2.7 Propriedades das Linguagens Regulares

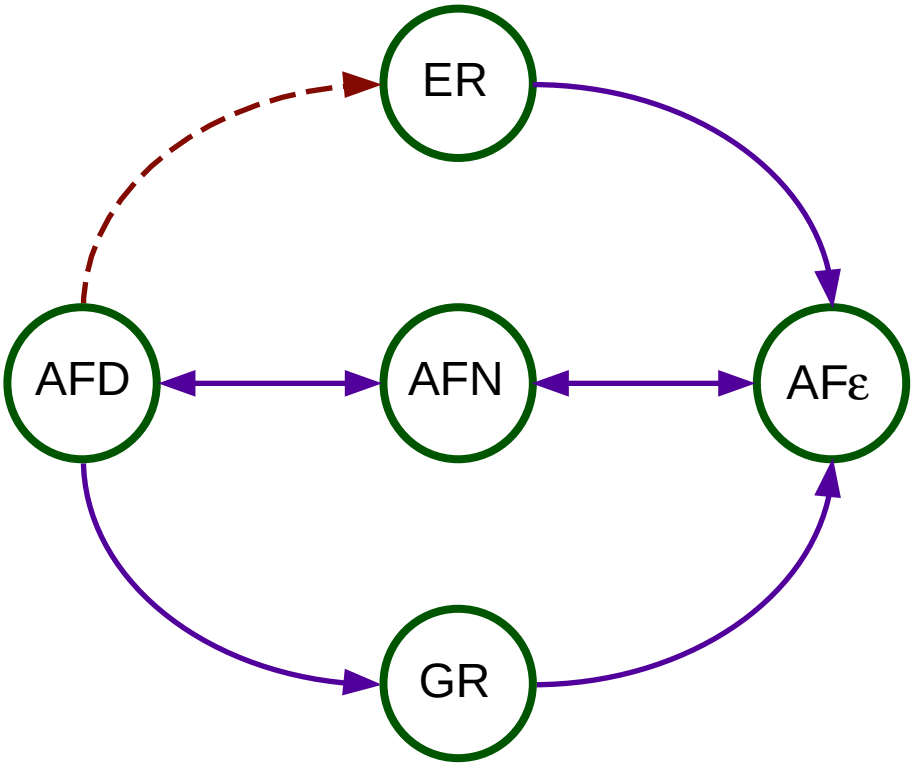
- **LR podem ser representadas por formalismos**
  - baixa complexidade
  - grande eficiência
  - fácil implementação
- **Entretanto, é uma classe de linguagens**
  - restrita
  - limitada
  - é fácil definir linguagens que *não* são regulares

♦ Algumas questões sobre LR necessitam ser analisadas

- como determinar se uma
  - linguagem *é regular*?
- como verificar se uma LR é
  - *infinita*?
  - *finita*?
  - *vazia*?
- é possível analisar duas LR e concluir se são
  - *iguais*?
  - *diferentes*?
- ou seja, é possível verificar
  - se duas especificações são equivalentes?
  - se dois compiladores são equivalentes?
  - se o compilador aceita exatamente a linguagem especificada?
- a classe das LR é fechada para operações de
  - *união*?
  - *concatenação*?
  - *intersecção*?

♦ A análise de cada propriedade

- é desenvolvida para *um* formalismo
- demais formalismos
  - ∗ é suficiente traduzir
  - ∗ AFD $\rightarrow$ ER não foi apresentada



♦ Bombeamento para as LR

- proposição útil no estudo das propriedades
- LR
  - é aceita por um AFD com n estados
- AFD reconhece w tq $|w| \geq n$
  - obrigatoriamente assume algum estado q mais de uma vez
  - existe um ciclo no programa que passa por q
- logo, w pode ser dividida em três subpalavras
  - $w = uvz$
  - $|uv| \leq n$ e $|v| \geq 1$
  - v é a parte reconhecida pelo ciclo
- claramente uv^iz para qq $i \geq 0$
  - é aceita pelo AFD
  - executando o ciclo i vezes

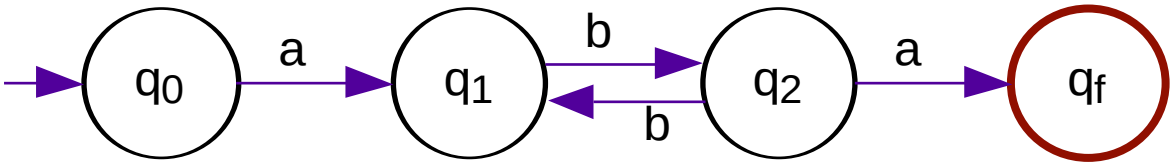
♦ *Teorema*: Se L é LR, então

- existe n tq,
- para qq $w \in L$ onde $|w| \geq n$,
- w pode ser definida como $w = uvz$
  - $|uv| \leq n$ e $|v| \geq 1$
- para todo $i \geq 0$, $uv^iz \in L$

♦ Prova

- L é LR
  - existe AFD $M = (\Sigma, Q, \delta, q_0, F)$ tq $L(M) = L$
- seja n o cardinal de Q
- seja $w = a_1a_2...a_m \in L$ tq $m \geq n$
- suponha $\delta(q_0,a_1) = q_1$, $\delta(q_1,a_2) = q_2$,...,$\delta(q_{m-1},a_m) = q_m$
- como $m \geq n$ existem r, s onde $0 \leq r < s \leq n$ tq
  - $q_r = q_s$
  - $\underline{\delta}(q_0, a_1...a_r) = q_r$, $\underline{\delta}(q_r, a_{r+1}...a_s) = q_s$
  - $\underline{\delta}(q_s, a_{s+1}...a_m) = q_m$
- sejam $u = a_1...a_r$, $v = a_{r+1}...a_s$ e $z = a_{s+1}...a_m$
- como $r < s \leq n$, então $|v| \geq 1$ e $|uv| \leq n$
- como $q_r = q_s$, então
  - v é reconhecida em um ciclo
  - portanto, $uv^iz \in L$, para todo $i \geq 0$

♦ Exemplo. Bombeamento das LR



- n = 4
- no caso particular de w = abbba
  - $q_r = q_s = q_1$
  - u = a
  - v = bb
  - z = ba

