Capítulo 2: Máquinas de Estados Finitos

Newton José Vieira

Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais

19 de abril de 2018

Sumário

Sumário

Sumário

Sumário

Sumário

Sumário

Sumário

Sumário

1. Alguns Exemplos
   - Um quebra-cabeça
   - Um probleminha de matemática
   - Modelagem do funcionamento de um elevador

Quebra-cabeça

Leão, coelho e repolho

Um homem, um leão, um coelho e um repolho devem atravessar um rio usando uma canoa, com a restrição de que o homem deve transportar no máximo um dos três de cada vez de uma margem à outra. Além disso, o leão não pode ficar na mesma margem que o coelho sem a presença do homem, e o coelho não pode ficar com o repolho sem a presença do homem. O problema consiste em determinar se é possível fazer a travessia e, caso seja, a sequência de movimentações que propicie a travessia.

Modelagem de um problema

Na fase de modelagem:

a) as informações *relevantes* são identificadas (abstração)

b) as informações relevantes são estruturadas (representadas), de forma a facilitar a posterior solução

$\Longrightarrow$ Menos informações $\rightarrow$ solução mais fácil e/ou eficiente

Modelagem do quebra-cabeça

Informações relevantes abstraídas:

a) em que margem estão o homem, o leão, o coelho e o repolho

b) a sequência de movimentações entre as margens que propiciou a situação indicada em (a)

Modelagem do quebra-cabeça

Informações relevantes abstraídas:

a) em que margem estão o homem, o leão, o coelho e o repolho

b) a sequência de movimentações entre as margens que propiciou a situação indicada em (a)

Representação das informações:

a) $O/D \in \{\mathsf{h}, \mathsf{l}, \mathsf{c}, \mathsf{r}\}^2$ representa que os elementos em O estão na margem origem e os em D na margem destino

b) palavra $a_0 a_1 a_2 \ldots a_n$, em que cada a_i pode ser `s`, `l`, `c` ou `r`, representa a sequência de movimentos até o momento

Modelagem por meio de estados e transições

- Estado: uma fotografia da realidade

Modelagem por meio de estados e transições

- Estado: uma fotografia da realidade
- Transição de um estado para outro: provocada por uma ação

Modelagem por meio de estados e transições

- Estado: uma fotografia da realidade
- Transição de um estado para outro: provocada por uma ação
- Solução: sequência de ações que levam de um estado **inicial** a um estado **final**

Modelagem por meio de estados e transições

- Estado: uma fotografia da realidade
- Transição de um estado para outro: provocada por uma ação
- Solução: sequência de ações que levam de um estado **inicial** a um estado **final**

Para o quebra-cabeça:

- Estado: um par de subconjuntos de $\{h, l, c, r\}$

  inicial: $\{h, l, c, r\}/\{\}$
  final: $\{\}/\{h, l, c, r\}$

Modelagem por meio de estados e transições

- Estado: uma fotografia da realidade
- Transição de um estado para outro: provocada por uma ação
- Solução: sequência de ações que levam de um estado **inicial** a um estado **final**

Para o quebra-cabeça:

- Estado: um par de subconjuntos de $\{h, l, c, r\}$

  inicial: $\{h, l, c, r\}/\{\}$
  final: $\{\}/\{h, l, c, r\}$

- Transição: provocada por uma das ações: `s`, `l`, `c` ou `r`

Modelagem por meio de estados e transições

- Estado: uma fotografia da realidade
- Transição de um estado para outro: provocada por uma ação
- Solução: sequência de ações que levam de um estado **inicial** a um estado **final**

Para o quebra-cabeça:

- Estado: um par de subconjuntos de $\{h, l, c, r\}$

  inicial: $\{h, l, c, r\}/\{\}$
  final: $\{\}/\{h, l, c, r\}$
- Transição: provocada por uma das ações: `s`, `l`, `c` ou `r`
- Solução: uma palavra no alfabeto $\{\texttt{s}, \texttt{l}, \texttt{c}, \texttt{r}\}$

