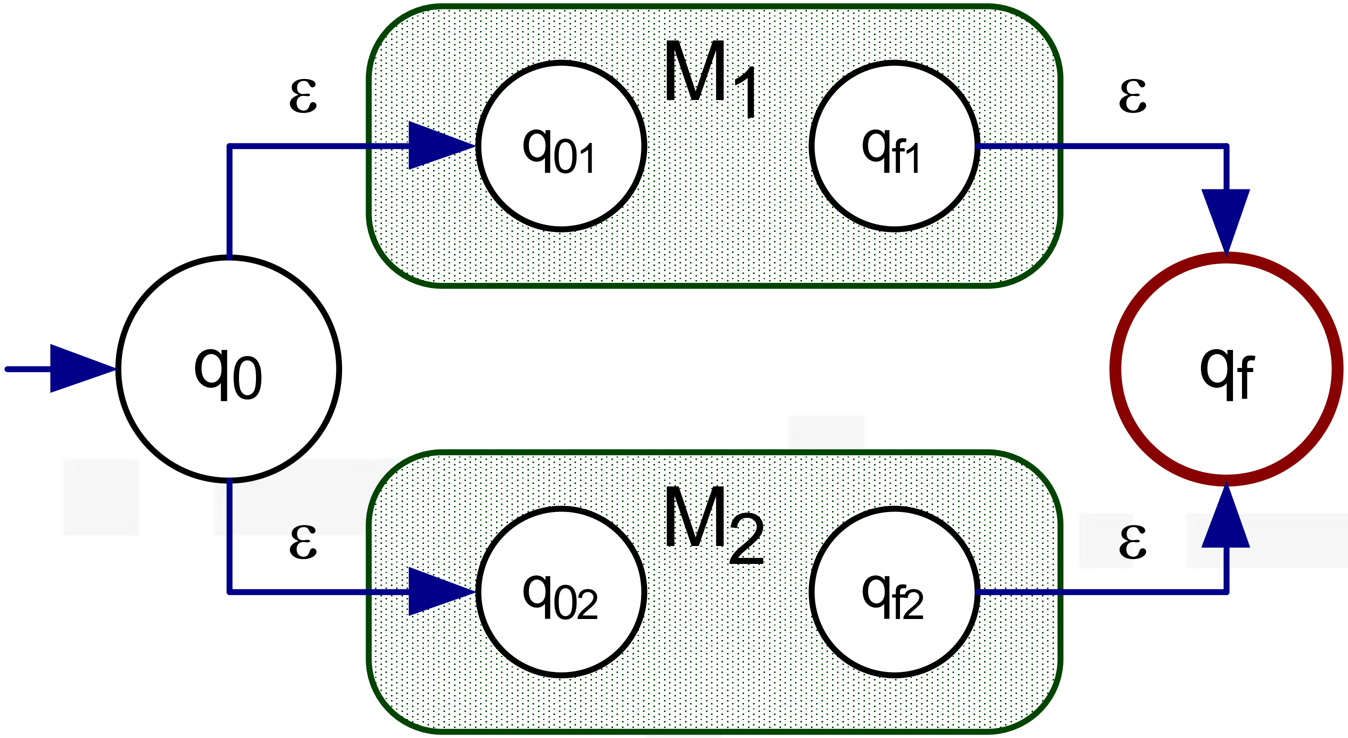
- Autômato???

$r = r_1{}^*$

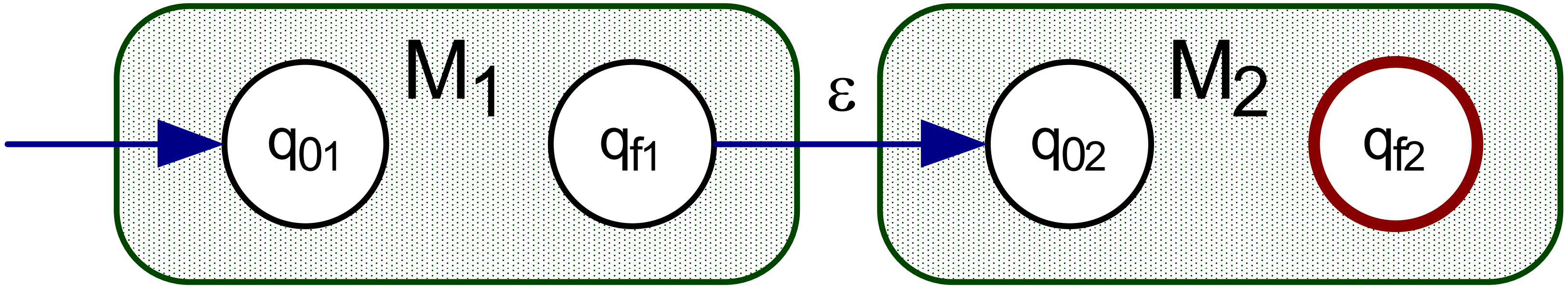
- Autômato???

