Linguagens Formais e Autômatos

P. Blauth Menezes

blauth@inf.ufrgs.br

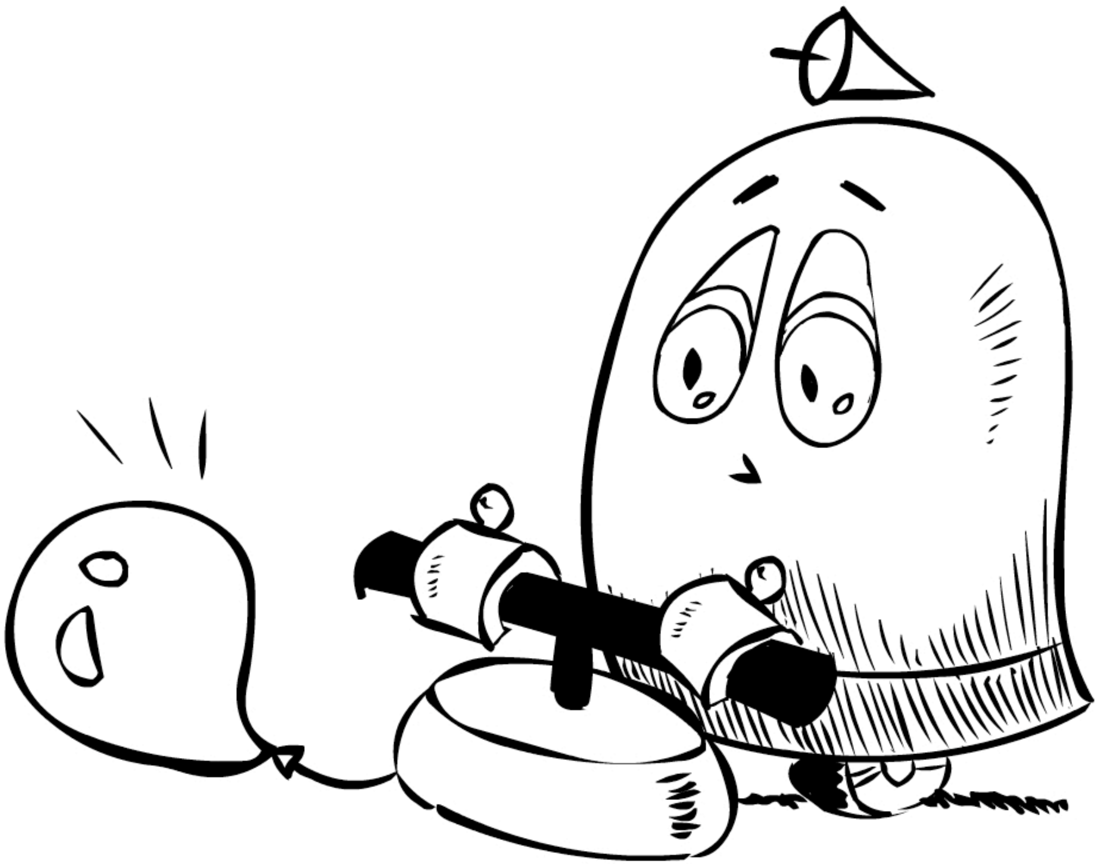
Departamento de Informática Teórica

Instituto de Informática / UFRGS

Linguagens Formais e Autômatos

P. Blauth Menezes

4 – Propriedades das Linguagens Regulares



4 – Propriedades das LR

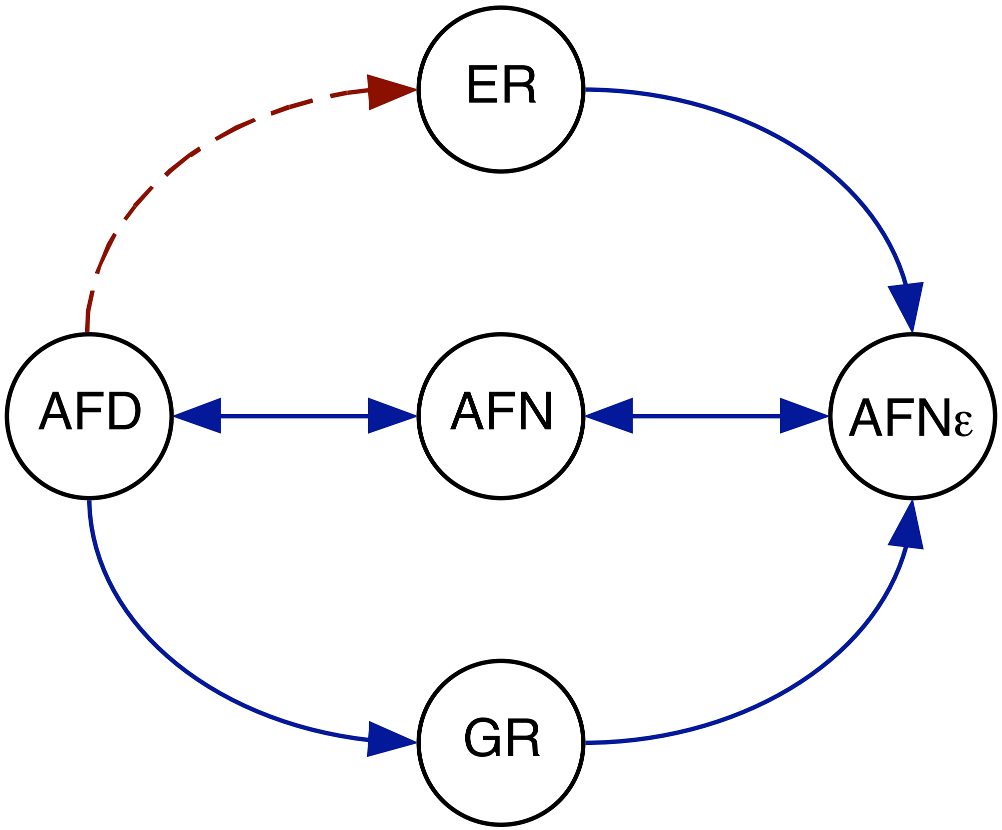
4 Propriedades das LR

- **LR podem ser representadas por formalismos**
  - pouca complexidade
  - grande eficiência
  - fácil implementação
- **Entretanto, por ser uma classe relativamente simples**
  - é restrita e limitada
  - fácil definir linguagens não-regulares

- **Assim, algumas questões sobre LR necessitam ser analisadas**
  - como determinar se uma linguagem é regular?
  - é fechada para operações de união, concatenação e intersecção?
  - como verificar se uma LR é infinita, finita ou vazia?
  - é possível analisar duas LR e concluir se são iguais ou diferentes?

