2 Linguagens Regulares

2.1 Sistema de Estados Finitos

2.2 Autômato Finito Determinístico

2.3 Autômato Finito Não-Determinístico

2.4 Autômato Finito com Movimentos Vazio

2.5 Expressão Regular

2.6 Gramática Regular

2 Linguagens Regulares

♦ Linguagens Regulares ou Tipo 3

- formalismos operacionais ou reconhecedores
  - *Autômato Finito Determinístico*
  - *Autômato Finito Não-Determinístico*
  - *Autômato Finito com Movimentos Vazio*
- formalismo axiomático ou gerador
  - *Gramática Regular*
- formalismo denotacional
  - *Expressão Regular*
  - também considerado formalismo gerador

♦ Algoritmos Geradores e Reconhecedores

- pouca complexidade
- grande eficiência
- fácil implementação

♦ Hierarquia de Chomsky

- classe de linguagens mais simples
- será estudado adiante

♦ Historicamente

- origem em abordagens específicas
- exemplo
  - *circuitos de chaveamento*

♦ Atualmente

- estudos e aplicações são variados e abrangentes
- exemplos
  - editores de texto
  - processadores de texto em geral
  - pesquisa de dados
  - interface simples homem×máquina
  - protocolos de comunicação
  - lógica
  - ...

2.1 Sistema de Estados Finitos

♦ *Sistema de Estados Finitos*

- modelo matemático de sistema
  - entradas e saídas discretas
- assume um número
  - finito
  - pré-definido
  - de estados
- *estado*
  - resume informações passadas
  - necessárias para determinar as ações para a próxima entrada

♦ Motivacional

- podem ser associados a diversos tipos de sistemas
  - naturais
  - construídos

♦ *Exemplo:* Elevador

- *entrada*
  - requisições pendentes
- *estado*
  - andar corrente
  - direção de movimento
- *não* memoriza as requisições anteriores

♦ *Exemplo:* Analisador Léxico e Processador de Textos

- *entrada*
  - texto
- *estados*
  - informações resumidas do prefixo
- *não* memoriza as unidades (palavras, etc) anteriores

♦ Restrições

- alguns sistemas com estados finitos *não* são adequadamente representados

♦ *Exemplo:* Cérebro Humano

- neurônio
  - em princípio, pode ser representado por um número finito de bits
- composto por cerca de 2^{35} células
- elevado número de estados
  - abordagem pouco eficiente

♦ *Exemplo:* Computador

- processadores e memórias
  - sistema de estados finitos
  - elevado número de estados
- *não* é adequado para o estudo das noções de computabilidade e solucionabilidade
  - desejável memória sem limite predefinido
  - exemplo: *Máquina de Turing*

2.2 Autômato Finito Determinístico

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

♦ *Fita*

- finita (à esquerda e à direita)
- dividida em células
- cada célula
  - armazena um símbolo
- símbolos
  - pertencem a um alfabeto de entrada
- *não* é possível gravar sobre a fita
- palavra de entrada (a ser processada)
  - ocupa toda a fita

♦ *Unidade de Controle*

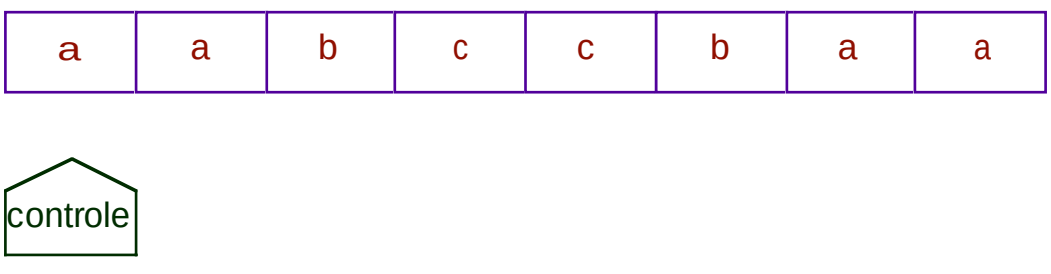
- *estados*
- *unidade de leitura*

♦ *Estados*

- número de estados
  - finito
  - predefinido

♦ *Unidade de Leitura*

- inicialmente
  - cabeça posicionada na célula mais à esquerda da fita
- lê o símbolo de uma célula de cada vez
- após a leitura
  - move a cabeça uma célula para a direita



♦ *Programa*

- função *parcial*
- dependendo
  - do estado corrente
  - e do símbolo lido
  - determina o novo estado
- memorização de informações passadas
  - se necessária
  - usa a estrutura de estados

♦ *Autômato Finito Determinístico*

- ou *AFD* ou simplesmente *Autômato Finito*
- 5-upla

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

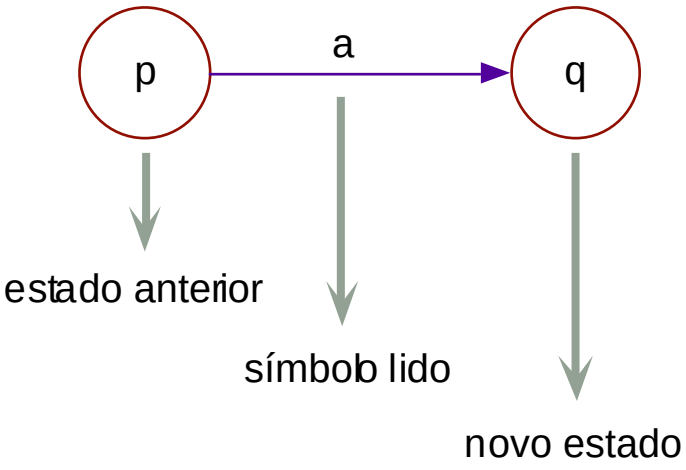
- Σ
  - *alfabeto*
  - símbolos de *entrada*
- Q
  - *conjunto de* estados possíveis de *estados*
  - finito
- δ
  - *função programa* ou de *transição*

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

  - função parcial
- q_0
  - *estado inicial*
  - $q_0 \in Q$
- F
  - *conjunto de estados finais*
  - $F \subseteq Q$

♦ Função Programa

- pode ser interpretada como um grafo finito direto



- representação dos estados
  - inicial
  - finais

♦ Processamento

- sucessiva aplicação da função programa
  - para cada símbolo da entrada
  - da esquerda para a direita
  - até parar
- definição formal do comportamento
  - necessário estender a função programa
- argumento da função programa estendida
  - um estado
  - uma *palavra*

♦ *Função Programa Estendida*

- Seja $M = (\Sigma, Q, \delta, q_0, F)$ um AFD
- $\underline{\delta}: Q \times \Sigma^* \rightarrow Q$ é indutivamente definida
  - $\underline{\delta}(q, \varepsilon) = q$
  - $\underline{\delta}(q, aw) = \underline{\delta}(\delta(q, a), w)$
- Por simplicidade
  - δ e $\underline{\delta}$ são ambas denotadas simplesmente por δ

♦ Um AFD *sempre pára*

- qq palavra de entrada é finita
  - *não* existe a possibilidade de "loop" infinito
  - por quê?
- pára
  - aceitando a entrada
  - rejeitando a entrada

♦ Pára no fim da fita

- após processar o último símbolo da fita
  - *aceita*: atinge um estado final
  - *rejeita*: atinge um estado não-final

♦ Pára por indefinição

- a função programa é indefinida para o argumento (estado corrente e símbolo lido)
  - pára
  - *rejeita*
  - não importa qual o estado corrente

♦ *Linguagem Aceita por um AFD*

- conjunto
  - todas as palavras pertencentes a $\sum^*$
  - aceitas pelo AFD M
- ACEITA(M) ou L(M)

♦ *Linguagem Rejeitada por um AFD*

- conjunto
  - todas as palavras pertencentes a $\sum^*$
  - rejeitadas pelo AFD M
- REJEITA(M)

♦ Note-se que

- ACEITA(M) $\cap$ REJEITA(M) = $\varnothing$
- ACEITA(M) $\cup$ REJEITA(M) = $\sum^*$
- REJEITA(M) = $\sum^*$ - ACEITA(M)
- ACEITA(M) = $\sum^*$ - REJEITA(M)