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

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

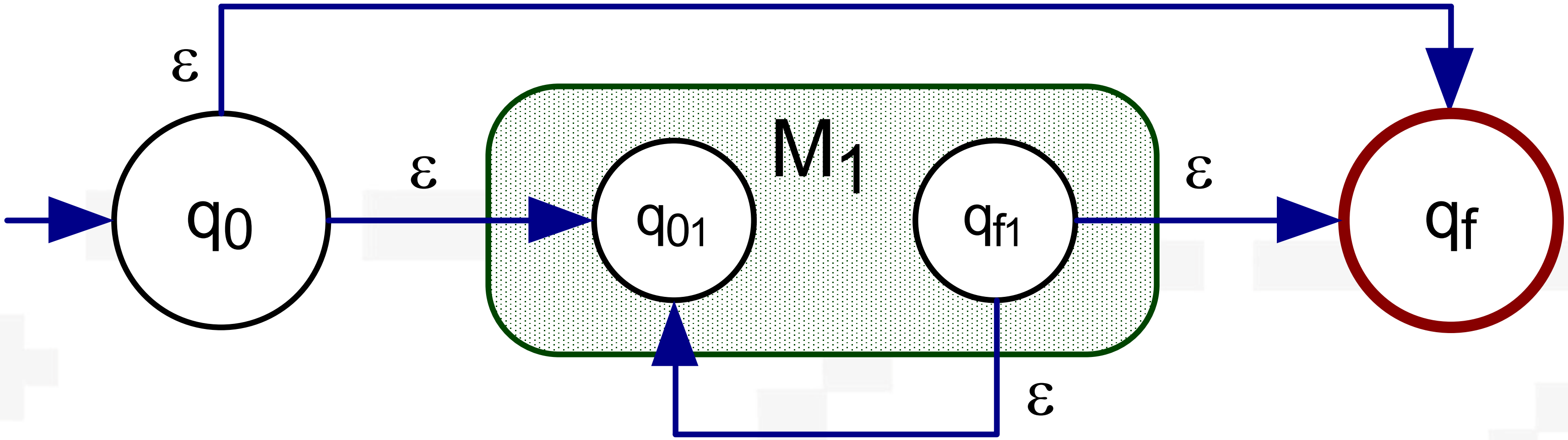


$r = r_1 r_2$. Autômato $M = (\Sigma_1 \cup \Sigma_2, Q_1 \cup Q_2, \delta, q_{0_1}, \{ q_{f_2} \})$