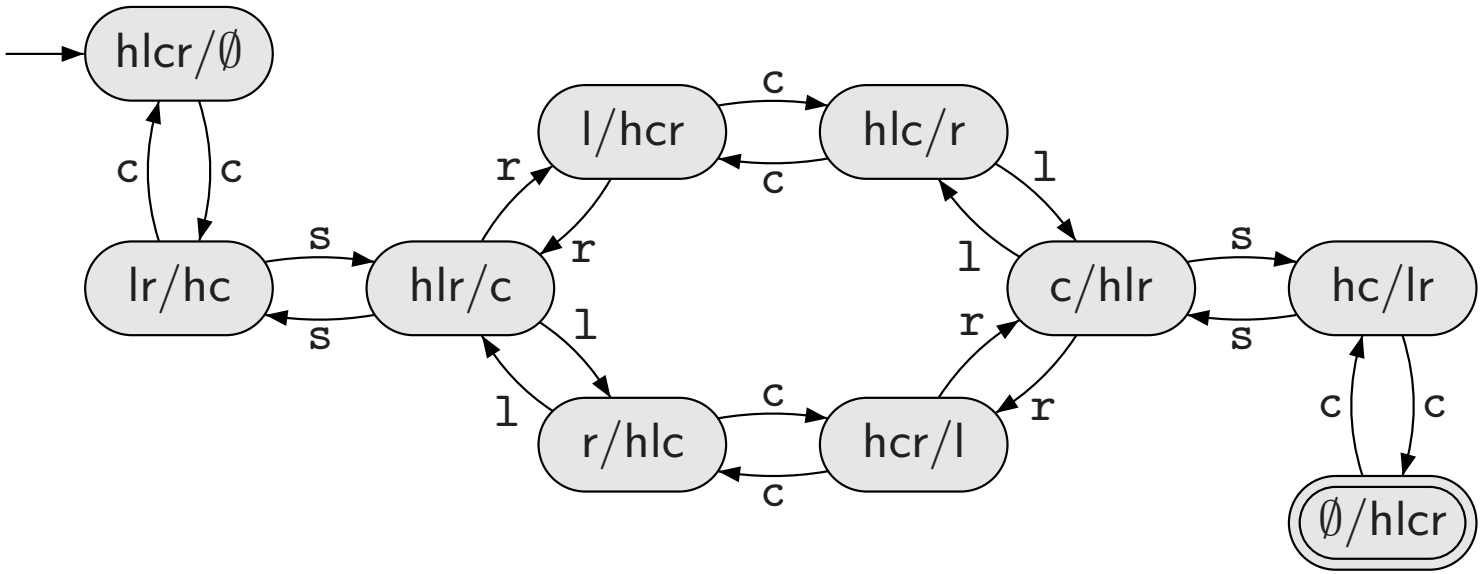
Diagrama de estados

Representação dos estados e transições por meio de um grafo:

- Estado: vértice do grafo
  - inicial: ressaltado por uma seta que o aponta;
  - final: oval dupla.
- Transição de e para e' provocada por $\mathtt{s}$: aresta de e para e' com rótulo $\mathtt{s}$.

Diagrama de estados para o quebra-cabeça

▸ Formalização



Reconhecimento e rejeição

Dada $w \in \{\mathtt{s}, \mathtt{l}, \mathtt{c}, \mathtt{r}\}^*$, começando no estado inicial:

- w é reconhecida se o "caminho correspondente" termina no estado final
- w é rejeitada, caso contrário

Configuração instantânea

Configuração instantânea durante processamento de w: $[e, y]$, sendo:

- e: o estado atual após processar um prefixo x de w
- y: o sufixo ainda não processado (assim, $w = xy$)

Computação

Definição

$[e_1, w] \vdash [e_2, y]$ *sse existe uma transição de* e_1 *para* e_2 *sob a e* $w = ay$.

Computação

Definição

$[e_1, w] \vdash [e_2, y]$ *sse existe uma transição de* e_1 *para* e_2 *sob a e* $w = ay$.

Exemplo

$[lr/hc, \texttt{sllr}] \vdash [hlr/c, \texttt{llr}] \vdash [r/hlc, \texttt{lr}] \vdash [hlr/c, \texttt{r}] \vdash [l/hcr, \lambda]$

Isso é uma computação iniciada no estado lr/hc, *processando* `sllr`.

Características do autômato do quebra-cabeça

- para cada transição existe uma transição inversa
- há um único estado final
- determinismo: para cada par (estado, símbolo) existe, no máximo, uma transição;
- o conjunto de estados é finito

$\Longrightarrow$ Única característica geral: a última.

1. Alguns Exemplos
   - Um quebra-cabeça
   - Um probleminha de matemática
   - Modelagem do funcionamento de um elevador

Probleminha de matemática

Binário divisível por 6

Projetar uma máquina que, dada uma sequência de 0s e 1s, determine se o número representado por ela na base 2 é divisível por 6. O que se deseja é um projeto independente de implementação, ou seja, que capture apenas a essência de tal máquina, não importando se ela será mecânica, eletrônica, um programa ou o que quer que seja.