♦ *Linguagem Regular ou Tipo 3*

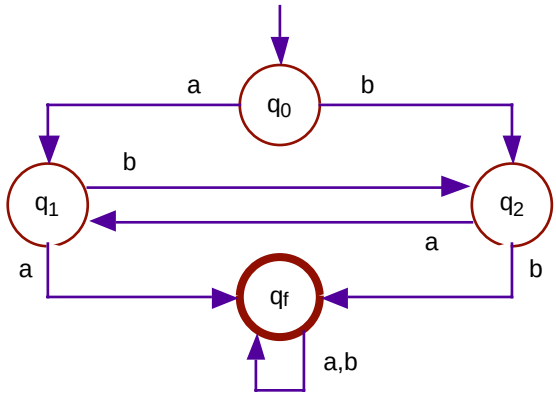
- reconhecida (aceita) por um AFD

♦ *Exemplo*

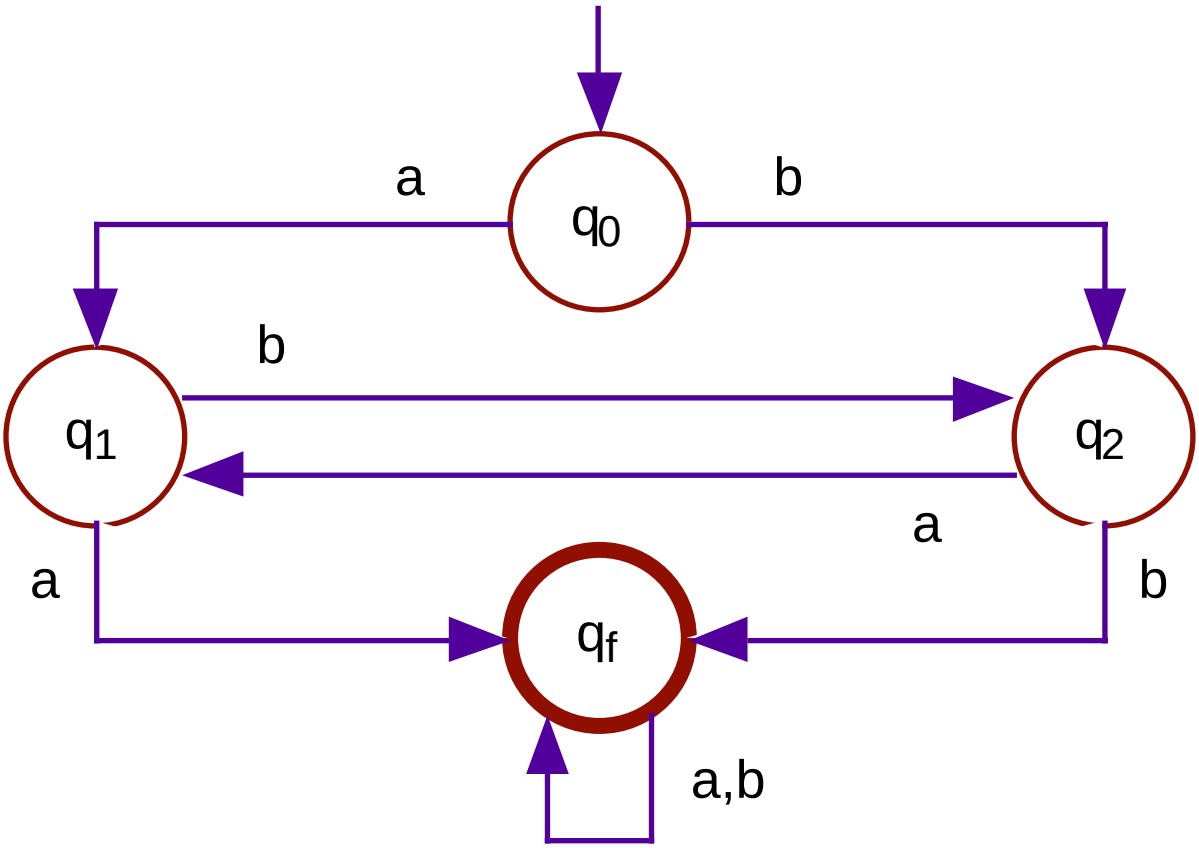
$L_1 = \{w \mid w \text{ possui } aa \text{ ou } bb \text{ como subpalavra}\}$

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

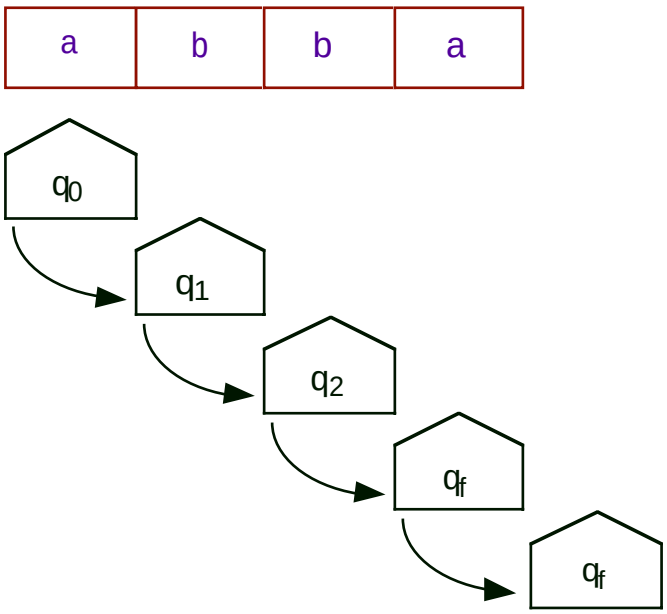
δ_1	a	b
q_0	q_1	q_2
q_1	q_f	q_2
q_2	q_1	q_f
q_f	q_f	q_f



- algoritmo
  - q_1: o símbolo anterior é a
  - q_2: o símbolo anterior é b
  - após identificar aa ou bb: estado final q_f e varre o sufixo somente para terminar o processamento



- processamento de w = abba (aceita)



♦ *Exemplo*

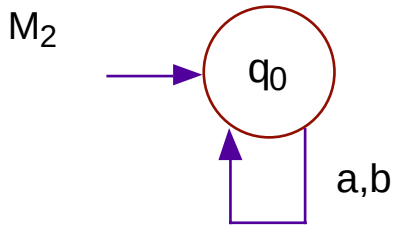
$L_2 = \{\}$ $\quad\quad$ $L_3 = \Sigma^*$

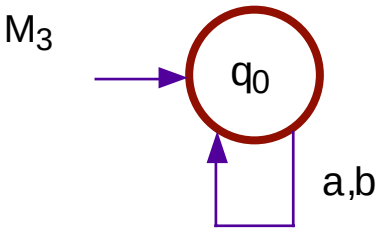
$M_2 = (\{a, b\}, \{q_0\}, \delta_2, q_0, \{\})$