Linguagem Regular × Não-Regular

- ***É* regular**
  - é suficiente representar usando
    - autômato finito
    - expressão regular
    - gramática regular
- ***Não* é regular**
  - necessita ser desenvolvida para cada caso
  - algumas ferramentas específicas
    - ex: bombeamento
    - em geral, demonstração por absurdo

♦ *Exemplo*: Não é LR

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

♦ Prova

- bombeamento + absurdo
- suponha que L é LR
- então existe AFD M com n estados que aceita L
- seja $w = a^n b^n$
- pelo bombeamento
  - $w = uvz$ onde $|uv| \leq n$, $|v| \geq 1$
  - para todo $i \geq 0$, uv^iz é palavra de L
- ABSURDO!!!
  - $|uv| \leq n$ e uv é composta exclusivamente por símbolos a
  - então, por exemplo uv^2z, *não* pertence a L (não possui o mesmo número de símbolos a e b)

Operações Fechadas sobre LR

- **Obs**
  - para uma linguagem L sobre Σ^*,
    L' denota o seu complemento em Σ^*
- ***Teorema*: A classe das LR é fechada para**
  - união
  - concatenação
  - complemento
  - intersecção

♦ Prova

- *união* e *concatenação*
  - decorrem trivialmente da definição de ER
- *complemento*
  - a idéia consiste em inverter as condições de ACEITA/REJEITA do AF
- inversão das condições ACEITA/REJEITA
  - modifica o AF, garantindo que somente irá parar ao terminar de ler toda a entrada
  - transforma os estados finais em não-finais e vice-versa
- *intersecção*
  - tratar a intersecção em termos da união e complemento

♦ Complemento

- seja L uma LR sobre Σ^*
- seja M = $(\Sigma, Q, F, q_0, \delta)$ um AFD tq ACEITA(M) = L
- seja M' = $(\Sigma, Q', F', q_0, \delta')$ tq ACEITA(M') = L'
  - Q' = Q $\cup$ {d}
  - F' = Q' - F
  - δ' é como δ, excetuando-se

    $\delta'(q, a) = d$ se $\delta(q, a)$ não é definida

    $\delta'(d, a) = d$

♦ Intersecção

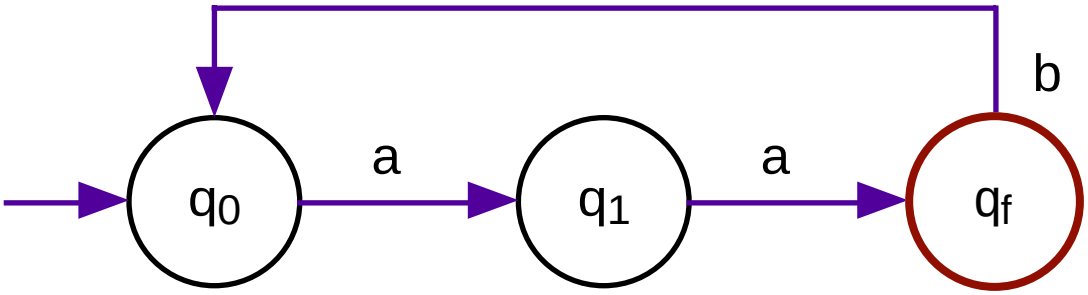
- Sejam L_1 e L_2 LR
- $L_1 \cap L_2 = (L_1' \cup L_2')'$
- como a classe das LR é fechada para
  - complemento
  - união

  então é fechada para a intersecção

Investigação se uma LR é Vazia, Finita ou Infinita

- ***Teorema*: Se L é LR aceita por um AF M com n estados, então L é:**
  - **vazia** sse M *não* aceita qualquer w tq $|w| < n$
  - **finita** sse M *não* aceita w tq $n \leq |w| < 2n$
  - **infinita** sse M aceita w tq $n \leq |w| < 2n$
- **Ou seja, prova-se que**
  - *existe um algoritmo* para verificar se uma LR representada por um AF é *vazia*, *finita* ou *infinita*
- **Pelo bombeamento**
  - se aceita w tq $|w| \geq n$
  - então a linguagem é infinta
  - mas como verificar se existe tal w?

♦ *Exemplo*. LR Infinita



- infinita sse aceita w tq $n \leq |w| < 2n$
- menor palavra aceita de comprimento maior ou igual a 3
  - $aabaa$
  - possui comprimento 5
  - ou seja, $3 \leq |aabaa| < 6$

- ♦ *Aceita* w tq $n \leq |w| < 2n$, então é *Infinita*
  - seja $w \in L$ tq $n \leq |w| < 2n$
  - bombeamento
    - $w = uvz$ onde $|uv| \leq n$, $|v| \geq 1$
    - para todo $i \geq 0$, uv^iz é palavra de L
  - logo, L é *infinita*

- ♦ *Infinita*, então *aceita* w tq $n \leq |w| < 2n$
  - se é infinita, então existe w tq $|w| \geq n$
  - se $|w| < 2n$, então a prova esta completa
  - suponha que *não* existe w tq $|w| < 2n$
  - suponha $|w| \geq 2n$
    - mas de comprimento menor ou igual a qualquer outra t tq $|t| \geq 2n$
  - então $w = uvz$ onde $|uv| \leq n$, $|v| \geq 1$ e $uv^iz \in L$
  - em, particular, $1 \leq |v| \leq n$ e $uz \in L$
  - ABSURDO!!!, pois relativamente a uz tem-se que:
  - se $|uz| \geq 2n$
    - w *não* é a menor palavra tq $|w| \geq 2n$!!!
  - se $|uz| < 2n$
    - como $|uvz| \geq 2n$ e $1 \leq |v| \leq n$
    - então $n \leq |uz| < 2n$
    - logo existe $w = uz$ tq $n \leq |w| < 2n$

- *Vazia* sse *não* aceita qualquer w tq $|w| < n$
  - processa M
    - para todo w tq $|w| < n$
  - se rejeita todas as palavras
    - a linguagem é *vazia*
  - prova
    - simples usando o bombeamento
    - exercício
- *Finita* sse *não* aceita w tq $n \leq |uv| < 2n$
  - conseqüência direta do caso infinita
    - negação do caso *infinita*
  - processa M
    - para todo w tq $n \leq |w| < 2n$
  - se rejeita todas as palavras
    - a linguagem é *finita*