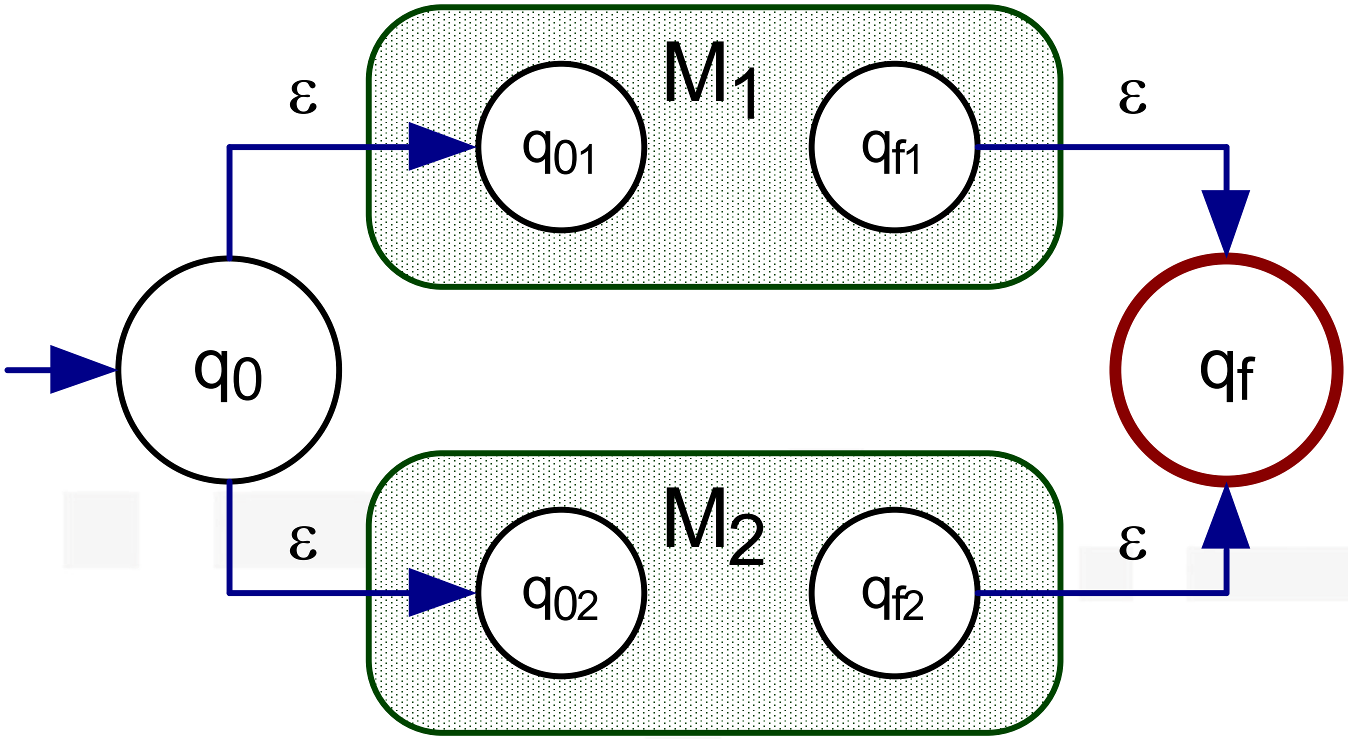
$r = r_1{}^*$. Autômato (suponha $q_0 \notin Q_1$, $q_f \notin Q_1$)