Alguns fatos aritméticos

Note, inicialmente, que:

- $\eta(x0) = 2\eta(x)$
- $\eta(x1) = 2\eta(x) + 1$

Alguns fatos aritméticos

Note, inicialmente, que:

- $\eta(x0) = 2\eta(x)$
- $\eta(x1) = 2\eta(x) + 1$

Se $\eta(x)$ mod $6 = r$, então o próximo resto é:

- $\eta(x0)$ mod $6 = 2r$ mod 6 ou
- $\eta(x1)$ mod $6 = (2r + 1)$ mod 6

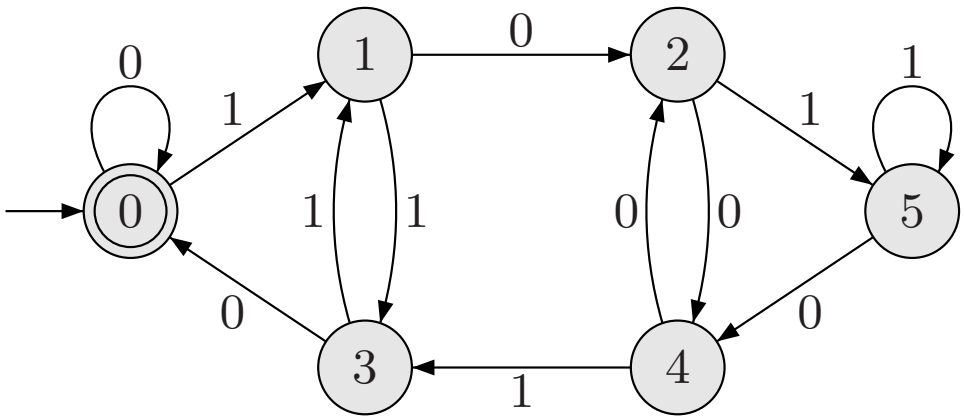
O autômato

As características do autômato:

- Um estado para cada resto de 0 a 5
- Do estado r emanam duas transições:
  - transição sob 0 para o estado $2r \bmod 6$ e
  - transição sob 1 para o estado $(2r + 1) \bmod 6$
- Estado inicial: 0
- Estado final: 0

Diagrama de estados para binário módulo 6

▸ Formalização



1. Alguns Exemplos
   - Um quebra-cabeça
   - Um probleminha de matemática
   - Modelagem do funcionamento de um elevador

Elevador para 3 andares

Modelagem de elevador

Modelar um elevador destinado a servir a um prédio de três andares, satisfazendo:

1. caso não haja chamada, o elevador fica parado onde estiver;
2. o elevador dá prioridade ao chamado mais próximo no sentido em que estiver se movimentando;
3. um chamado pode ser "desligado" manualmente. Assim, por exemplo, é possível existir uma chamada para um andar em certo instante e, logo em seguida, não existir mais, sem que o elevador se mova.

Modelando o elevador

Informações relevantes:

- o andar em que o elevador se encontra (estado)
- o sentido em que o elevador está se movendo (estado)
- os andares em que o elevador está sendo solicitado (provoca transição)
- saída em transição: subir, descer, ficar imóvel.

Modelando o elevador

Informações relevantes:

- o andar em que o elevador se encontra (estado)
- o sentido em que o elevador está se movendo (estado)
- os andares em que o elevador está sendo solicitado (provoca transição)
- saída em transição: subir, descer, ficar imóvel.

Opções quanto ao funcionamento do elevador:

- se o elevador está sendo chamado para o andar em que já está, ele fica parado;
- o elevador dá preferência para o sentido em que está se movendo.

Componentes do autômato

Detalhes:

- Estados: 1, 2 $\uparrow$, 2 $\downarrow$ e 3;
- Não há estado inicial ou final;
- Alfabeto de entrada: $\mathcal{P}(\{1, 2, 3\})$;
- Alfabeto de saída: $\{\uparrow, \downarrow, \circ\}$;