Igualdade de Linguagens Regulares

♦ ***Teorema.***

- se M_1 e M_2 são AF
- então existe algoritmo para determinar se
  - $ACEITA(M_1) = ACEITA(M_2)$

♦ **Portanto**

- existe um algoritmo para verificar se
  - dois AF são equivalentes
  - reconhecem a mesma linguagem

♦ **Prova**

- sejam M_1 e M_2 AF tq
  - $ACEITA(M_1) = L_1$ e $ACEITA(M_2) = L_2$
- seja $L_3 = (L_1 \cap L_2') \cup (L_1' \cap L_2)$
- portanto, é possível construir um AF M_3 tq
  - $ACEITA(M_3) = L_3 = (L_1 \cap L_2') \cup (L_1' \cap L_2)$
- claramente, $L_1 = L_2$ sse L_3 é *vazia*
  - existe algoritmo para verificar se LR é vazia
  - existe algoritmo pra $\cup$, $\cap$ e complemento

Eficiência de um AF como Algoritmo de Reconhecimento

- **Simulador de qualquer AFD**
  - fácil implementação
  - algoritmo que
    - controla a mudança de estado
    - a cada símbolo lido da entrada
- **Tempo de processamento**
  - para aceitar ou rejeitar
    - diretamente proporcional ao tamanho da entrada
  - em termos de Complexidade
    - parte da Teoria da Computação que estuda os recursos necessários ao processamento
    - pertence à mais rápida classe de algoritmos

♦ Tempo de processamento

- *não* depende do AFD
- qq AFD que reconheça a linguagem
  - terá a mesma eficiência

♦ Otimização?

- redução do número de estados
- existe um algoritmo para construir
  - AFD mínimo
  - AFD com o menor número de estados

2.8 Minimização de um Autômato Finito

♦ Objetivo

- gerar um AF equivalente
- com o menor número de estados possível

♦ Minimização do número de estados

- adotada na maioria das soluções práticas
- entretanto, em algumas aplicações
  - minimizar do número de estados pode não implicar no menor custo de implementação
- exemplo
  - desenho de circuitos eletrônicos
  - pode ser desejável introduzir estados intermediários
  - para melhorar a eficiência
  - ou simplesmente facilitar as ligações físicas
- nestes casos o algoritmo deve ser modificado
  - prevendo as variáveis específicas da aplicação

♦ O autômato mínimo é único

- a minimização de AF distintos
  - que aceitam a mesma linguagem
  - geram o mesmo AF mínimo

♦ Idéia básica do algoritmo

- unificar os *estados equivalentes*

♦ *Definição. Estados Equivalentes*

- q e p são equivalentes sse
  - para qq w,
  - $\delta(q, w)$ e $\delta(p, w)$
  - resultam simultaneamente em estados finais, ou não-finais
- ou seja
  - processamento de uma entrada qq
  - a partir de estados equivalentes
  - gera o mesmo resultado aceita/rejeita

♦ *Pré-Requisitos do Algoritmo*

- AF deve ser determinístico
- não pode ter estados inacessíveis
  - não-atingíveis a partir do estado inicial
- a função programa deve ser total

♦ Caso o AF não satisfaça algum dos pré-requisitos

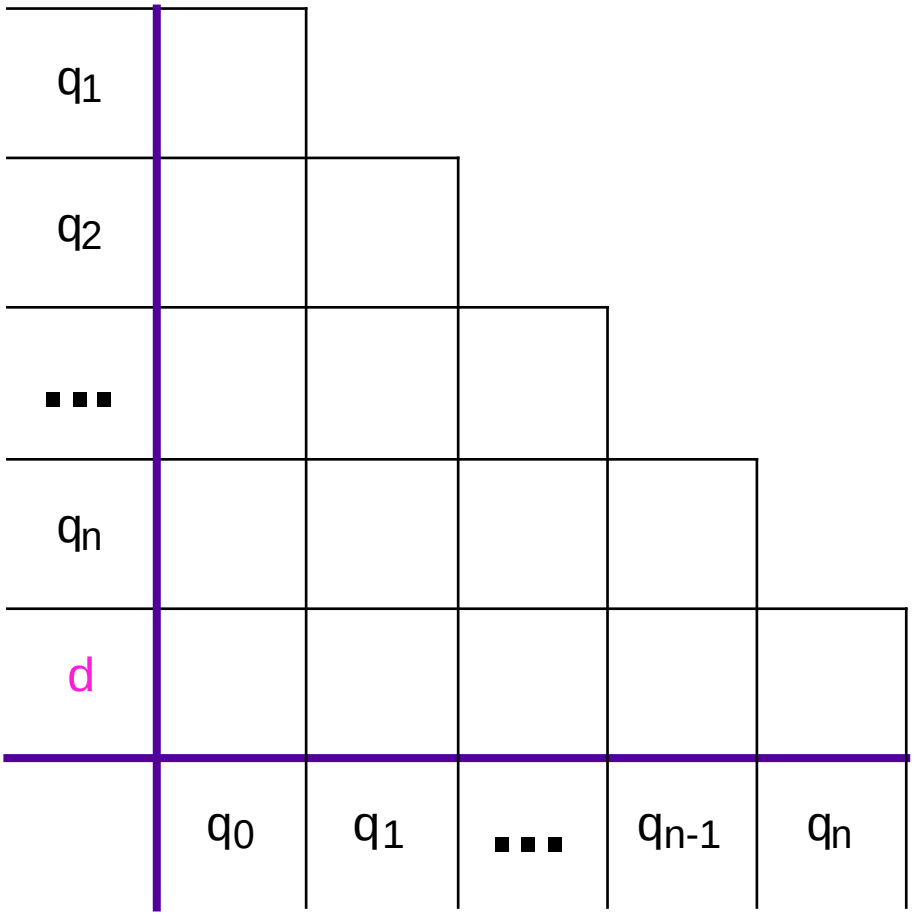
- gerar um AFD equivalente
  - algoritmos introduzidos nos teoremas
- eliminar os estados inacessíveis e suas correspondentes transições
  - algoritmo relativamente simples
  - sugerido como exercício
- função programa total
  - introduzir um novo estado não-final `d`
  - incluir as transições não-previstas, tendo como resultado o estado `d`
  - incluir um ciclo em `d` para todos os símbolos do alfabeto

♦ Idéia básica do algoritmo

- identifica os estados equivalentes
  - por *exclusão*
- a partir de uma tabela de estados
  - são marcados os estados não-equivalentes
- ao final do algoritmo
  - as referências não-marcadas
  - representam os estados equivalentes.