◆ Análise de cada propriedade

- desenvolvida para um dos formalismos estudados
- para os demais: é suficiente traduzi-los



Obs: Autômato Finito × Complexidade de Algoritmos

Autômatos finitos pertencem à classe de algoritmos

- mais eficientes em termos de tempo de processamento
- supondo que toda a entrada necessita ser lida
  - se relaxada, podem-se imaginar formalismos mais eficientes
  - exemplo: reconhecer a linguagem Σ^* (por quê?)

Qq autômato finito que solucione um problema é igualmente eficiente

- a menos de eventual redundância de estados
- *não* influi no tempo de processamento

Redundância de estados pode ser facilmente eliminada

- Autômato Finito (Determinístico) Mínimo

4 – Propriedades das Linguagens Regulares

4.1 Bombeamento para as LR

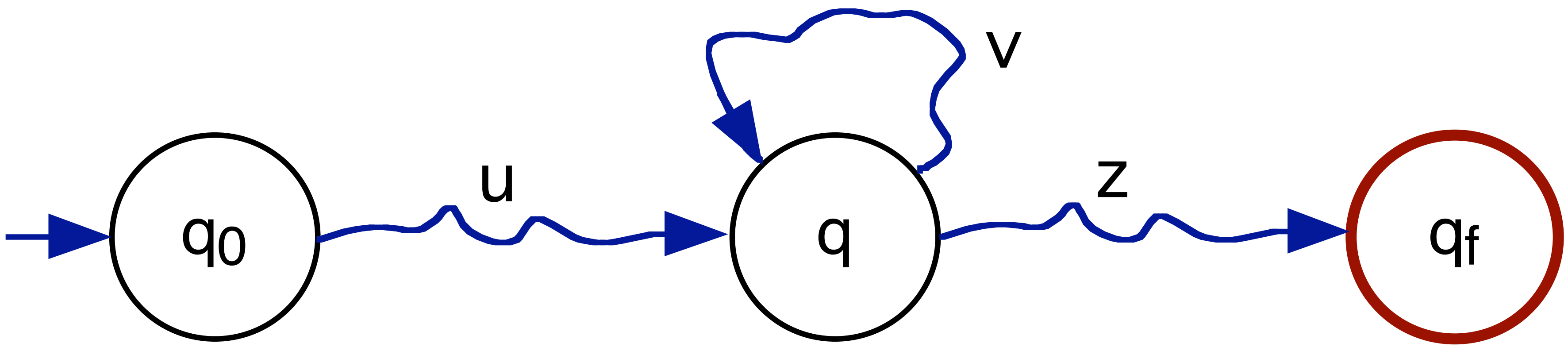
Lema do Bombeamento

- útil no estudo das propriedades das LR

Idéia básica

- se uma linguagem é regular, então
  - é aceita por um AFD com n estados
- se o autômato reconhece w de comprimento maior ou igual a n
  - assume algum estado q mais de uma vez
  - existe um ciclo na função programa que passa por q

- **w pode ser dividida em três subpalavras $w = u v z$**
  - $|uv| \leq n$, $|v| \geq 1$
  - v é a parte de w reconhecida pelo ciclo
  - tal ciclo pode ser executado (*bombeado*) zero ou mais vezes
    - para qualquer $i \geq 0$, $u v^i z$, é aceita pelo autômato



Lema do Bombeamento para as LR

Se L é LR, então:

- **existe** uma constante n tal que,
- para **qualquer** palavra w de L onde $|w| \geq n$,
- w pode ser definida como $w = u v z$
  - $|u v| \leq n$,
  - $|v| \geq 1$
- sendo que, **para todo** $i \geq 0$, $u v^i z$ é palavra de L

Prova: (*direta*)

Se L é uma LR, então **existe** um AFD $M = (\Sigma, Q, \delta, q_0, F)$ tal que

$$ACEITA(M) = L$$

Suponha que

- n é o cardinal de Q
- existe $w = a_1 a_2 \ldots a_m$ palavra de L de comprimento m tal que $m \geq n$
- $\delta(q_0, a_1) = q_1$
- $\ldots$
- $\delta(q_{m-1}, a_m) = q_m$

Como $m \geq n$, então existem r e s com $0 \leq r < s \leq n$ tais que

- $q_r = q_s$
- $\delta(q_0, a_1 \ldots a_r) = q_r$
- $\delta(q_r, a_{r+1} \ldots a_s) = q_s$
- $\delta(q_s, a_{s+1} \ldots a_m) = q_m$

ou seja, o autômato passa mais de uma vez no estado $q_r = q_s$

Sejam:

- $u = a_1 \ldots a_r$
- $v = a_{r+1} \ldots a_s$
- $z = a_{s+1} \ldots a_m$

Como $r < s \leq n$, então:

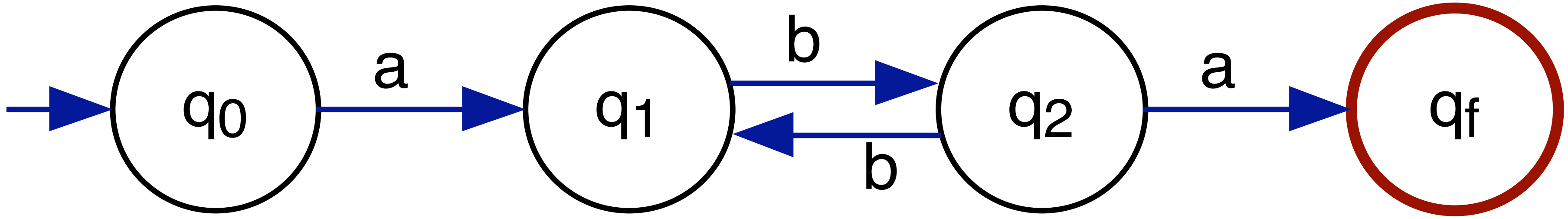
$$| v | \geq 1 \quad \text{e} \quad | u v | \leq n$$

Como $q_r = q_s$, então:

- v é reconhecida em um ciclo

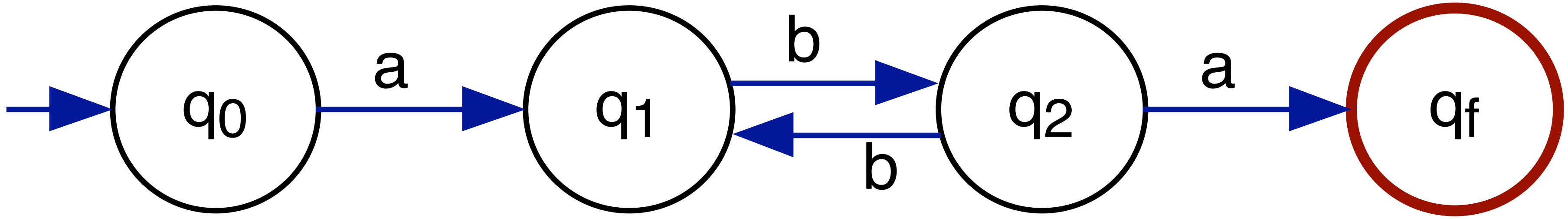
Portanto, para todo $i \geq 0$, $u v^i z$ é palavra de L

Exp: Bombeamento para as Linguagens Regulares



- $n = 4$
- para $w = abbba$???
  - $q_r = q_s = ??$
  - $u = ??$
  - $v = ??$
  - $z = ??$

Exp: Bombeamento para as Linguagens Regulares



- $n = 4$
- para $w = abbba$
  - $q_r = q_s = q_1$
  - $u = a$
  - $v = bb$
  - $z = ba$

◆ Diversos bombeamentos???

- duplo bombeamento: $\{ a^n b^m \mid n \geq 0 \text{ e } m \geq 0 \}$
- triplo bombeamento: $\{ a^n b^m a^r \mid n \geq 0, m \geq 0 \text{ e } r \geq 0 \}$

4 – Propriedades das Linguagens Regulares

4.1 Bombeamento para as LR
4.2 Investigação se é LR
4.3 Operações Fechadas sobre as LR
4.4 Investigação se uma LR é Vazia, Finita ou Infinita
4.5 Igualdade de LR
4.6 Minimização de um Autômato Finito

4.2 Investigação se é LR

- **Mostrar que é LR**
  - representar usando um dos formalismos regulares
- **Mostrar que *não* é LR**
  - desenvolvida para cada caso
  - ferramenta útil Lema do Bombeamento

Exp: Linguagem Não-Regular

$$L = \{ w \mid w \text{ possui o mesmo número de símbolos a e b} \}$$

Por absurdo. Suponha que L é regular

- existe AFD M com n estados que aceita L

Seja $w = a^n b^n$ sendo $| w | = 2n \geq n$. Logo (Bombeamento) $w = u v z$ tq

- $| u v | \leq n$
- $| v | \geq 1$
- para todo $i \geq 0$, $u v^i z$ é palavra de L

Absurdo !!! Como $| u v | \leq n$

- $u v$ é composta exclusivamente por símbolos a
- $u v^2 z$ *não* pertence a L (número de a será maior que o de b)

4 – Propriedades das Linguagens Regulares

4.3 Operações Fechadas sobre as LR

- **Operações sobre LR podem ser usadas para**
  - álgebra de LR
    - construir novas linguagens a partir de linguagens conhecidas
  - provar propriedades
  - construir algoritmos
- **Classe de Linguagens Regulares é fechada para**
  - união
  - concatenação
  - complemento
  - intersecção

Teorema: Operações Fechadas sobre as Linguagens Regulares

A Classe das Linguagens Regulares é fechada para

- união
- concatenação
- complemento
- intersecção

Prova:

União & Concatenação

Decorrem trivialmente da definição de expressão regular (por quê?)

Complemento: (*direta*)

Suponha L, LR sobre Σ^*. Então existe AFD

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

tal que $ACEITA(M) = L$

Idéia

inverter condições ACEITA/REJEITA de M para reconhecer ~L

- como fazer?
- e se δ for não-total (rejeitar por indefinição)?

Construção do AFD M_C tal que $\text{ACEITA}(M_C) = \sim L$

$$M_C = (\Sigma, Q_C, \delta_C, q_0, F_C)$$

- $Q_C = Q \cup \{ d \}$ (suponha $d \notin Q$)
- $F_C = Q_C - F$
- δ_C é como δ, adicionando as transições (para todo $a \in \Sigma$ e $q \in Q$)
  - $\delta_C(q, a) = d$ se $\delta(q, a)$ não é definida
  - $\delta_C(d, a) = d$

Claramente, o autômato finito M_C é tal que

$$\text{ACEITA}(M_C) = \sim L \quad \text{ou seja} \quad \text{ACEITA}(M_C) = \text{REJEITA}(M)$$

Intersecção: (*direta*)

Suponha L_1 e L_2 LR

Propriedade de DeMorgan para conjuntos

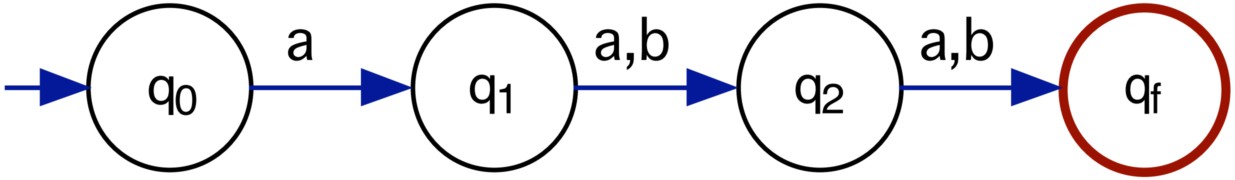
$$L_1 \cap L_2 = \sim(\sim L_1 \cup \sim L_2)$$