$M_3 = (\{a, b\}, \{q_0\}, \delta_3, q_0, \{q_0\})$

δ_2	a	b
q_0	q_0	q_0

δ_3	a	b
q_0	q_0	q_0

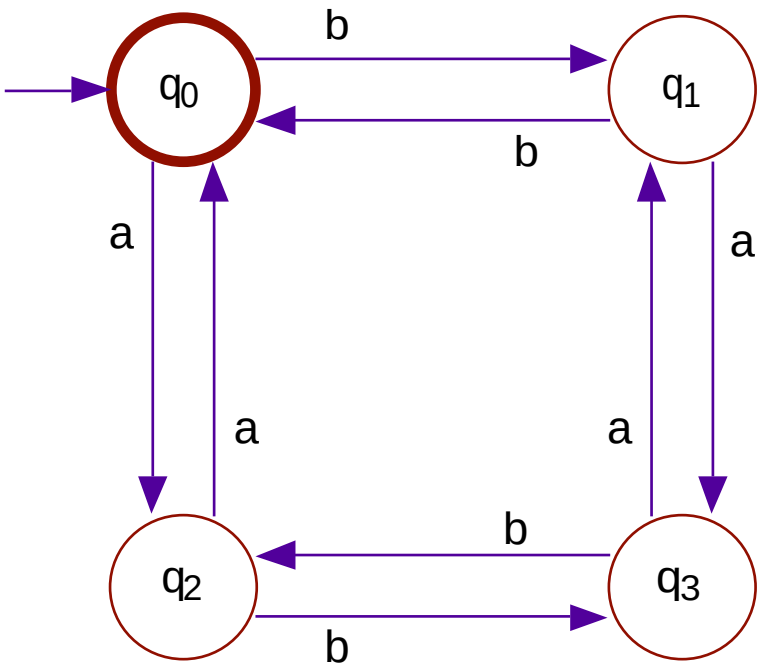


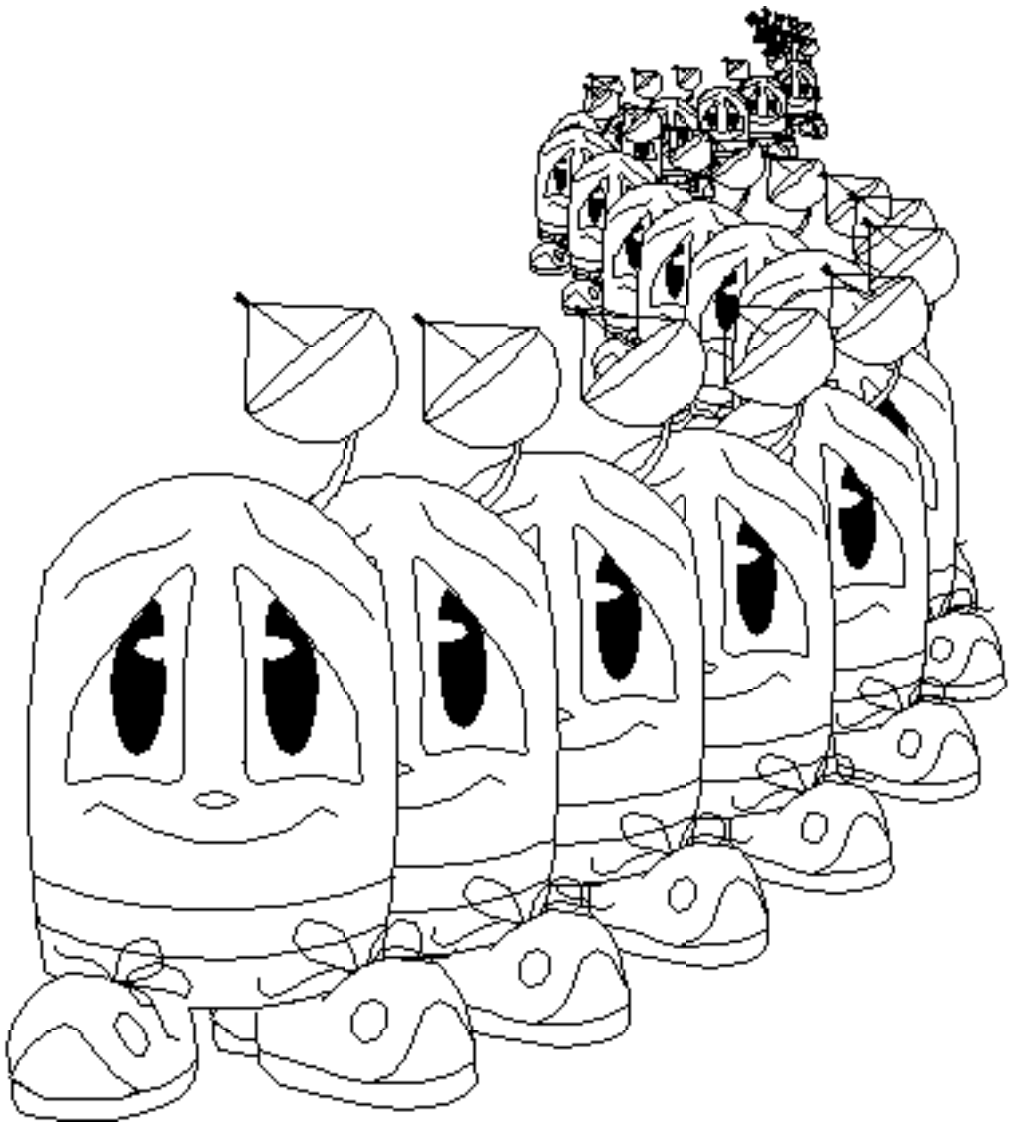


♦ *Exemplo*

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

$M_4 = (\{a, b\}, \{q_0, q_1, q_2, q_3\}, \delta_4, q_0, \{q_0\})$





2.3 Autômato Finito Não-Determinístico

♦ Não-Determinismo

- generalização dos modelos de máquinas
- de fundamental importância
  - teoria da computação
  - linguagens formais
- nem sempre aumenta o poder computacional
  - de uma classe de autômatos
- em particular
  - qq AF não-determinístico
  - pode ser simulado por um AFD

♦ Idéia básica

- o processamento de uma entrada
  - resulta em um *conjunto* de novos estados

♦ Visto como uma máquina

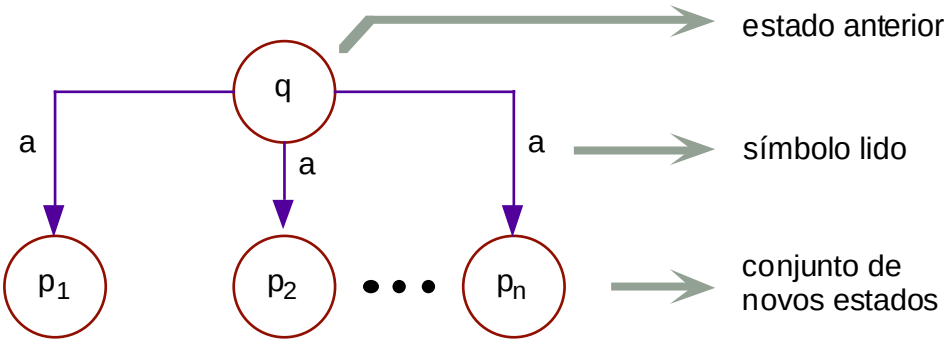
- fita + unidade de controle + programa
- assume um conjunto de estados alternativos
  - "multiplicação" da unidade de controle
  - uma para cada alternativa
- processamento de um caminho
  - não influi no nos demais caminhos alternativos
  - estado, símbolo lido e posição da cabeça dos "independentes"

♦ *Definição. AF Não-Determinístico*

- ou simplesmente AFN
- 5-upla $M = (\Sigma, Q, \delta, q_0, F)$
- Σ alfabeto de símbolos de entrada
- Q conjunto finito de estados
- δ função programa ou função de transição
  - $\delta: Q \times \Sigma \rightarrow 2^Q$
  - função parcial
- q_0 estado inicial do autômato tq $q_0 \in Q$
- F conjunto de estados finais tq $F \subseteq Q$

♦ Função Programa

- pode ser interpretada como um grafo finito direto, de duas formas



♦ Processamento

- união dos resultados da função programa aplicada a cada estado alternativo
- definição formal do comportamento
  - ∗ necessário estender a função programa
  - ∗ argumento: um conjunto finito de estados e uma palavra

♦ *Função Programa Estendida*

- Seja $M = (\Sigma, Q, \delta, q_0, F)$ um AFN
- $\underline{\delta}: 2^Q \times \Re^* \rightarrow 2^Q$ é indutivamente definida
  - ∗ $\underline{\delta}(P, \varepsilon) = P$
  - ∗ $\underline{\delta}(P, aw) = \underline{\delta}(\cup_{q \in P} \delta(q, a), w)$
- portanto

$$\underline{\delta}(\{q_1, q_2, ..., q_n\}, a) =$$

$$\delta(q_1, a) \cup \delta(q_2, a) \cup ... \cup \delta(q_n, a)$$

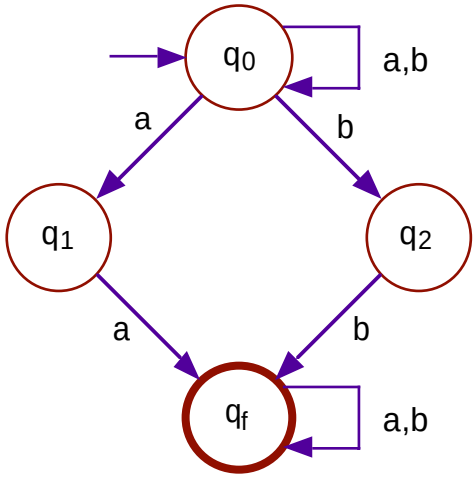
♦ Linguagem Aceita / Rejeitada

- $w \in$ ACEITA(M)
  - ∗ pelo menos um caminho alternativo aceita w
- $w \in$ REJEITA(M)
  - ∗ todas as alternativas rejeitam w