♦ *Algoritmo de Minimização*

- seja $M = (\Sigma, Q, \delta, q_0, F)$ um AFD
  - * satisfaz aos pré-requisitos
- tabela: relaciona os estados distintos



- marcar na tabela os pares
  - * {estado final, estado não-final}
  - * estados finais não são equivalentes a não-finais

♦ Para $\{q_u, q_v\}$ não-marcado e a $\in \sum$

- suponha $\delta(q_u, a) = p_u$ e $\delta(q_v, a) = p_v$
- se $p_u = p_v$
  - q_u é equivalente a q_v
  - para o símbolo a
  - *não marcar*
- se $p_u \neq p_v$ e o par $\{p_u, p_v\}$ é não-marcado
  - $\{q_u, q_v\}$ é incluído em uma lista
  - a partir de $\{p_u, p_v\}$
  - para posterior análise
- se $p_u \neq p_v$ e o par $\{p_u, p_v\}$ é marcado
  - $\{q_u, q_v\}$ é não-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

♦ Pares não-marcados são equivalentes: *unificar*

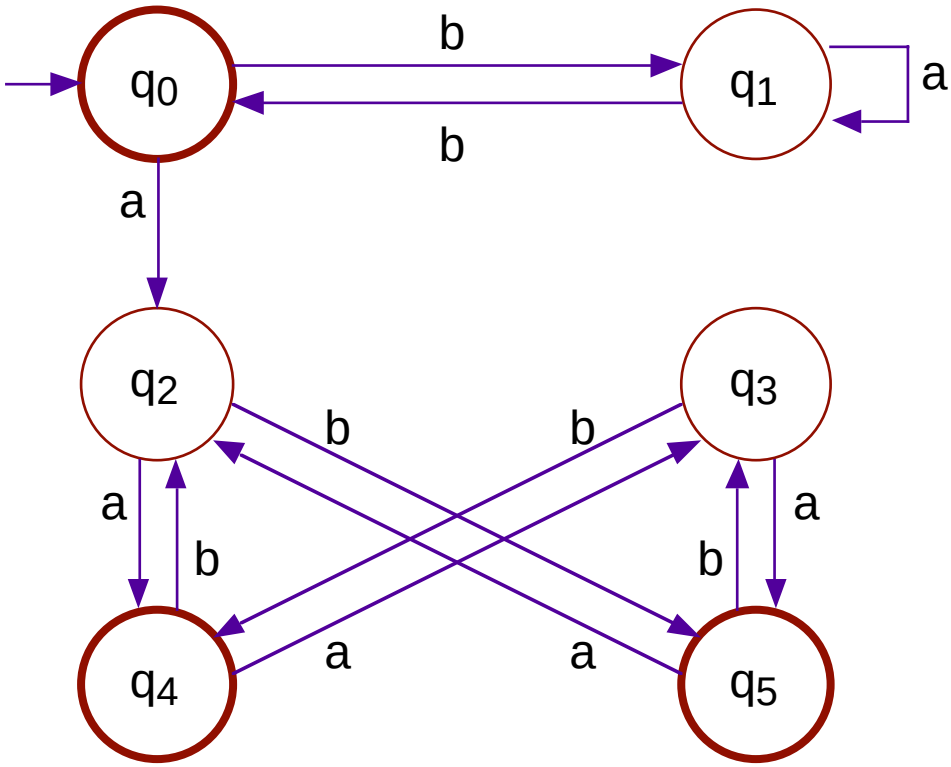
- a equivalência de estados é transitiva
- pares de estados equivalentes
  - unificados como um único estado
- se algum dos estados equivalentes é inicial
  - estado unificado é inicial

♦ Estados inúteis devem ser excluídos

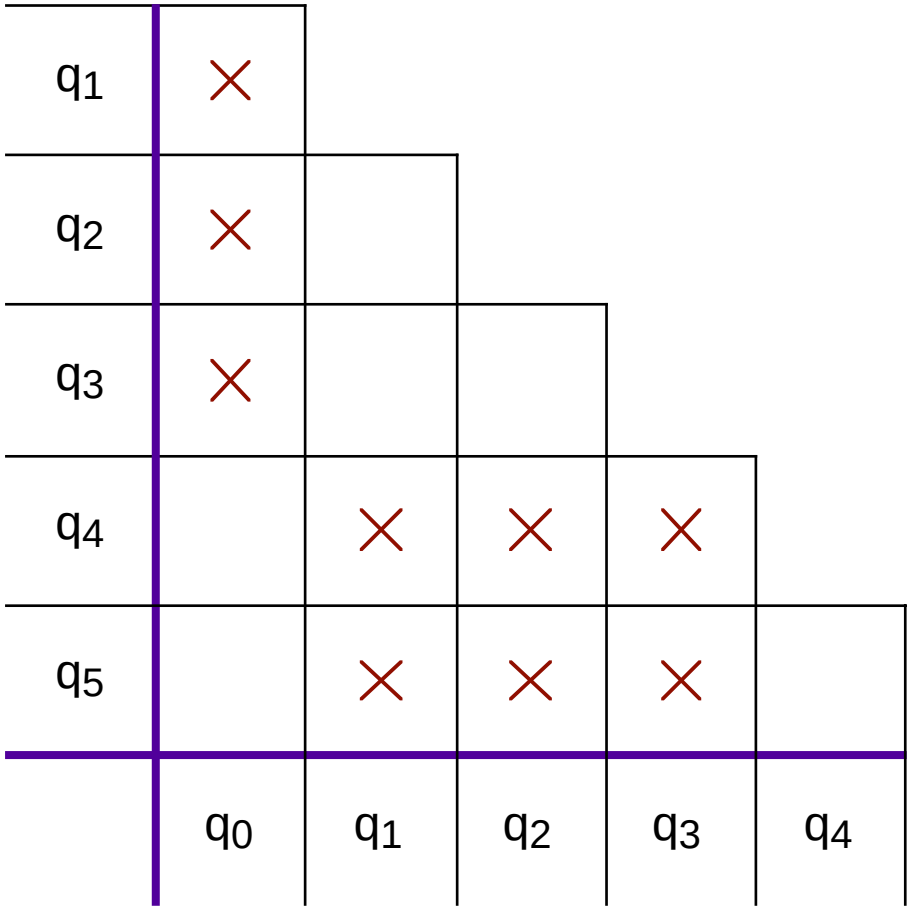
- um estado `q` é inútil
  - se é não-final
  - e a partir de `q` não é possível atingir um estado final
- o estado `d`
  - se incluído
  - sempre é inútil
- algoritmo para excluir os estados inúteis
  - e suas correspondentes transições
  - é simples
  - exercício