Como a Classe das LR é fechada para complemento e união

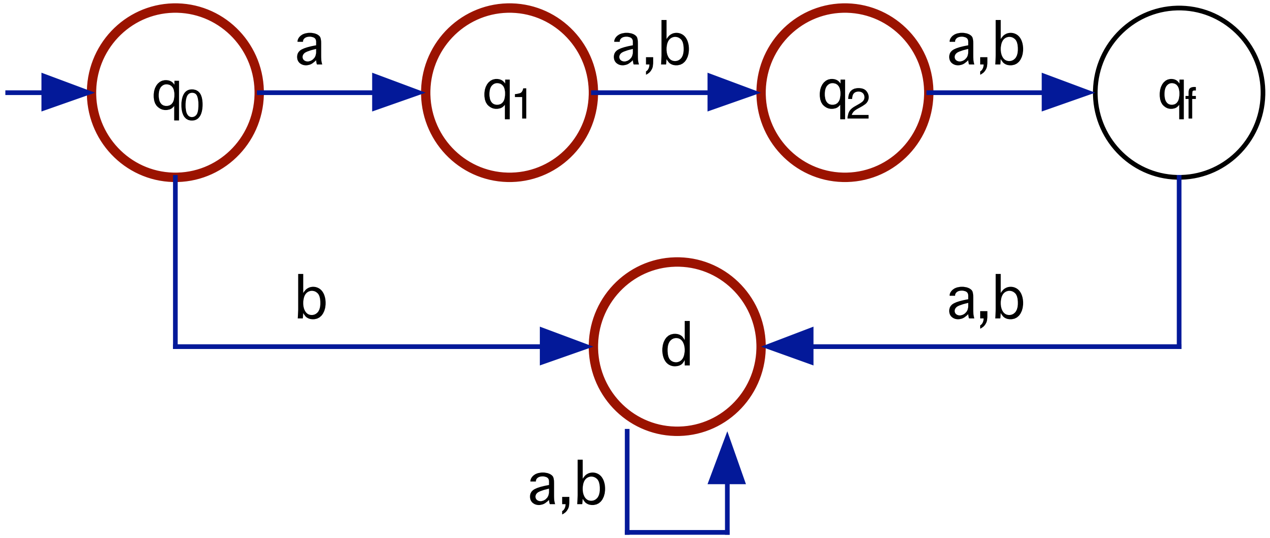
- então também é fechada para a intersecção

Exp: Complemento

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



$$M_C = (\{ a, b \}, \{ q_0, q_1, q_2, q_f, d \}, \delta_C, q_0, \{ q_0, q_1, q_2, d \})$$



4 – Propriedades das Linguagens Regulares

4.1 Bombeamento para as LR
4.2 Investigação se é LR
4.3 Operações Fechadas sobre as LR
4.4 Investigação se uma LR é Vazia, Finita ou Infinita
4.5 Igualdade de LR
4.6 Minimização de um Autômato Finito

4.4 Investigação se uma LR é Vazia, Finita ou Infinita

Teorema: Linguagem Regular Vazia, Finita ou Infinita

Se L é LR aceita por um autômato finito $M = (\Sigma, Q, \delta, q_0, F)$ com n estados, então L é

Vazia sse M *não aceita* qualquer palavra w tal que $|w| < n$

Finita sse M *não aceita* qualquer palavra w tal que $n \leq |w| < 2n$

Infinita sse M *aceita* palavra w tal que $n \leq |w| < 2n$

Prova:

Infinita sse M *aceita* palavra w tal que $n \leq | w | < 2n$

($\Leftarrow$ *direta*)

Verificar se M aceita alguma palavra w tal que $n \leq | w | < 2n$

- processar o autômato para todas as entradas neste intervalo
- se existe $w \in L$ pode ser definida como $w = u v z$
  - $| u v | \leq n$
  - $| v | \geq 1$
- para todo $i \geq 0$, $u v^i z$ é palavra de L

Logo, L é infinita

Infinita sse M *aceita* palavra w tal que $n \leq |w| < 2n$

($\Rightarrow$)

Se L é infinita, então existe w tal que $|w| \geq n$

Duas possibilidades

Caso 1 (*direta*): $|w| < 2n$

- prova está completa

Caso 2 (*por absurdo*): $|w| \geq 2n$

Suponha que

- não existe palavra de comprimento menor que $2n$
- w é a menor palavra tal que $|w| \geq 2n$

w pode ser definida como $w = u\,v\,z$

- $|uv| \leq n$ e $|v| \geq 1$
- em, particular, $1 \leq |v| \leq n$

Logo, $u\,z$ é palavra de L. Absurdo!!!

- $|uz| \geq 2n$
  - contradiz a suposição: w a menor palavra tal que $|w| \geq 2n$
- $|uz| < 2n$
  - $n \leq |uz| < 2n$ (pois $|uvz| \geq 2n$, $1 \leq |v| \leq n$)
  - contradiz a suposição: não existe w tal que $n \leq |w| < 2n$

Vazia sse M *não aceita* qualquer palavra w tal que $|w| < n$

Processa M para todas as palavras de comprimento menor que n

Se rejeita todas as palavras: linguagem vazia

- detalhamento da prova: exercício

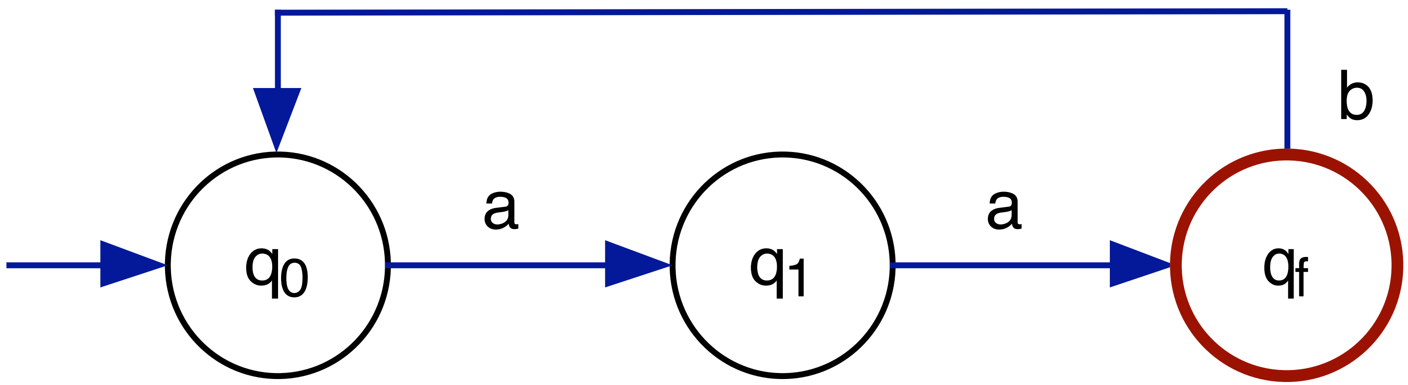
Finita sse M *não aceita* qualquer palavra w tal que $n \leq |w| < 2n$

(*por contraposição*)

$$(p \leftrightarrow q) \Leftrightarrow (\neg p \leftrightarrow \neg q)$$

Mesma prova do caso Infinita (por que?)