♦ *Exemplo:* $L_5 = \{w \mid w$ possui aa ou bb como subpalavra$\}$

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

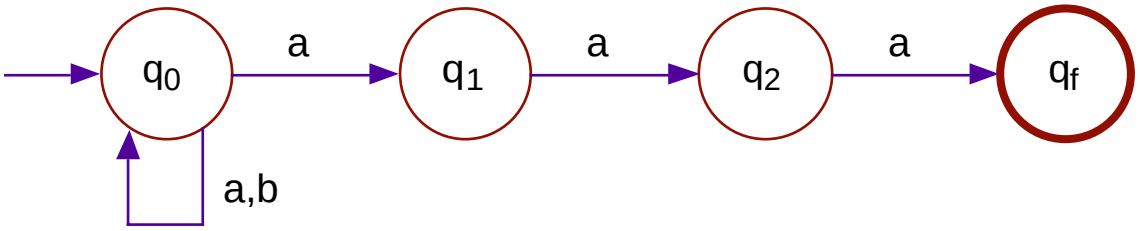
δ_5	a	b
q_0	$\{q_0,q_1\}$	$\{q_0,q_2\}$
q_1	$\{q_f\}$	-
q_2	-	$\{q_f\}$
q_f	$\{q_f\}$	$\{q_f\}$



- a cada ocorrência de a (ou b)
  - uma alternativa é iniciada
  - verifica se o próximo símbolo é a (ou b)
- construções
  - ciclo em q_0: varre toda a entrada
  - caminho $q_0/q_1/q_f$: ocorrência de aa
  - caminho $q_0/q_2/q_f$: ocorrência de bb

♦ *Exemplo:* $L_6 = \{w \mid w$ possui aaa como sufixo$\}$

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



♦ Determinismo × Não-Determinismo

- não-determinismo
  - aparentemente, um significativo acréscimo
  - ao poder computacional de um AF
- na realidade
  - não aumenta o poder computacional
- para cada AFN
  - é possível construir um AFD equivalente
  - (que realiza o mesmo processamento)
  - o contrário também é verdadeiro

♦ *Teorema:* A classe dos AFD é equivalente à classe dos AFN

- uma linguagem é regular sse é aceita por um AFN
- a capacidade de reconhecimento dos AFN é a mesma dos AFD

♦ Prova

- mostrar que
  - a partir de um AFN M qq
  - é possível construir um AFD M'
  - que realiza o mesmo processamento
  - (ou seja, M' simula M)
- AFN →AFD
  - estados de M' que simulam as diversas combinações de estados alternativos de M
  - demonstração de que o M' simula M: indução no tamanho da palavra
- AFD → AFN
  - decorre trivialmente das definições (por que?)

♦ Prova AFN $\rightarrow$AFD

- seja $M = (\Sigma, Q, \delta, q_0, F)$ um AFN qq
- AFD $M' = (\Sigma, Q', \delta', \langle q_0 \rangle, F')$
- Q'
  - todas as combinações de estados de Q
  - denotadas por $\langle q_1 q_2 ... q_n \rangle$
  - a ordem dos elementos não identifica mais combinações: $\langle q_u q_v \rangle = \langle q_v q_u \rangle$
- δ'
  - $\delta'(\langle q_1 ... q_n \rangle, a) = \langle p_1 ... p_m \rangle$ sse $\delta(\{q_1, ..., q_n\}, a) = \{p_1, ..., p_m\}$
- $\langle q_0 \rangle$
  - estado inicial
- F'
  - conjunto de $\langle q_1 q_2 ... q_n \rangle \in Q'$
  - tq alguma componente $q_i \in F$

♦ ¿ AFD M' simula AFN M ?

- indução no tamanho da palavra
- deve-se mostrar que

$$\delta'(\langle q_0 \rangle, w) = \langle q_1 ... q_u \rangle \text{ sse } \delta(\{q_0\}, w) = \{q_1, ..., q_u\}.$$

- *Base da indução*. $|w| = 0$
  - $\delta'(\langle q_0 \rangle, \varepsilon) = \langle q_0 \rangle$ sse $\delta(\{q_0\}, \varepsilon) = \{q_0\}$
  - V por definição de função programa estendida
- *Hipótese de indução*. $|w| = n, n \geq 1$
  - suponha que
  - $\delta'(\langle q_0 \rangle, w) = \langle q_1 ... q_u \rangle$ sse $\delta(\{q_0\}, w) = \{q_1,...,q_u\}$
- *Passo de Indução*. $|wa| = n + 1$ e $n \geq 1$
  - $\delta'(\langle q_0 \rangle, wa) = \langle p_1 ... p_v \rangle$ sse $\delta(\{q_0\}, wa) = \{p_1,...,p_v\}$

o que equivale, por hipótese de indução

$\delta'(\langle q_1 ... q_u \rangle, a) = \langle p_1 ... p_v \rangle$ sse $\delta(\{q_1, ..., q_u\}, a) = \{p_1, ..., p_v\}$

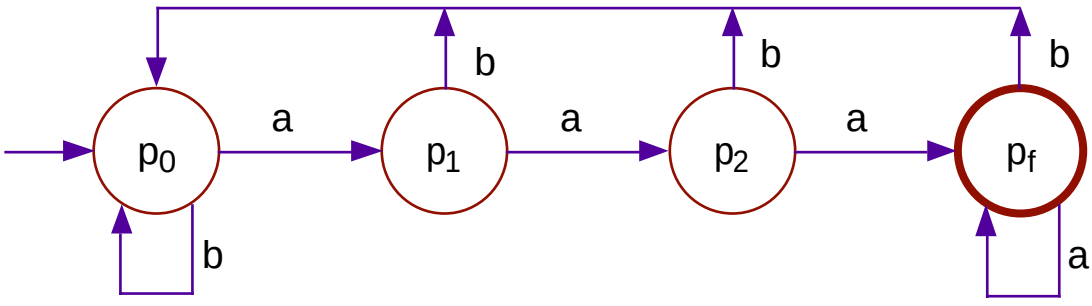
o que é verdadeiro, por definição de δ'

♦ Logo, M' simula M para qq w

♦ *Exemplo:*

- AFN $M_6 = (\{a, b\}, \{q_0, q_1, q_2, q_f\}, \delta_6, q_0, \{q_f\})$
  - exemplo anterior
- AFD $M_6' = (\{a, b\}, Q', \delta_6', \langle q_0 \rangle, F')$
  - $Q' = \{\langle q_0 \rangle, \langle q_1 \rangle, \langle q_2 \rangle, \langle q_f \rangle, \langle q_0 q_1 \rangle, \langle q_0 q_2 \rangle, ..., \langle q_0 q_1 q_2 q_f \rangle\}$
  - $F' = \{\langle q_f \rangle, \langle q_0 q_f \rangle, \langle q_1 q_f \rangle, ..., \langle q_0 q_1 q_2 q_f \rangle\}$

δ_6'	a	b
$p_0 = \langle q_0 \rangle$	$\langle q_0 q_1 \rangle$	$\langle q_0 \rangle$
$p_1 = \langle q_0 q_1 \rangle$	$\langle q_0 q_1 q_2 \rangle$	$\langle q_0 \rangle$
$p_2 = \langle q_0 q_1 q_2 \rangle$	$\langle q_0 q_1 q_2 q_f \rangle$	$\langle q_0 \rangle$
$p_f = \langle q_0 q_1 q_2 q_f \rangle$	$\langle q_0 q_1 q_2 q_f \rangle$	$\langle q_0 \rangle$



-

2.4 Autômato Finito com Movimentos Vazio

♦ Movimento Vazio

- generalização do não-determinismo
- importante no estudo
  - da computação
  - das linguagens (sintaxe e semântica)
- nem sempre aumenta o poder computacional
  - de uma classe de autômatos
- em particular
  - qq AFN com movimentos vazio
  - pode ser simulado por um AFN
- relativamente aos AF, facilita
  - construções
  - demonstrações

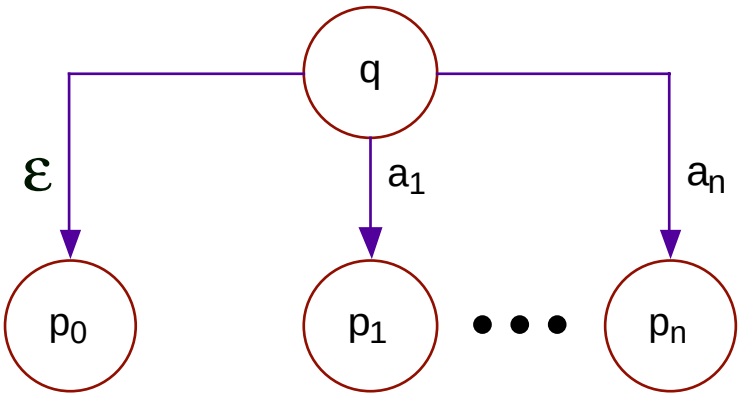
♦ Idéia básica

- função programa pode incluir
  - transições sem leitura de símbolo da fita
- em semântica formal
  - transições encapsuladas