♦ *Exemplo*: Considere o AFD

- satisfaz os pré-requisitos de minimização



- contrução da tabela
- marcação dos pares
  - {estado final, estado não-final}



- análise dos pares de estado não-marcados
- **{q_0, q_4}**

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

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

  $\{q_1, q_2\}$ e $\{q_2, q_3\}$ são não-marcados:

  $\{q_0, q_4\}$ é *incluído* nas listas de $\{q_1, q_2\}$ e $\{q_2, q_3\}$

- **{q_0, q_5}**

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

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

  $\{q_1, q_3\}$ é não-marcado (e como $\{q_2, q_2\}$ é trivialmente equivalente):

  $\{q_0, q_5\}$ é *incluído* na lista de $\{q_1, q_3\}$

- **{q_1, q_2}**

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

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

  $\{q_1, q_4\}$ é marcado: $\{q_1, q_2\}$ é *marcado*

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

- **{q_1, q_3}**

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

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

  $\{q_1, q_5\}$ e $\{q_0, q_4\}$ são marcados: $\{q_1, q_3\}$ é *marcado*

  $\{q_1, q_3\}$ encabeça uma lista: $\{q_0, q_5\}$ é *marcado*

- **{q_2, q_3}**

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

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

  $\{q_4, q_5\}$ é não-marcado:

  $\{q_2, q_3\}$ é *incluído* na lista de $\{q_4, q_5\}$

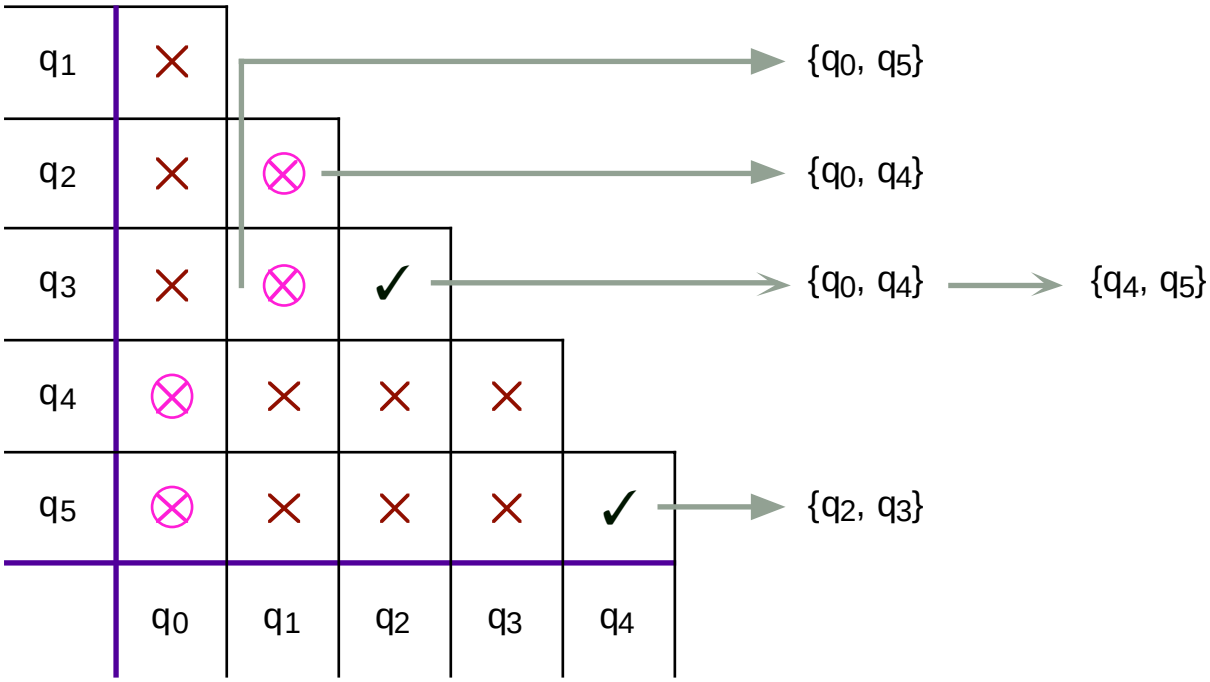
- **{q_4, q_5}**

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

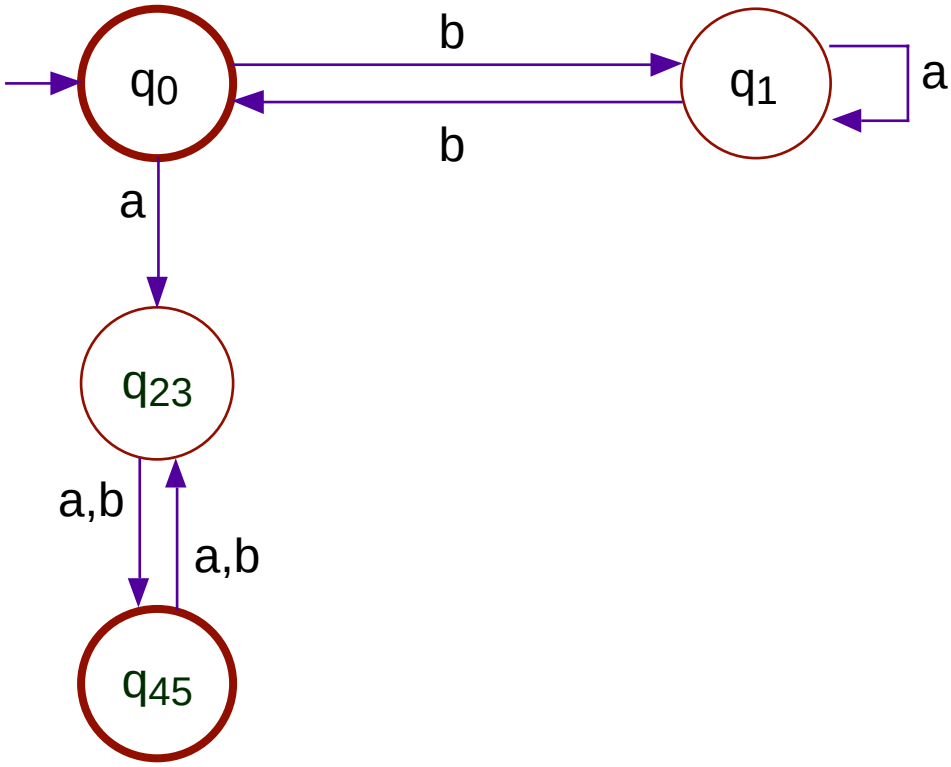
  $\delta(q_4, b) = q_2$ $\quad$ $\delta(q_5, b) = q_3$