Exp: Linguagem Regular Infinita



A linguagem é infinita sse aceita palavra w tal que $n \leq |w| < 2n$

- aabaa é aceita
- $3 \leq |aabaa| < 6$

Logo, a linguagem é infinita

4 – Propriedades das Linguagens Regulares

4.5 Igualdade de Linguagens Regulares

- **Teorema mostra que**
  - existe um algoritmo para verificar se dois autômatos finitos são equivalentes
    - reconhecem a mesma linguagem
- **Importante conseqüência**
  - existe um algoritmo que permite verificar se duas implementações são equivalentes

Teorema: Igualdade de Linguagens Regulares

Se M_1 e M_2 são AF, então existe um algoritmo para determinar se

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

Prova: (*direta*)

Suponha M_1 e M_2 AF tais que $ACEITA(M_1) = L_1$ e $ACEITA(M_2) = L_2$

Portanto, é possível construir um AF M_3 tal que $ACEITA(M_3) = L_3$

$$L_3 = (L_1 \cap \sim L_2) \cup (\sim L_1 \cap L_2)$$

Claramente, $L_1 = L_2$ sse L_3 é vazia

- existe um algoritmo para verificar se uma LR é vazia

4 – Propriedades das Linguagens Regulares

4.1 Bombeamento para as LR
4.2 Investigação se é LR
4.3 Operações Fechadas sobre as LR
4.4 Investigação se uma LR é Vazia, Finita ou Infinita
4.5 Igualdade de LR
4.6 Minimização de um Autômato Finito

4.6 Minimização de um AF

◆ Simulador de AFD

- exercício

- algoritmo que controla
  - ⁎ mudança de estado do autômato
  - ⁎ cada símbolo lido da entrada

- tempo de processamento para aceitar ou rejeitar
  - ⁎ diretamente proporcional ao tamanho da entrada

◆ Complexidade de algoritmos

- autômatos finitos pertencem à classe de algoritmos mais eficientes
  - em termos de tempo de processamento
  - *supondo* que toda a fita de entrada necessita ser lida

◆ Tempo de processamento

- *não* depende do autômato considerado
- *qualquer* AFD
  - que reconheça a linguagem
  - terá a *mesma eficiência*

◆ Otimização possível

- minimização do número de estados

◆ AFD Mínimo ou Autômato Finito Mínimo

- AFD equivalente, com o menor número de estados possível

◆ Minimização em algumas aplicações especiais

- não necessariamente o menor custo de implementação

- exemplo: circuitos lógicos ou redes lógicas
  - pode ser desejável introduzir estados intermediários
  - de forma a melhorar eficiência ou facilitar ligações físicas

- prever variáveis específicas da aplicação

◆ Autômato finito mínimo é único

- a menos de isomorfismo
- diferenciando-se, eventualmente, na identificação dos estados

◆ Algoritmo de minimização

- unifica os estados equivalentes

◆ Estados equivalentes

- processamento de uma entrada qualquer
- a partir de estados equivalentes
- resulta na mesma condição de aceita/

Def: Estados Equivalentes

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

q e p de Q são Estados Equivalentes sse, para qualquer $w \in \Sigma^*$

$$\delta(q, w) \quad \text{e} \quad \delta(p, w)$$

resultam simultaneamente em estados finais, ou não-finais

Def: Autômato Finito Mínimo

L linguagem regular. O Autômato Finito Mínimo é um AFD

$$M_m = (\Sigma, Q_m, \delta_m, q_{0_m}, F_m)$$

tal que

- $\text{ACEITA}(M_m) = L$
- para qualquer AFD $M = (\Sigma, Q, \delta, q_0, F)$ tal que $\text{ACEITA}(M) = L$

$$\#Q \geq \#Q_m$$

Obs: Pré-Requisitos do Algoritmo de Minimização

- determinístico
- estados alcançáveis a partir do estado inicial
- função programa total

◆ Caso não satisfaça algum dos pré-requisitos

- gerar um autômato determinístico equivalente
  - algoritmos de tradução apresentados nos teoremas
- eliminar estados inacessíveis (e transições): exercício
- função programa total
  - introduzir um estado não-final d
  - incluir transições não-previstas, tendo d como estado destino
  - incluir um ciclo em d para todos os símbolos do alfabeto

◆ Algoritmo de minimização

- identifica os estados equivalentes por exclusão
- tabela de estados
  - ⁎ marca estados não-equivalentes
  - ⁎ entradas não-marcadas: estados equivalentes

Def: Algoritmo de Minimização

$M = (\Sigma, Q, \delta, q_0, F)$ AFD que satisfaz aos pré-requisitos

Passo 1: Construção da Tabela: relaciona estados distintos

q_1					
q_2					
...					
q_n					
d					
	q_0	q_1	...	q_{n-1}	q_n

Passo 2: Marcação dos Estados Trivialmente Não-Equivalentes

- pares do tipo { estado final, estado não-final }

Passo 3: Marcação dos Estados Não-Equivalentes

Para $\{ q_u, q_v \}$ não-marcado e $a \in \Sigma$, suponha que

$$\delta(q_u, a) = p_u \quad \text{e} \quad \delta(q_v, a) = p_v$$

- $p_u = p_v$
  - q_u é equivalente a q_v para a: não marcar
- $p_u \neq p_v$ e $\{ p_u, p_v \}$ não está marcado
  - $\{ q_u, q_v \}$ incluído na lista encabeçada por $\{ p_u, p_v \}$
- $p_u \neq p_v$ e $\{ p_u, p_v \}$ está marcado
  - $\{ q_u, q_v \}$ não é equivalente: marcar
  - se $\{ q_u, q_v \}$ encabeça uma lista: marcar todos os pares da lista (e, recursivamente, se algum par da lista encabeça outra lista)

Passo 4: Unificação dos Estados Equivalentes

Pares não-marcados são equivalentes

- equivalência de estados é transitiva
- pares de estados não-finais equivalentes
  - ∗ um único estado não-final
- pares de estados finais equivalentes
  - ∗ um único estado final
- se algum dos estados equivalentes é inicial
  - ∗ estado unificado é inicial
- transições com origem (destino) em um estado equivalente
  - ∗ origem (destino) no estado unificado