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



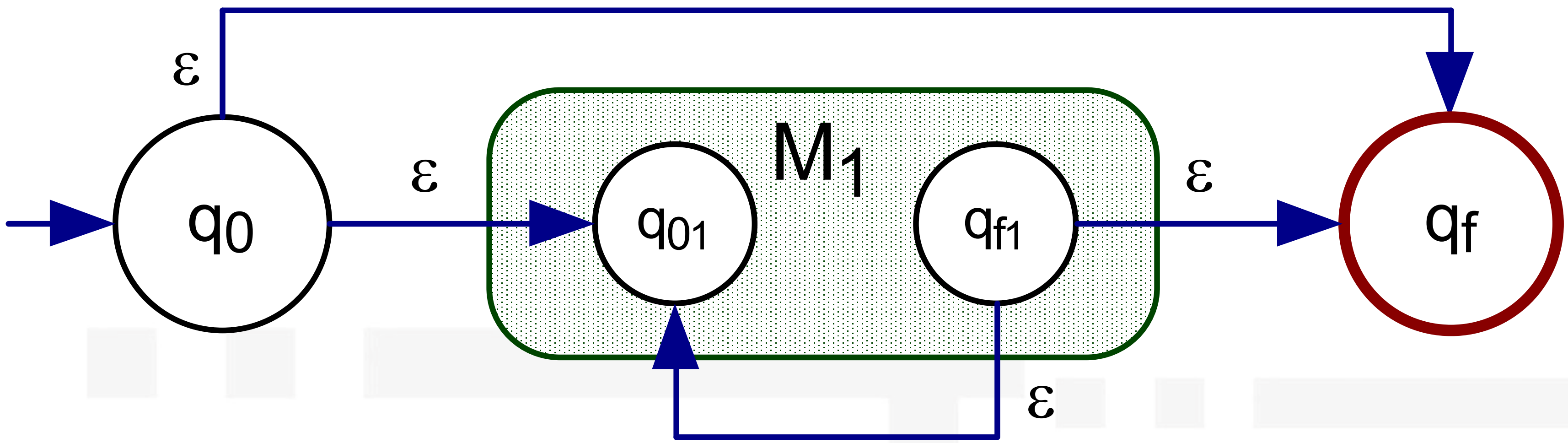
Exercício: no caso $r = r_1 + r_2$

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

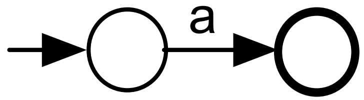
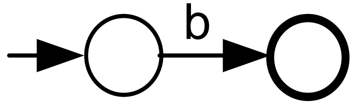
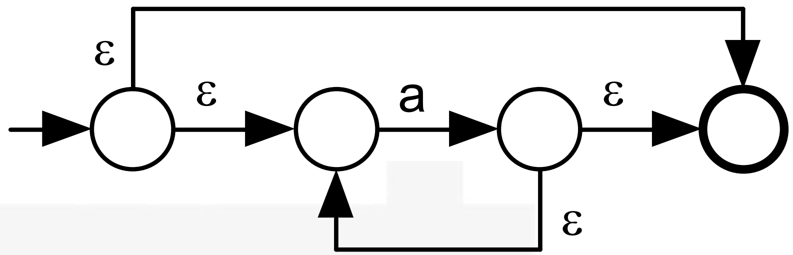
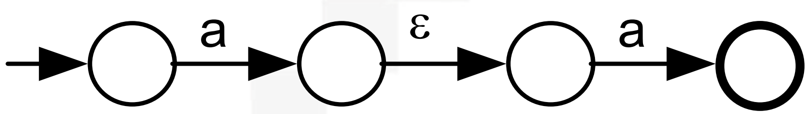


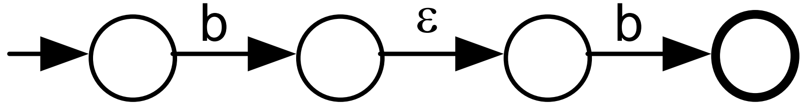
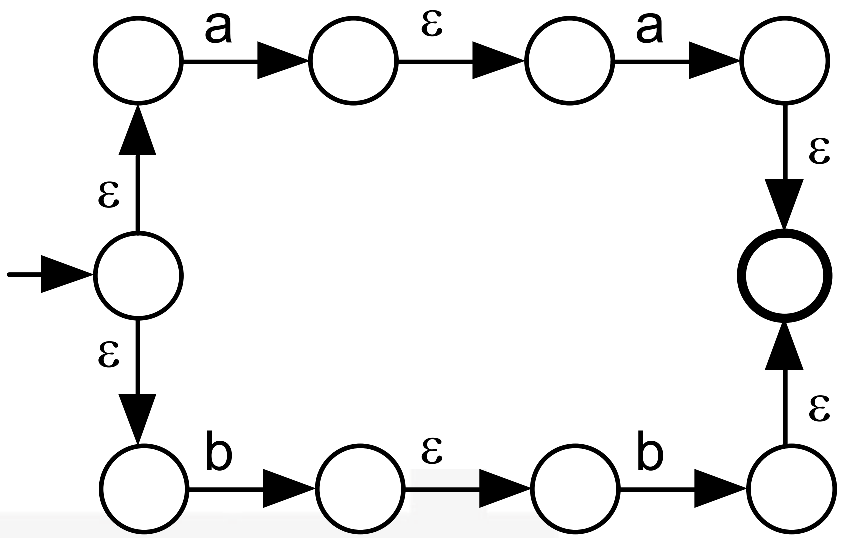
Exercício: no caso $r = r_1{}^*$