  Como $\{q_2, q_3\}$ é não-marcado:

  $\{q_4, q_5\}$ é *incluído* na lista de $\{q_2, q_3\}$



- Como os pares {q2, q3} e {q4, q5} são não-marcados
  - * q23: unificação dos estados não-finais q2 e q3;
  - * q45: unificação dos estados finais q4 e q5.



- ***Teorema*:**
  - O AFD construído
  - usando o algoritmo de minimização
  - é o autômato com menor número de estados para a linguagem

- ***Teorema*:**
  - O *AFD mínimo* de uma linguagem é *único*
  - a menos de ismomorfismo

2.9 AF com Saída

♦ O conceito básico de AF

- aplicações restritas
- saída
  - limitada à lógica binária aceita/rejeita

♦ Geração de uma palavra de saída

- estende a definição de AF
- *não* altera a capacidade de reconhecimento
  - reconhece a classe de liguagens regulares
- saída
  - *não* pode ser lida
  - *não* pode ser usada como memória auxiliar
- as saídas podem ser associadas
  - às ***transições*** - Máquina de **Mealy**
  - aos ***estados*** - Máquina de **Moore**

♦ Saída

- definida sobre um alfabeto especial
  - *alfabeto de saída*
  - pode ser igual ao alfabeto de entrada
- saída
  - fita independente da de entrada
- cabeça da fita de saída
  - move uma célula para direita
  - a cada símbolo gravado
- resultado do processamento
  - estado final (condição de aceita/rejeita)
  - informação contida na fita de saída

♦ Máquinas de Mealy e Moore

- modificações sobre o AFD
- exercício
  - não-determinismo
  - movimentos vazio

Máquina de Mealy

- para cada transição
  - gera uma palavra de saída
  - pode ser vazia

♦ *Definição. Máquina de Mealy*

- 6-upla $M = (\Sigma, Q, \delta, q_0, F, \Delta)$
- Σ alfabeto de símbolos de entrada
- Q conjunto finito de estados
- δ função programa ou de transição
  - $\delta: Q \times \Sigma \rightarrow Q \times \Delta^*$
  - função parcial
- q_0 estado inicial tq $q_0 \in Q$
- F conjunto de estados finais tq $F \subseteq Q$
- Δ alfabeto de símbolos de saída

♦ Máquina de Mealy × AFD

- Σ, Q, F e q_0 são como no AFD

♦ Processamento para uma entrada w

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

♦ Palavra vazia como saída

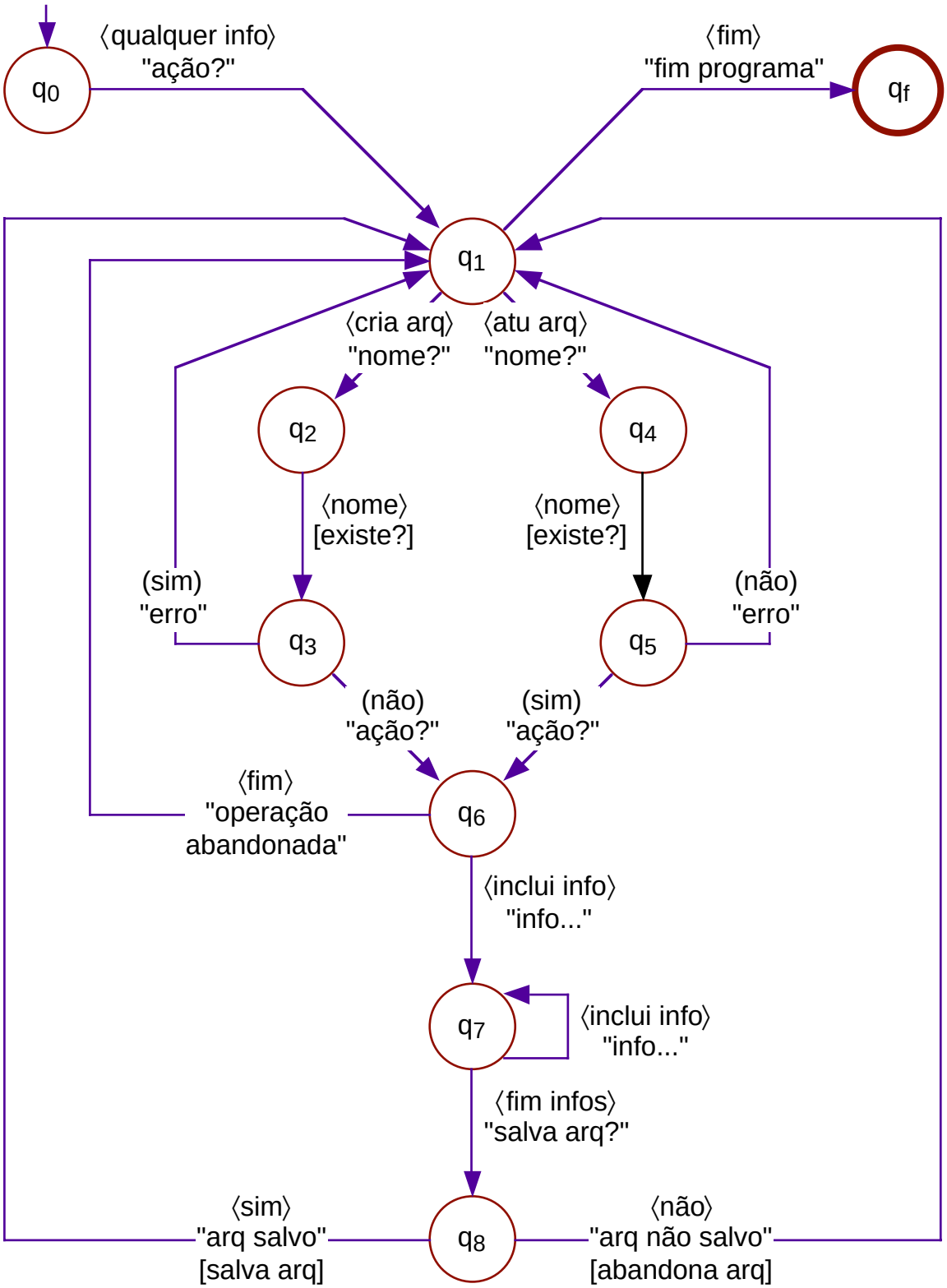
- nenhuma gravação é realizada
  - não move a cabeça da fita de saída
- se todas as transições geram saída vazia
  - Máq. de Mealy processa como um AFD