Passo 5: Exclusão dos Estados Inúteis

q é um estado inútil

- não-final
- a partir de q não é possível atingir um estado final
- d (se incluído) é inútil

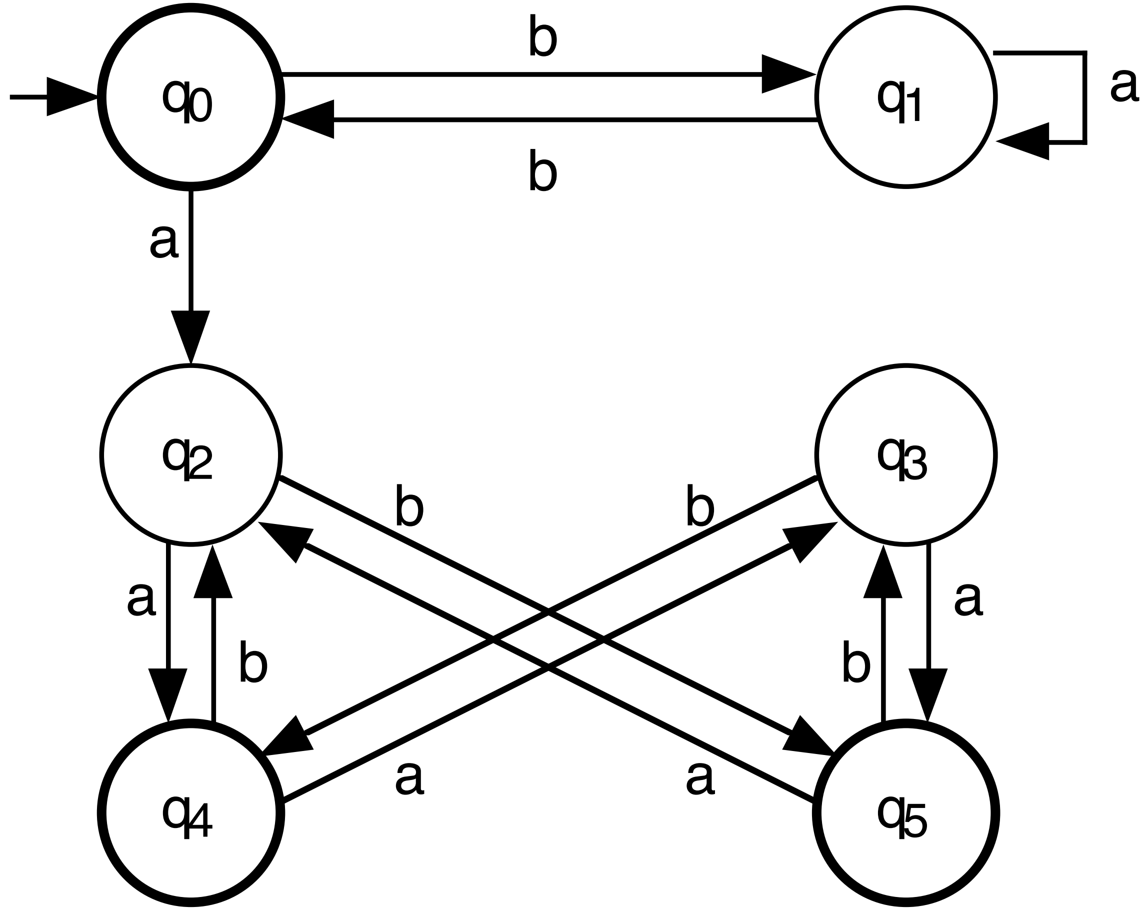
Transições com origem ou destino em estado inútil

- excluir

Algoritmo para excluir os estados inúteis

- exercício

Exp: Minimização



Pré-requisitos de minimização ???

Passo 1. Construção da tabela

Passo 2. Marcação dos pares { estado final, estado não-final }

	q_0	q_1	q_2	q_3	q_4
q_1	×				
q_2	×				
q_3	×				
q_4		×	×	×	
q_5		×	×	×	

Passo 3. Análise dos pares de estado não-marcados

$\{ q_0, q_4 \}$

$$\delta(q_0, a) = q_2 \quad \delta(q_0, b) = q_1$$
$$\delta(q_4, a) = q_3 \quad \delta(q_4, b) = q_2$$

- $\{ q_1, q_2 \}$ e $\{ q_2, q_3 \}$ são não-marcados
  - ∗ inclui $\{ q_0, q_4 \}$ nas listas de $\{ q_1, q_2 \}$ e $\{ q_2, q_3 \}$

$\{ q_0, q_5 \}$

$$\delta(q_0, a) = q_2 \quad \delta(q_0, b) = q_1$$
$$\delta(q_5, a) = q_2 \quad \delta(q_5, b) = q_3$$

- $\{ q_1, q_3 \}$ é não-marcado (e $\{ q_2, q_2 \}$ é trivialmente equivalente)
  - ∗ inclui $\{ q_0, q_5 \}$ na lista de $\{ q_1, q_3 \}$

$\{ q_1, q_2 \}$

$$\delta(q_1, a) = q_1 \quad \delta(q_1, b) = q_0$$
$$\delta(q_2, a) = q_4 \quad \delta(q_2, b) = q_5$$

- $\{ q_1, q_4 \}$ é marcado: marca $\{ q_1, q_2 \}$
- $\{ q_1, q_2 \}$ encabeça uma lista: marca $\{ q_0, q_4 \}$

$\{ q_1, q_3 \}$

$$\delta(q_1, a) = q_1 \quad \delta(q_1, b) = q_0$$
$$\delta(q_3, a) = q_5 \quad \delta(q_3, b) = q_4$$

- $\{ q_1, q_5 \}$ e $\{ q_0, q_4 \}$ são marcados: marca $\{ q_1, q_3 \}$
- $\{ q_1, q_3 \}$ encabeça uma lista: marca $\{ q_0, q_5 \}$