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

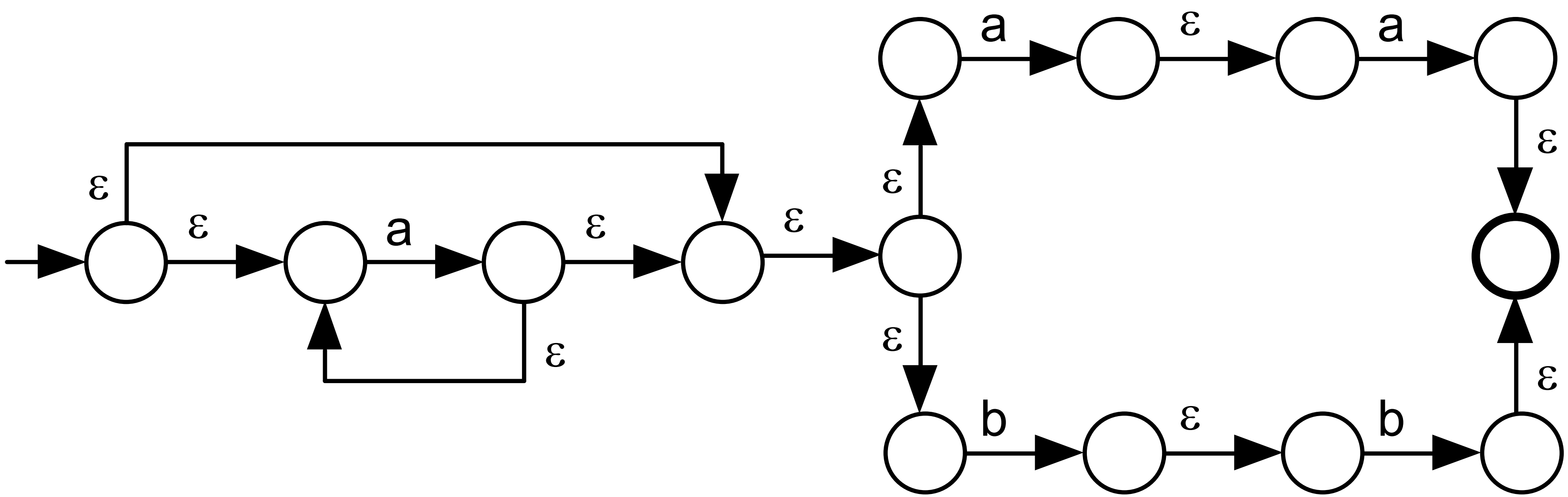


Exp: AFNε a partir de a*(aa + bb)

ER	AFNε
a	
b	
a*	
aa	

<table>
<tr><th>ER</th><th>AFNε</th></tr>
<tr><td>bb</td><td>
</td></tr>
<tr><td>(aa + bb)</td><td>
</td></tr>
</table>

- Autômato resultante: a*(aa + bb)



Teorema: linguagem regular $\rightarrow$ expressão regular

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

$$GERA(r) = L$$

O teorema não será provado

3 – Linguagens Regulares

3.1 Sistema de Estados Finitos
3.2 Composição Sequencial, 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)

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

Gramática linear à esquerda (GLE)

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

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 esquerdo de uma produção

- exatamente uma variável

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

$$L(G) \quad \text{ou} \quad GERA(G)$$

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

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

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

- $S \rightarrow Sba \mid a$

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

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

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

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

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

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

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 à 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 → 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 AFNε 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 AFNε

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

- $\Sigma = T$
- $Q = V \cup \{ q_f \}$ (suponha $q_f \notin V$)
- $F = \{ q_f \}$
- $q_0 = S$

Tipo da produção	Transição gerada
$A \rightarrow \varepsilon$	$\delta(A, \varepsilon) = q_f$
$A \rightarrow a$	$\delta(A, a) = q_f$
$A \rightarrow B$	$\delta(A, \varepsilon) = B$
$A \rightarrow aB$	$\delta(A, a) = B$

M simula as derivações de G

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

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

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

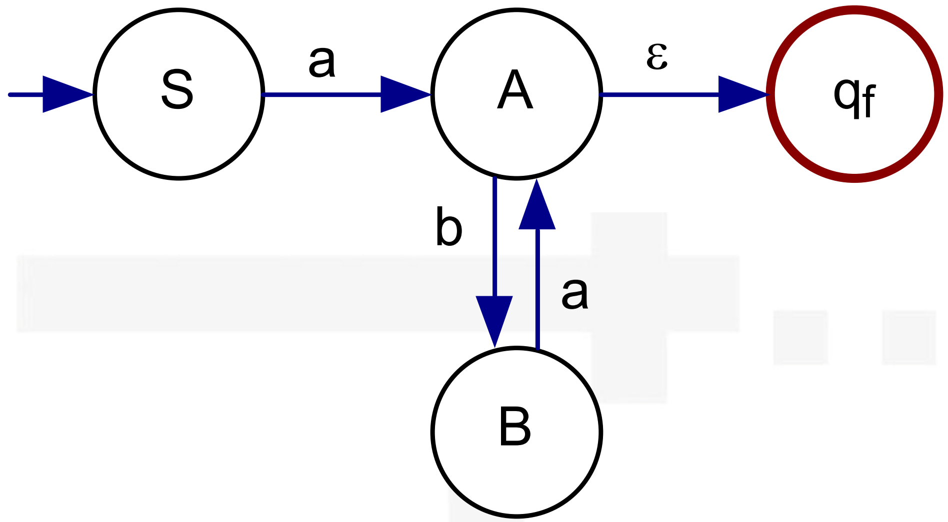
$$\underline{\delta}^*(S, wb) = \delta(\underline{\delta}^*(S, w), b) = \delta(A, b) = B$$