♦ Função programa estendida

- definição formal para uma Máquina de Mealy
  - exercício

♦ *Exemplo*: programa × usuário

- cria e atualiza arquivos
- ⟨...⟩ entrada fornecida pelo usuário
  - exemplo: teclado
- "..." saída gerada pelo programa
  - exemplo: vídeo
- **[...]** ação interna ao programa
  - sem comunicação com o usuário
- **(...)** resultado de uma ação interna ao programa
  - entrada no diagrama
- $M = (\Sigma, \{q_0, q_1, ..., q_8, q_f\}, \delta, q_0, \{q_f\}, \Delta)$
  - $\Sigma = \Delta$
  - símbolos válidos no diálogo


q0
⟨qualquer info⟩
"ação?"
⟨fim⟩
"fim programa"
qf
q1
⟨cria arq⟩
"nome?"
⟨atu arq⟩
"nome?"
q2
q4
⟨nome⟩
[existe?]
⟨nome⟩
[existe?]
(sim)
"erro"
(não)
"erro"
q3
q5
(não)
"ação?"
(sim)
"ação?"
⟨fim⟩
"operação
abandonada"
q6
⟨inclui info⟩
"info..."
q7
⟨inclui info⟩
"info..."
⟨fim infos⟩
"salva arq?"
⟨sim⟩
"arq salvo"
[salva arq]
q8
⟨não⟩
"arq não salvo"
[abandona arq]

Máquina de Moore

- para cada estado da máquina
  - gera uma palavra de saída
  - pode ser vazia
- possui uma função de saída

♦ *Máquina de Moore*

- 7-upla $M = (\Sigma, Q, \delta, q_0, F, \Delta, \delta_S)$
- Σ alfabeto de símbolos de entrada
- δ função programa ou de transição
  - $\delta: Q \times \Sigma \rightarrow Q$
  - função parcial
- Q conjunto finito de estados
- q_0 estado inicial tq $q_0 \in Q$
- F conjunto de estados finais tq $F \subseteq Q$
- Δ alfabeto de símbolos de saída
- δ_S função de saída
  - $\delta_S: Q \rightarrow \Delta^*$
  - *é* total

♦ Moore × AFD × Mealy

- Σ, Q, δ, F e q_0 são como no AFD
- Δ é como na Máquina de Mealy

♦ Processamento para uma entrada w

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

♦ Palavra vazia como saída

- nenhuma gravação é realizada
  - não move a cabeça da fita de saída
- se todos os estados geram saída vazia
  - Máquina de Moore processa como um AFD

♦ Função programa estendida

- definição formal para uma Máquina de Moore
  - exercício

♦ *Exemplo*: Analisadores Léxicos

- p/ compiladores ou tradutores de linguagens
- analisador léxico
  - basicamente é um AF
  - em geral, determinístico
- identifica os componentes básicos da linguagem
  - números
  - identificadores
  - separadores
  - etc
- *estado final*
  - associado a cada unidade léxica identificada
  - *saída*: descreve ou codifica a unidade léxica
- *estado não-final*
  - *saída*: palavra vazia
- como seria uma correspondente Máquina de Mealy?

Equivalência Moore × Mealy

♦ Equivalência

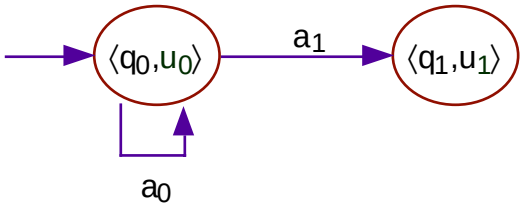
- *não* é válida para a entrada vazia
- Máquina de Moore
  - gera a palavra correspondente ao estado inicial
- Máquina de Mealy
  - não gera saída
  - não executa transição alguma
- demais casos
  - *equivalência*

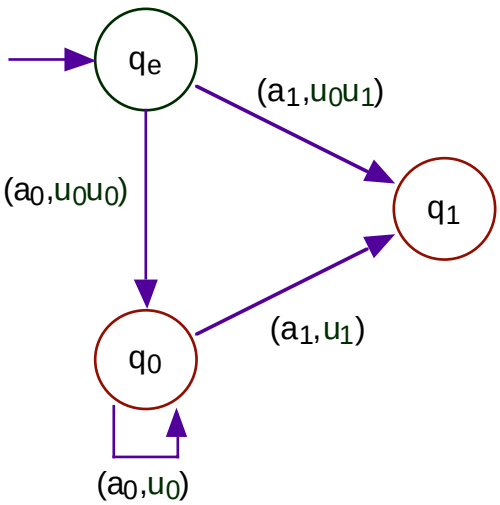
♦ *Teorema*: Moore → Mealy

- qq Máq de Moore
- pode ser simulada por uma Máq de Mealy
- para *entradas não-vazias*

♦ Correspondente Máq de Mealy

- simulação da saída do estado inicial de Moore?
  - ∗ introduz um novo estado q_e
  - ∗ referenciado somente na primeira transição a ser executada
  - ∗ saída referente ao estado inicial de Moore