♦ *AF com Movimentos Vazio*

- AFNε ou simplesmente AFε
- $M = (\Sigma, Q, \delta, q_0, F)$
- Σ, Q, F, q_0 como em um AFN
  - Σ alfabeto de símbolos de entrada
  - Q conjunto finito de estados
  - q_0 estado inicial do autômato tq $q_0 \in Q$
  - F conjunto de estados finais tq $F \subseteq Q$
- δ função programa ou função de transição
  - $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$
  - função parcial

♦ Função Programa



♦ Processamento (semântica)

- análogo ao de um AFN
- processamento de uma transição vazia
  - também é não-determinista
  - assume simultaneamente os estados destino e origem
  - origem de uma transição vazia sempre é um caminho alternativo

♦ Processamento (formal)

- função programa estendida
  - conjunto de estados
  - palavra
  - baseado na noção de fecho vazio

♦ Fecho Vazio

- de um estado (ou conjunto de estados)
- resulta em um conjunto de estados
- atingíveis exclusivamente por zero ou mais movimentos vazio

♦ *Definição. Fecho Vazio*

- FECHO-ε ou Fε
- seja $M = (\Sigma, Q, \delta, q_0, F)$ um AFε
- $F\varepsilon: Q \rightarrow 2^Q$ é indutivamente definida
- $\delta(q, \varepsilon)$ *não* é definido
  - $F\varepsilon(q) = \{q\}$
- $\delta(q, \varepsilon)$ *é* definido
  - $F\varepsilon(q) = \{q\} \cup \delta(q, \varepsilon) \cup \cup_{p \in \delta(q,\varepsilon)} F\varepsilon(p)$

♦ *Definição. Fecho Vazio Estendida*

- para conjunto de estados
- $\underline{F\varepsilon}: 2^Q \rightarrow 2^Q$ tq
  - $\underline{F\varepsilon}(P) = \cup_{q \in P} F\varepsilon(q)$

♦ *Função Programa Estendida*

- Seja $M = (\Sigma, Q, \delta, q_0, F)$ um AFε
- $\underline{\delta}: 2^Q \times \Re^* \rightarrow 2^Q$, é indutivamente definida
  - $\underline{\delta}(P, \varepsilon) = F\varepsilon(P)$
  - $\underline{\delta}(P, wa) = F\varepsilon(R)$
  - $R = \{r \mid r \in \delta(s, a) \text{ e } s \in \underline{\delta}(P, w)\}$

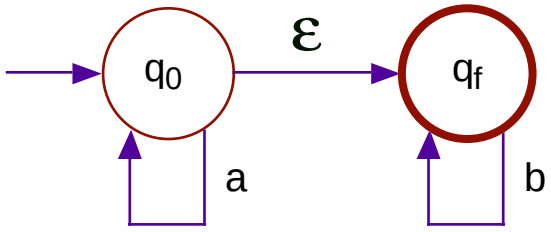
♦ Linguagem Aceita / Rejeitada

- análogo ao AFN
- $w \in$ ACEITA(M)
  - pelo menos um caminho alternativo aceita w
- $w \in$ REJEITA(M)
  - todas as alternativas rejeitam w

♦ *Exemplo:* $L_7 = \{w \mid$ qualquer símbolo a antecede qualquer símbolo b$\}$

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

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

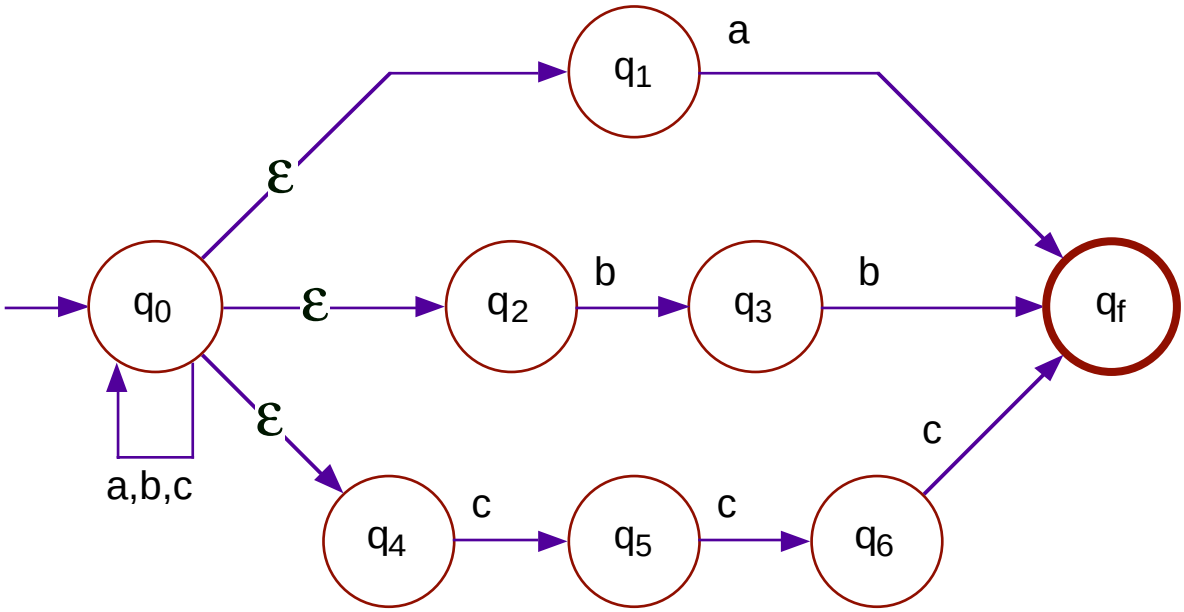


♦ *Exemplo:* AFε M_7

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

♦ *Exemplo:* $L_8 = \{w \mid w$ possui como sufixo a ou bb ou ccc$\}$

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



$F\varepsilon(\{q_0\}) = \{q_0, q_1, q_2, q_4\}$

$\underline{\delta}(\{q_0\}, abb) = F\varepsilon(\{r \mid r \in \delta(s, b)$ e $s \in \underline{\delta}(\{q_0\}, ab)\})$

$\underline{\delta}(\{q_0\}, ab) = F\varepsilon(\{r \mid r \in \delta(s, b)$ e $s \in \underline{\delta}(\{q_0\}, a)\})$

$\underline{\delta}(\{q_0\}, a) = F\varepsilon(\{r \mid r \in \delta(s, a)$ e $s \in \underline{\delta}(\{q_0\}, \varepsilon)\})$

- $\underline{\delta}(\{q_0\}, \varepsilon) = F\varepsilon(\{q_0\}) = \{q_0, q_1, q_2, q_4\}$
- $\underline{\delta}(\{q_0\}, a) = \{q_0, q_1, q_2, q_4, q_f\}$
- $\underline{\delta}(\{q_0\}, ab) = \{q_0, q_1, q_2, q_3, q_4\}$
- $\underline{\delta}(\{q_0\}, abb) = \{q_0, q_1, q_2, q_4, q_f\}$

♦ Não-Determinismo × Movimentos Vazio

- movimentos vazio
  - aparentemente, um significativo acréscimo
  - ao poder computacional de um AFN
- na realidade
  - não aumenta o poder computacional
- para cada AFε
  - é possível construir um AFN equivalente
  - (que realiza o mesmo processamento)
  - o contrário também é verdadeiro

♦ *Teorema:* A classe dos AFε é equivalente à classe dos AFN

- uma linguagem é regular sse é aceita por um AFε
- a capacidade de reconhecimento dos AFε é a mesma dos AFD e dos AFN

♦ Prova

- mostrar que
  - ∗ a partir de um AFε qq
  - ∗ é possível construir um AFN
  - ∗ que realiza o mesmo processamento
- AFε →AFN
  - ∗ cada transição (não-vazia)
  - ∗ estendida com todos os estados possíveis de serem atingidos por transições vazias
- AFN →AFε
  - ∗ decorre trivialmente das definições (por que?)