Exp: Construção de um AFNε a partir de uma GR

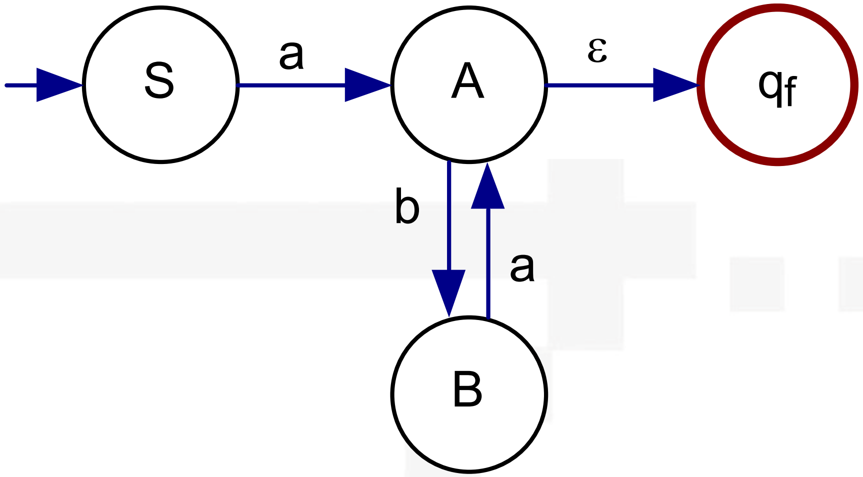
$G = (\{ S, A, B \}, \{ a, b \}, P, S)$

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

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



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 → gramática regular

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

<u>Prova:</u> (*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

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

- 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

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

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

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

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

Base de indução. $|w| = 0$

- por definição, $S \to q_0$ é produção
- se $\varepsilon \in ACEITA(M)$, então
  - q_0 é estado final
  - $q_0 \to \varepsilon$ é produção

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

Hipótese de indução. $|w| = n$ $(n \geq 1)$ e $\underline{\delta}^*(q_0, w) = q$. 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 $\underline{\delta}^*(q_0, wa) = p$. Então

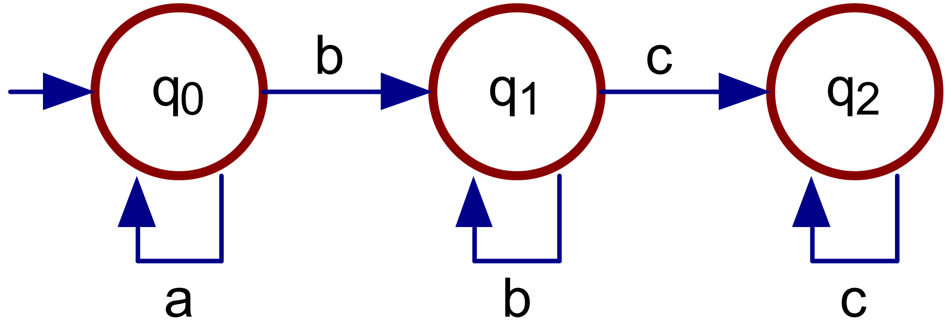
$$\delta(\underline{\delta}^*(q_0, w), a) = \delta(q, a) = p$$

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

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

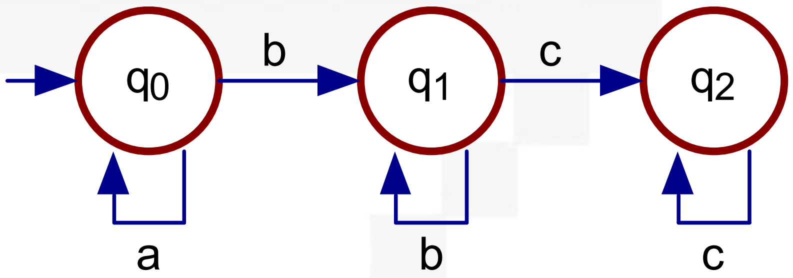
Exp: Construção de uma GR a partir de um AFD

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



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

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



Linguagens Formais e Autômatos

P. Blauth Menezes