♦ Prova

- MO = $(\Sigma, Q, \delta_{MO}, q_0, F, \Delta, \delta_S)$, Máq de Moore qq
- seja a Máquina de Mealy

  $ME = (\Sigma, Q \cup \{q_e\}, \delta_{ME}, q_e, F, \Delta)$

  (i) $\delta_{ME}(q_e, a) =$

  $(\delta_{MO}(q_0, a), \delta_S(q_0)\delta_S(\delta_{MO}(q_0, a)))$

  (ii) $\delta_{ME}(q, a) = (\delta_{MO}(q, a), \delta_S(\delta_{MO}(q, a)))$

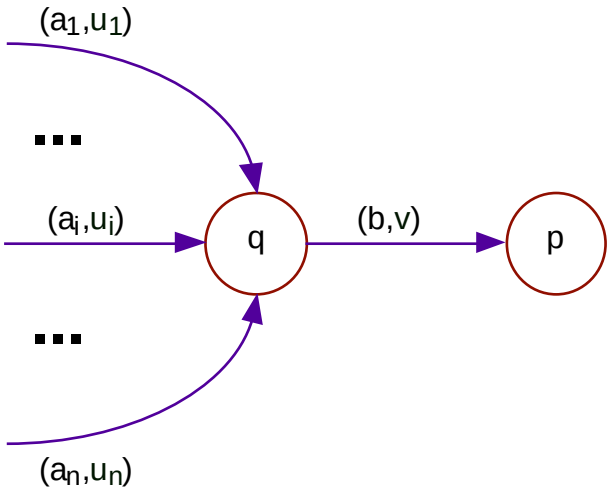
- ou seja, δ_{ME} construída a partir de
  * δ_{MO}
  * δ_S
- prova por indução que, de fato ME simula MO
  * em $n > 0$
  * ao reconhecer a entrada $a_1...a_n$
  * se M0 passa pelos estados $q_0, q_1, ..., q_n$
  * e gera as saídas $u_0, u_1, ..., u_n$
  * então ME passa pelos estados $q_e, q_0, q_1, ..., q_n$
  * e gera as saídas $u_0u_1, ..., u_n$
  * exercício

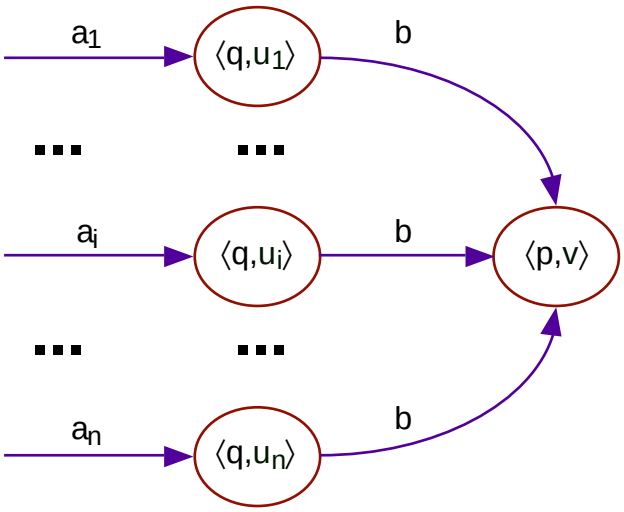
♦ *Teorema*. Mealy → Moore

- Qq Máquina de Mealy
- pode ser simulada por uma Máquina de Moore

♦ Correspondente Máq de Moore

- transições com saídas diferentes que atingem um mesmo estado
  - ∗ um estado para cada símbolo de saída
  - ∗ em geral, aumenta o número de estados





♦ Prova

- $ME = (\Sigma, Q, \delta_{ME}, q_0, F, \Delta)$, Máq de Mealy qq
- seja $S(\delta_{ME})$
  - imagem da função programa δ_{ME}
  - restrita à componente da palavra de saída
  - ou seja, o conjunto de todas as saídas possíveis
- seja a Máq de Moore

$$MO = (\Sigma, Q \times S(\delta_{ME}), \delta_{MO}, \langle q_0, \varepsilon \rangle, F \times S(\delta_{ME}), \Delta, \delta_S)$$

- para $\delta_{ME}(q_0, a) = (q, u)$, tem-se que

$$\delta_{MO}(\langle q_0, \varepsilon \rangle, a) = \langle q, u \rangle$$

- se $\delta_{ME}(q, b) = (p, v)$

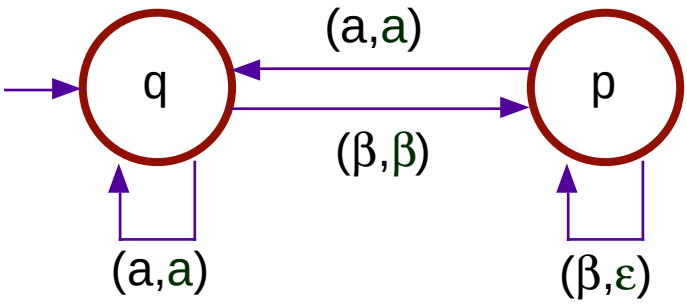
  então, para a_i e q_i tq $\delta_{ME}(q_i, a_i) = (q, u_i)$, tem-se que

$$\delta_{MO}(\langle q, u_i \rangle, b) = \langle p, v \rangle$$

- a função de saída δ_S é tq $\delta_S(\langle q, u \rangle) = u$
- prova por indução que, de fato MO simula ME
  - exercício

♦ *Exemplo*: Compactação de brancos

- compacta os brancos de um texto
- Máq. Mealy
  - ME = ({a, β}, {q, p}, δ_{ME}, q, {q, p}, {a, β})
  - a representa um símbolo qualquer
  - β o símbolo branco
  - δ_{ME} - na etiqueta de uma transição
    primeira componente: símbolo lido
    segunda componente: palavra gravada



- Máquina de Moore
  - $MO = (\{a, \beta\}, Q, \delta_{MO}, \langle q, \varepsilon\rangle, F, \{a, \beta\}, \delta_S)$
  - $Q = F = \{q, p\} \times \{\varepsilon, a, \beta\}$
  - δ_{MO} e δ_S onde, em cada estado, a segunda componente representa a saída