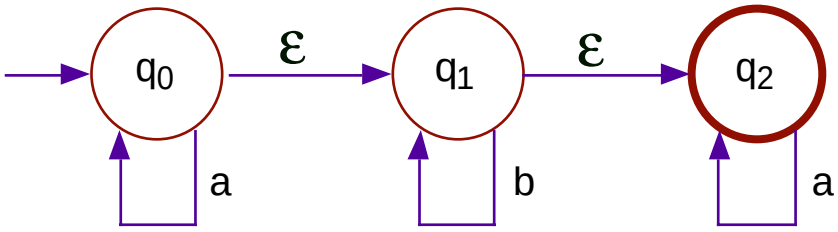
♦ Prova

- seja $M = (\Sigma, Q, \delta, q_0, F)$ um AFε qualquer
- seja $M' = (\Sigma, Q, \delta', q_0, F')$ um AFN
- δ'
  - $\delta' : Q \times \Sigma \rightarrow 2^Q$
  - $\delta'(q, a) = \underline{\delta}(\{q\}, a)$
- F'
  - conjunto de todos $q \in Q$ tq
  - algum elemento do $F\varepsilon(q)$ pertence a F
- ¿ AFN simula o AFε ?
  - indução no tamanho da palavra
  - exercício

♦ *Exemplo*

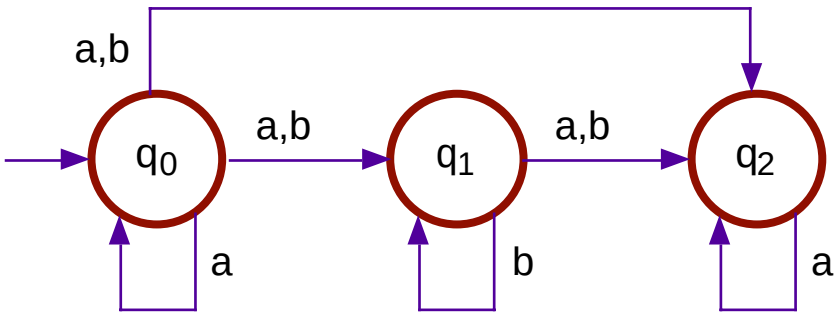
- $M_9 = (\{a, b\}, \{q_0, q_1, q_2\}, \delta_9, q_0, \{q_2\})$

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



- $M_9' = (\{a, b\}, \{q_0, q_1, q_2\}, \delta_9', q_0, F')$
- $F' = \{q_0, q_1, q_2\}$, pois
  - $F\varepsilon(q_0) = \{q_0, q_1, q_2\}$
  - $F\varepsilon(q_1) = \{q_1, q_2\}$
  - $F\varepsilon(q_2) = \{q_2\}$

- $\delta_9'(q_0, a) = \underline{\delta}_9(\{q_0\}, a) =$
  $F\varepsilon(\{r \mid r \in \delta(s, a) \text{ e } s \in \underline{\delta}(\{q_0\}, \varepsilon)\}) = \{q_0, q_1, q_2\}$
- $\delta_9'(q_0, b) = \underline{\delta}_9(\{q_0\}, b) =$
  $F\varepsilon(\{r \mid r \in \delta(s, b) \text{ e } s \in \underline{\delta}(\{q_0\}, \varepsilon)\}) = \{q_1, q_2\}$
- $\delta_9'(q_1, a) = \underline{\delta}_9(\{q_1\}, a) =$
  $F\varepsilon(\{r \mid r \in \delta(s, a) \text{ e } s \in \underline{\delta}(\{q_1\}, \varepsilon)\}) = \{q_2\}$
- $\delta_9'(q_1, b) = \underline{\delta}_9(\{q_1\}, b) =$
  $F\varepsilon(\{r \mid r \in \delta(s, b) \text{ e } s \in \underline{\delta}(\{q_1\}, \varepsilon)\}) = \{q_1, q_2\}$
- $\delta_9'(q_2, a) = \underline{\delta}_9(\{q_2\}, a) =$
  $F\varepsilon(\{r \mid r \in \delta(s, a) \text{ e } s \in \underline{\delta}(\{q_2\}, \varepsilon)\}) = \{q_2\}$
- $\delta_9'(q_2, b) = \underline{\delta}_9(\{q_2\}, b) =$
  $F\varepsilon(\{r \mid r \in \delta(s, b) \text{ e } s \in \underline{\delta}(\{q_2\}, \varepsilon)\})$ é indefinida



2.5 Expressão Regular

♦ Expressão Regular

- toda LR pode ser descrita por uma
  - *Expressão Regular*
- formalismo
  - simples
  - denotacional (gerador)
- definida a partir de
  - conjuntos (linguagens) básicos
  - operação de concatenação
  - operação de união
- são adequadas para a comunicação
  - homem × homem
  - homem × máquina

♦ *Definição. Expressão Regular*

- ou simplesmente ER
- sobre um alfabeto $\sum$
- indutivamente definida
  - ∗ $\varnothing$ é ER e denota a *linguagem vazia*
  - ∗ ε é ER e denota a linguagem $\{\varepsilon\}$
  - ∗ **x** é ER onde $x \in \sum$ e denota a linguagem $\{x\}$
  - ∗ se **r** e **s** são ER e denotam as linguagens R e S, então

(r+s)	é ER e denota $R \cup S$
(rs)	é ER e denota RS
(r*)	é ER e denota R*

♦ Omissão de parênteses em ER

- é usual
- concatenação sucessiva tem precedência sobre
  - ∗ concatenação
  - ∗ união
- concatenação tem precedência sobre
  - ∗ união

♦ Linguagem Gerada

- por uma ER r
- é representada por L(r) ou GERA(r)

♦ Exemplo

ER	Linguagem Representada
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 sub-palavra
a*ba*ba*	todas as palavras contendo exatamente dois b
(a + b)*(aa + bb)	todas as palavras que terminam com aa ou bb
(a + ε)(b + ba)*	todas as palavras que não possuem dois a consecutivos

♦ As ER denotam exatamente as LR

♦ *Teorema*

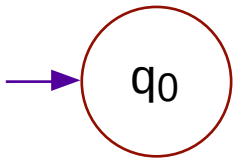
- Se r é uma ER,
- então GERA(r) é uma LR

♦ Prova

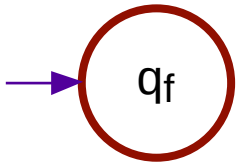
- L é LR sse é possível construir um
  - AF (AFD, AFN ou AFε), que reconheça L
- portanto, é necessário mostrar que,
  - dado uma ER r qq,
  - é possível construir um AF M tq
  - ACEITA(M) = GERA(r)
- demonstração de que ACEITA(M) = GERA(r)
  - indução no número de operadores
- é assumido que qq AF
  - pode ser simulado por um AF com exatamente um estado final (por que?)

♦ Base (ER com zero operadores)

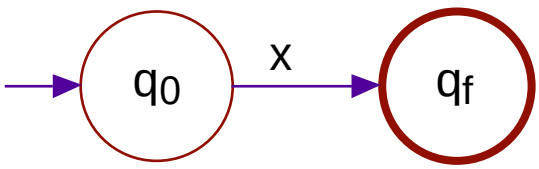
- se r tem zero operadores, então é da forma:
  - r = ∅
  - r = ε
  - r = x (x ∈ Σ)
- **r** = ∅. $M_1 = (\emptyset, \{q_0\}, \delta_1, q_0, \emptyset)$



- **r** = ε. $M_2 = (\emptyset, \{q_f\}, \delta_2, q_f, \{q_f\})$



- **r** = **x**. $M_3 = (\{x\}, \{q_0, q_f\}, \delta_3, q_0, \{q_f\})$



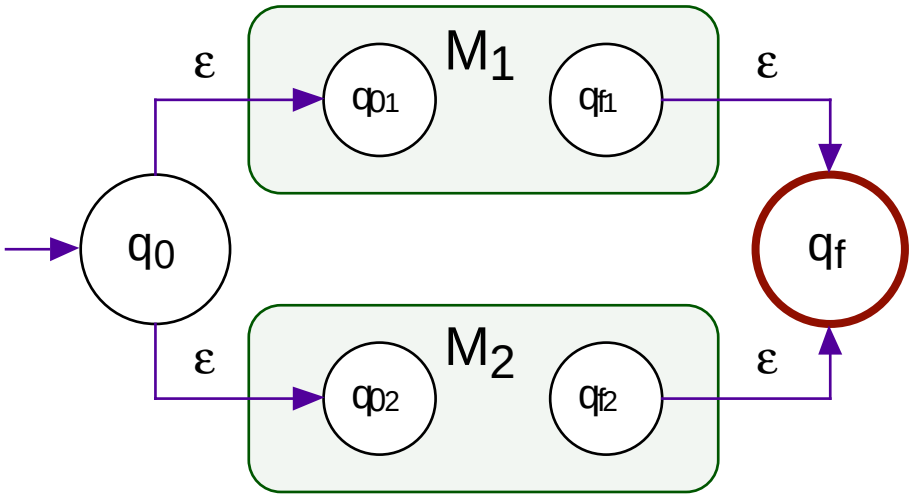
♦ Hipótese (ER com até n > 0 operadores)