$\{ q_2, q_3 \}$

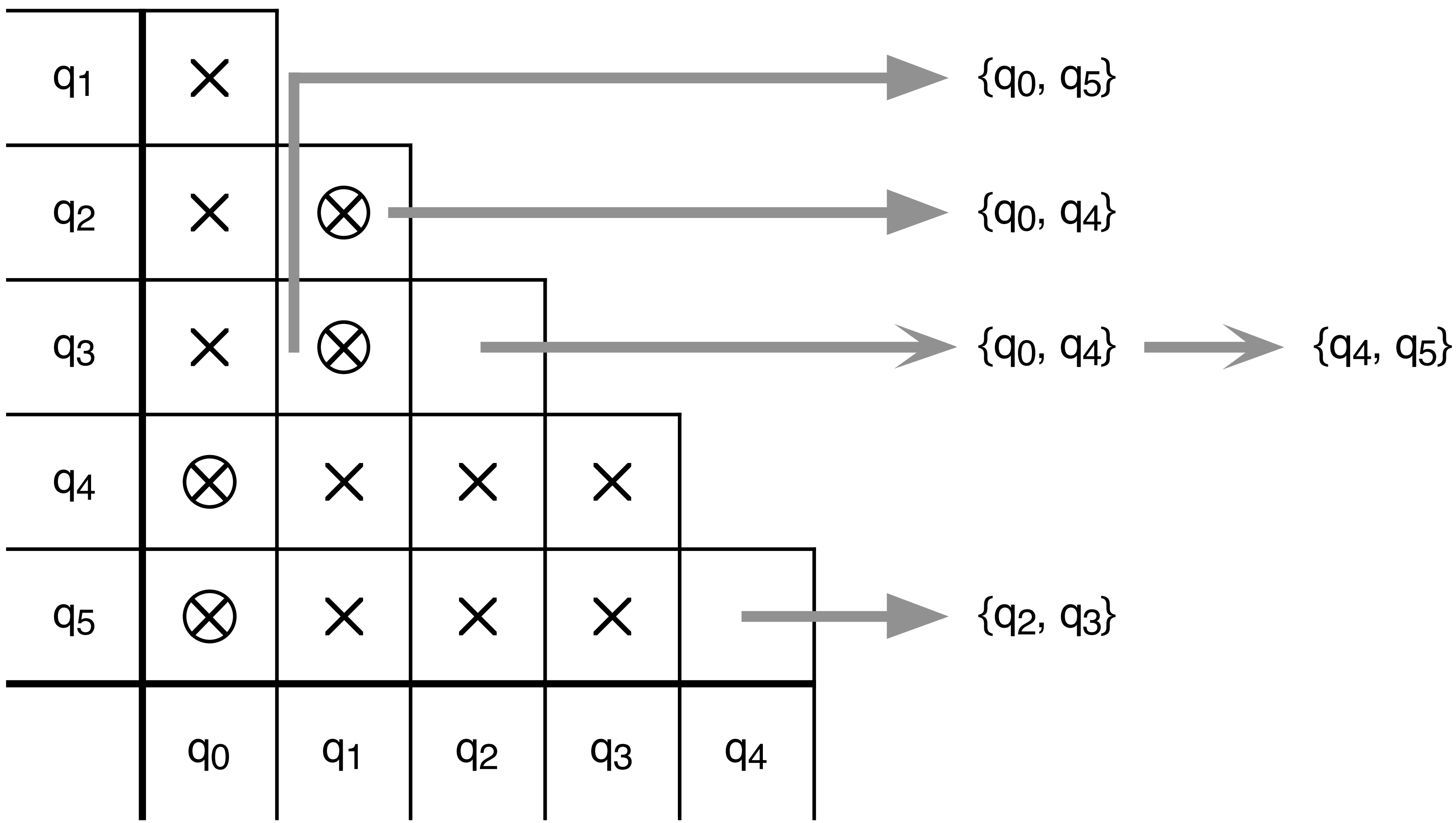
$$\delta(q_2, a) = q_4 \quad \delta(q_2, b) = q_5$$
$$\delta(q_3, a) = q_5 \quad \delta(q_3, b) = q_4$$

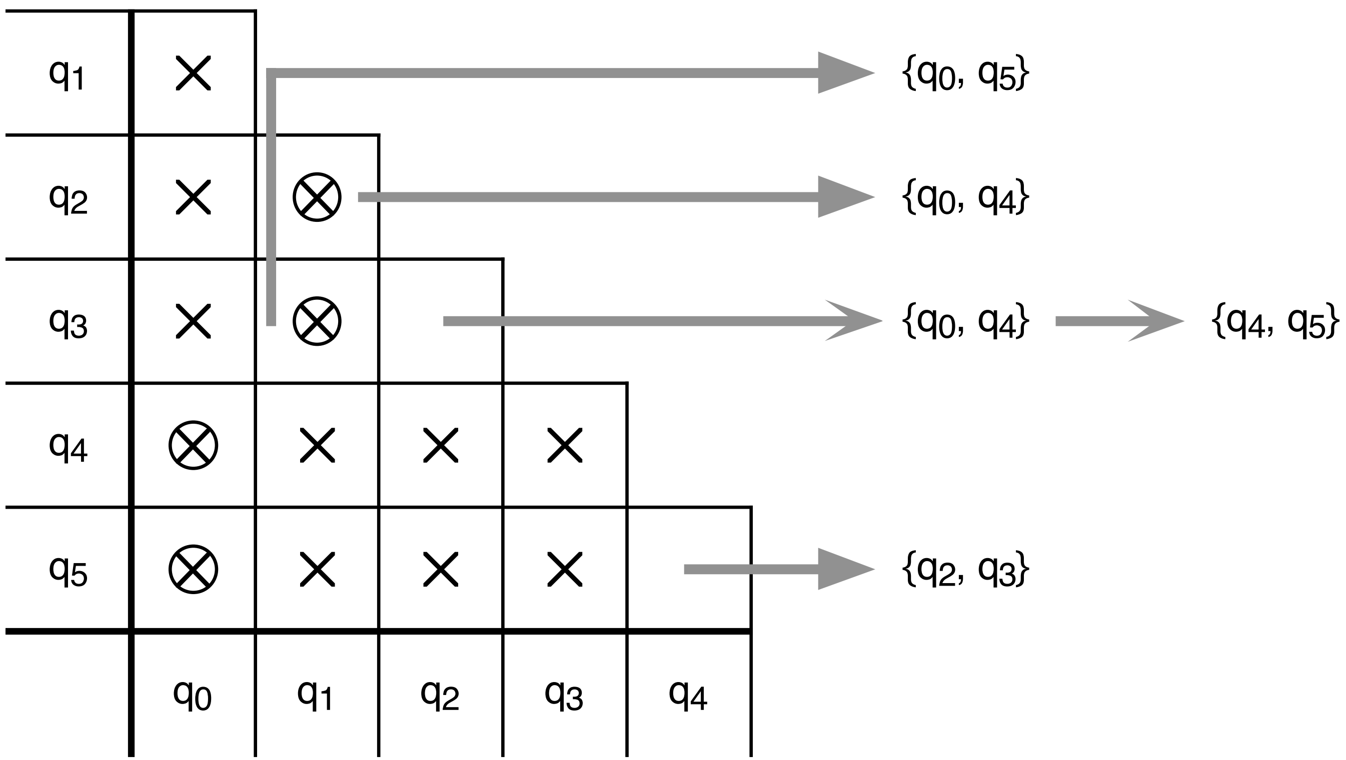
- $\{ q_4, q_5 \}$ é não-marcado: inclui $\{ q_2, q_3 \}$ na lista de $\{ q_4, q_5 \}$