- suponha que é possível construir um autômato finito que aceita a linguagem Gera(r)

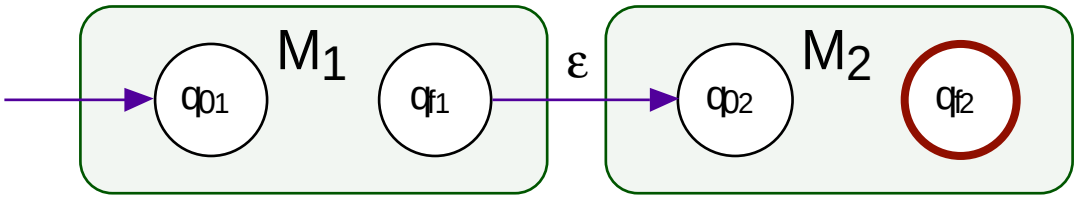
♦ Indução (ER com n + 1 operadores)

- se r possui n + 1 operadores, então a ER 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
  - $M_1 = (\Sigma_1, Q_1, \delta_1, q0_1, \{qf_1\})$ tq $L(M_1) = GERA(r_1)$
  - $M_2 = (\Sigma_2, Q_2, \delta_2, q0_2, \{qf_2\})$ tq $L(M_2) = GERA(r_2)$

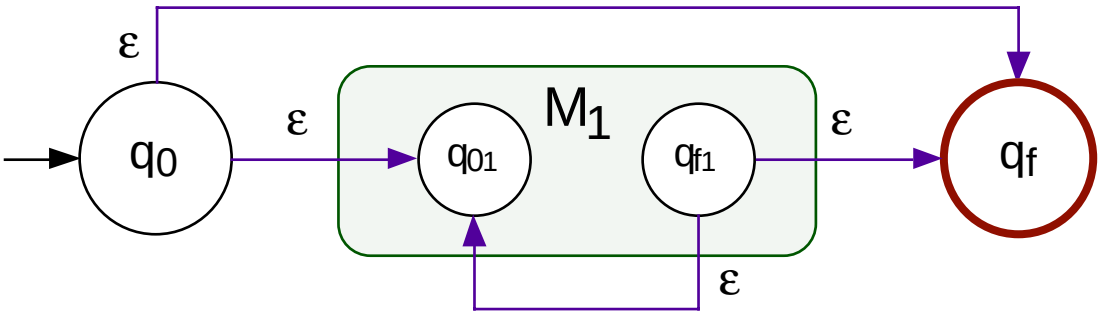
- **$r = r_1 + r_2$**
  - $M = (\Sigma_1 \cup \Sigma_2, Q_1 \cup Q_2 \cup \{q_0, q_f\}, \delta, q_0, \{q_f\})$



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



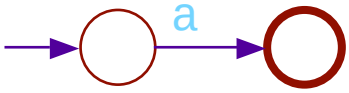
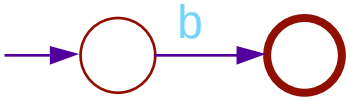
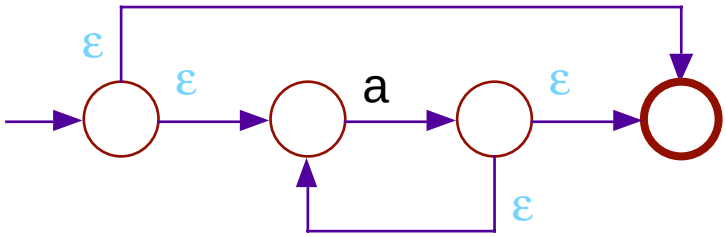
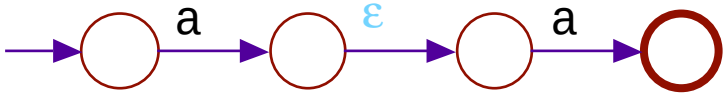
- **$r = r_1^*$**
  - $M = (\Sigma_1, Q_1 \cup \{q_0, q_f\}, \delta, q_0, \{q_f\})$



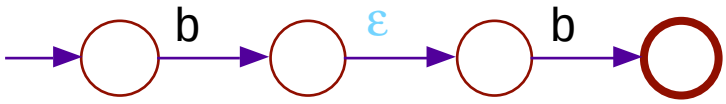
♦ Observação & Exercício

- algumas alterações na construção do autômato resultante podem gerar resultados indesejados
- $r = r_1 + r_2$. Não introduzir q_0 e q_f. Identificar estados iniciais/finais de M_1/M_2 (por que?)
- $r = r_1^*$. Não introduzir q_f. Manter q_{f_1} como estado final. Transição vazia de q_0 para q_{f_1} (por que?)

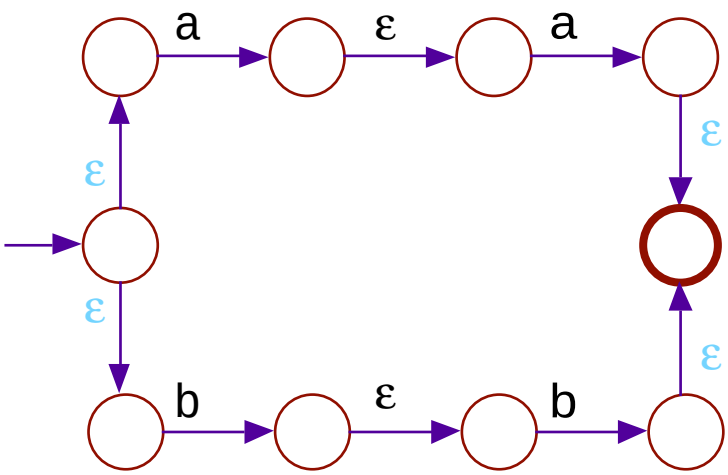
♦ *Exemplo:* a*(aa + bb)

ER	Autômato Finito correspondente
a	
b	
a*	
aa	

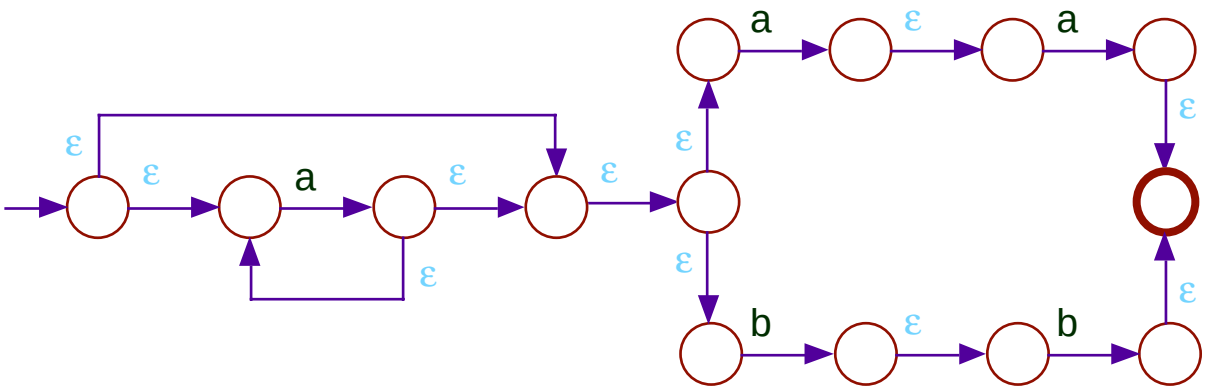
bb



(aa + bb)



- autômato correspondente a a*(aa + bb)



♦ *Teorema*

- Se L é uma LR,
- então existe uma ER r tal que GERA(r) = L

♦ Prova

- Não será demonstrado

2.6 Gramática Regular

♦ Gramáticas

- formalismo
  - axiomático ou
  - gerador
- permite definir linguagens
  - regulares
  - *não*-regulares
  - importante: é fácil definir gramáticas que geram linguagens não-regulares
- Linguagens Regulares
  - restrições nas regras de produção
  - tq definem exatamente a classe das LR