$\{ q_4, q_5 \}$

$$\delta(q_4, a) = q_3 \quad \delta(q_4, b) = q_2$$
$$\delta(q_5, a) = q_2 \quad \delta(q_5, b) = q_3$$

- $\{ q_2, q_3 \}$ é não-marcado: inclui $\{ q_4, q_5 \}$ na lista de $\{ q_2, q_3 \}$

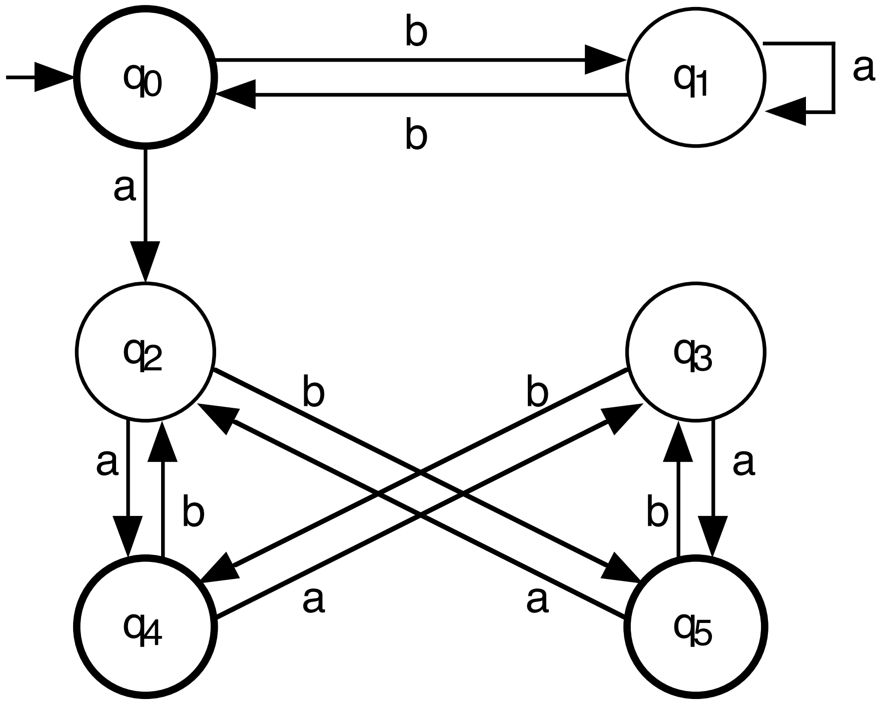

q1
q2
q3
q4
q5
q0
q1
q2
q3
q4
{q0, q5}
{q0, q4}
{q0, q4}
{q4, q5}
{q2, q3}

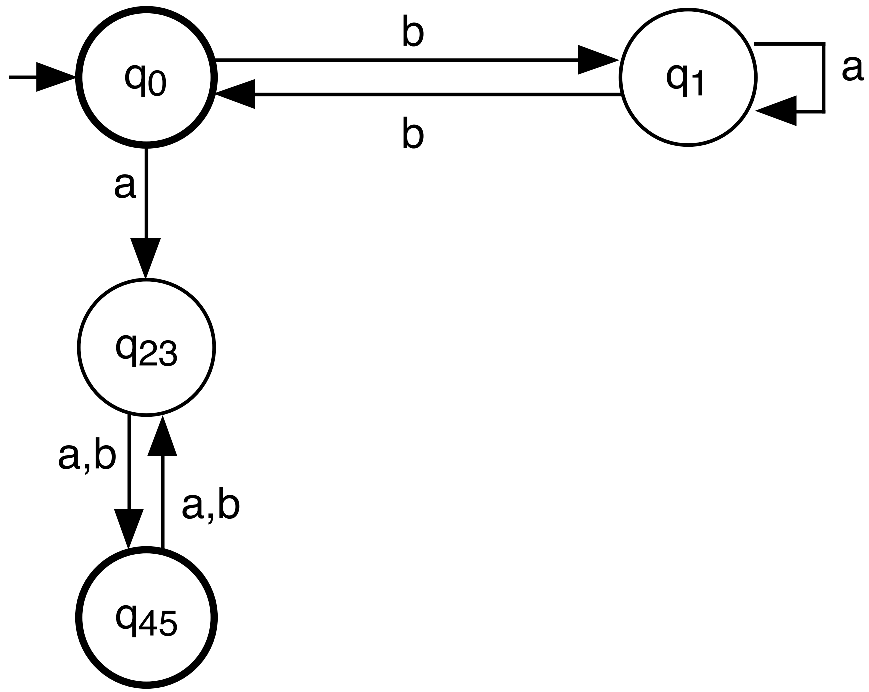


Passo 4. $\{ q_2, q_3 \}$ e $\{ q_4, q_5 \}$ são não-marcados

- q_{23}: unificação dos estados q_2 e q_3
- q_{45}: unificação dos estados finais q_4 e q_5

Autômato mínimo resultante





Teorema: Autômato Finito Mínimo

O autômato construído usando o algoritmo de minimização

- AFD com menor número de estados que aceita a linguagem

Teorema: Unicidade do Autômato Finito Mínimo

AFD mínimo de uma linguagem

- único, a menos de isomorfismo

◆ Isomorfismo de AFD

- diferenciam-se, eventualmente, na identificação dos estados
- definição formal: não será apresentada
- como um autômato finito mínimo é único, a menos de isomorfismo
  - usual ser referido como *o* (e não como *um*) autômato mínimo

Linguagens Formais e Autômatos

P. Blauth Menezes

Linguagens Formais e Autômatos

P. Blauth Menezes

blauth@inf.ufrgs.br

Departamento de Informática Teórica

Instituto de Informática / UFRGS