♦ *Definição. Gramáticas Lineares*

- Seja $G = (V, T, P, S)$ uma gramática e
  - sejam $A, B \in V$ (variáveis)
  - $w \in T^*$ (palavra de símbolos terminais)

  Então G é

- ***Gramática Linear à Direita (GLD)***
  - $A \rightarrow wB$ ou
  - $A \rightarrow w$

- ***Gramática Linear à Esquerda (GLE)***
  - $A \rightarrow Bw$ ou
  - $A \rightarrow w$

- ***Gramática Linear Unitária à Direita (GLUD)***
  - como na linear à direita
  - adicionalmente, $|w| \leq 1$

- ***Gramática Linear Unitária à Esquerda (GLUE)***
  - como na linear à esquerda
  - adicionalmente, $|w| \leq 1$

♦ Note-se que

- uma variável deriva, no máximo, uma variável

- ***Teorema.* Seja L uma linguagem**
  - 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
- **Ou seja**
  - as diversas formas das gramáticas lineares são formalismos equivalentes
  - demonstração: exercício
- ***Definição. Gramática Regular***
  - ou GR
  - qq gramática linear
- **Linguagem Gerada**
  - por uma gramática regular G
  - é representada por L(G) ou GERA(G)

♦ *Exemplo.* a(ba)*

- **GLD**. G = ({S, A}, {a, b}, P, S) onde P é tq
  - S → aA
  - A → baA | ε
- **GLE**. G = ({S}, {a, b}, P, S) onde P é tq
  - S → Sba | a
- **GLUD**. G = ({S, A, B}, {a, b}, P, S) onde P é tq
  - S → aA
  - A → bB | ε
  - B → aA
- **GLUE**. G = ({S, A}, {a, b}, P, S) onde P é tq
  - S → Aa | a
  - A → Sb

♦ *Exemplo.* (a + b)*(aa + bb)

- **GLD**. G = ({S, A}, {a, b}, P, S) onde P é tq
  - S → aS | bS | A
  - A → aa | bb
- **GLE**. G = ({S, A}, {a, b}, P, S) onde P é tq
  - S → Aaa | Abb
  - A → Aa | Ab | ε

♦ Gramáticas Regulares

- denotam exatamente as LR

♦ *Teorema.*

- Se L é gerada por uma GR,
- então L é uma LR

♦ Prova

- mostrar que
  - ∗ dado uma GR G qq,
  - ∗ é possível construir um AF M tq
  - ∗ ACEITA(M) = GERA(G)
- M simula as derivações de G
- demonstração de que ACEITA(M) = GERA(r)
  - ∗ indução no número de derivações

♦ Prova. Construção

- suponha $G = (V, T, P, S)$ uma GLUD
- seja AFε $M = (\Re, Q, \delta, q_0, F)$ tq
  - $\Re = T$
  - $Q = V \cup \{q_f\}$
  - $F = \{q_f\}$
  - $q_0 = S$
  - δ é como segue

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

♦ Prova. ¿ ACEITA(M) = GERA(G) ?

- indução no número de derivações
- suponha $\alpha \in (T \cup V)^*$
- suponha $w \in T^*$

♦ Base: $S \Rightarrow^1 \alpha$

- $\alpha = \varepsilon$ 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: $S \Rightarrow^n \alpha$

- $\alpha = w$ Suponha que $\underline{\delta}(S, w) = q_f$
- $\alpha = wA$ Suponha que $\underline{\delta}(S, w) = A$

♦ Indução: $S \Rightarrow^{n+1} \alpha$

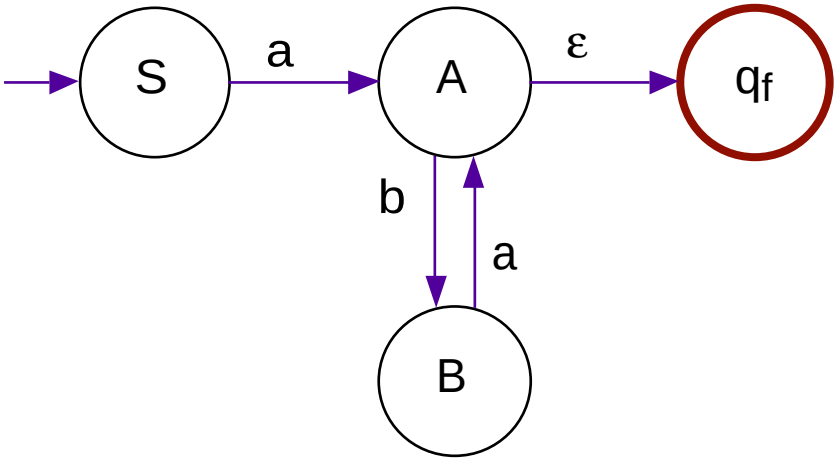
- então, $S \Rightarrow^n wA \Rightarrow^1 \alpha$
  - $\underline{\delta}(S, w) = A$ é a única hipótese que importa
- $\alpha = w\varepsilon = w$. Existe $A \rightarrow \varepsilon$. Logo,
  $\underline{\delta}(S, w\varepsilon) = \delta(\underline{\delta}(S, w), \varepsilon) = \delta(A, \varepsilon) = q_f$
- $\alpha = wb$. 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,
  $\underline{\delta}(S, wb) = \delta(\underline{\delta}(S, w), b) = \delta(A, b) = B$

♦ *Exemplo*

- G = ({S, A, B}, {a, b}, P, S)
  - ∗ S → aA
  - ∗ A → bB | ε
  - ∗ B → aA

- AFε M = ({a, b}, {S, A, B, q_f}, δ, S, {q_f}) onde δ é tq

Produção	Transição
S → aA	δ(S, a) = A
A → bB	δ(A, b) = B
A → ε	δ(A, ε) = q_f
B → aA	δ(B, a) = A



- ***Teorema:* Se L é uma LR, então existe G, GR que gera L**
  - dado um AFD M qq,
    - construção GR G
    - tq GERA(G) = ACEITA(M)
  - construção de uma GLUD
    - derivação simula função programa estendida
  - demonstração de que AGERA(G) = ACEITA(M)
    - indução no tamanho da palavra

♦ Prova. Construção

- suponha AFD $M = (\Sigma, Q, \delta, q_0, F)$ tq $ACEITA(M) = L$.
- seja $G = (V, T, P, S)$ uma GLUD tq
  - $V = Q \cup \{S\}$
  - $T = \Re$
  - P é tq (suponha $q_i, q_k \in Q$, $a \in \Re$ e $q_f \in F$)

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

♦ Prova. ¿ GERA(G) = ACEITA(M) ?

- indução no tamanho da palavra
- suponha w elemento de $\Re^*$

♦ Base $|w| = 0$

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

♦ Hipótese $|w| = n$ e $n > 1$

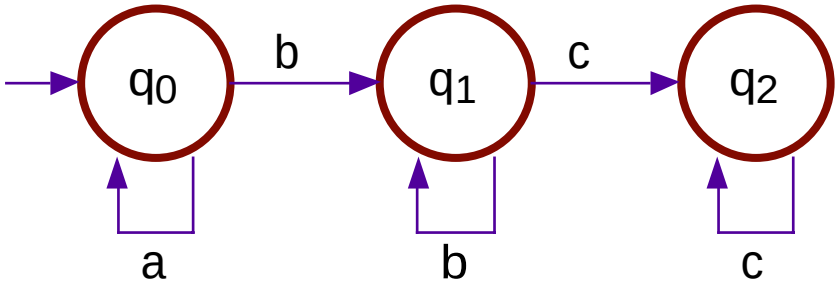
- seja w tal que $\underline{\delta}(q_0, w) = q$
  - q *não* é final, então suponha $S \Rightarrow wq$
  - q *é* final, então suponha $S \Rightarrow wq \Rightarrow w$

♦ Indução $|wa| = n + 1$

- seja w tal que $\underline{\delta}(q_0, wa) = p$
- então, $\delta(\underline{\delta}(q_0, w), a) = \delta(q, a) = p$
  - $S \Rightarrow wq$ é a única hipótese que importa
- Portanto, se:
  - p *não* é estado final
    então $S \Rightarrow^* wq \Rightarrow^1 wap$
  - p *é* estado final
    então $S \Rightarrow^* wq \Rightarrow^1 wap \Rightarrow^1 wa$

♦ *Exemplo:*

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



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

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