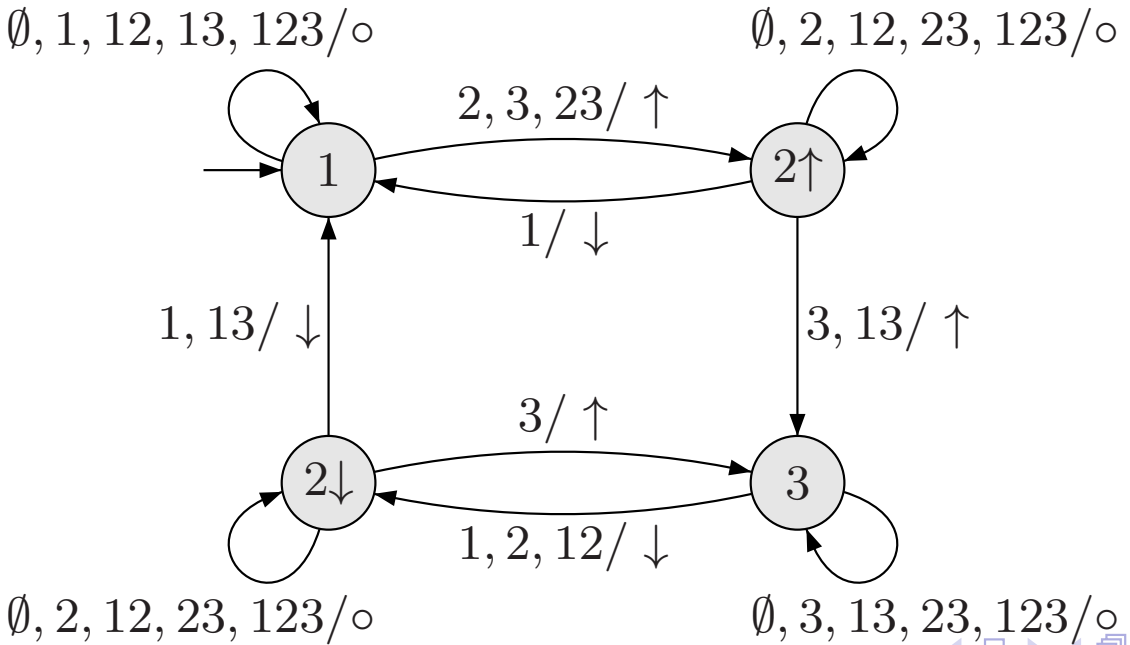
Componentes do autômato

Detalhes:

- Estados: 1, 2 $\uparrow$, 2 $\downarrow$ e 3;
- Não há estado inicial ou final;
- Alfabeto de entrada: $\mathcal{P}(\{1, 2, 3\})$;
- Alfabeto de saída: $\{\uparrow, \downarrow, \circ\}$;

$\Longrightarrow$ O autômato é uma máquina de Mealy.

Diagrama de estados para um elevador



Computação da máquina de Mealy

Configuração instantânea: $[e, x, y]$, em que:

- e: estado atual
- x: sufixo a ser processado
- y: sequência das saídas emitidas até o momento

Computação da máquina de Mealy

Configuração instantânea: $[e, x, y]$, em que:

- e: estado atual
- x: sufixo a ser processado
- y: sequência das saídas emitidas até o momento

Exemplo

Uma computação:

$$[1, 23131, \lambda] \vdash [2\uparrow, 131, \uparrow] \vdash [3, 1, \uparrow\uparrow] \vdash [2\downarrow, \lambda, \uparrow\uparrow\downarrow]$$

2. Autômatos Finitos Determinísticos
   - O que é autômato finito determinístico
   - Minimização de AFDs
   - Algumas propriedades dos AFDs

O que é autômato finito determinístico

Definição

Um autômato finito determinístico (AFD) é uma quíntupla $(E, \Sigma, \delta, i, F)$ *em que:*

- E *é um conjunto finito não vazio de estados*
- Σ *é um alfabeto*
- $\delta : E \times \Sigma \to E$ *é a função de transição, uma função total*
- $i \in E$ *é o estado inicial*
- $F \subseteq E$ *é o conjunto de estados finais*

Propriedades dos AFDs

- Determinismo: a partir do estado inicial é atingido um único estado, para uma dada palavra de entrada.
- Função de transição total: para toda palavra de entrada, atinge-se um estado consumindo-se toda a palavra.
- Um único estado inicial: com vários o poder computacional não é maior.
- Vários estados finais: com um só, o poder computacional é menor.
- Conjunto finito de estados: com conjunto infinito, o poder computacional é maior.

AFD para o quebra-cabeças

$$(E, \{\mathtt{s}, \mathtt{l}, \mathtt{c}, \mathtt{r}\}, \delta, \mathrm{hlcr}/\emptyset, \{\emptyset/\mathrm{hlcr}\})$$

$$E = \{\mathrm{hlcr}/\emptyset, \mathrm{lr}/\mathrm{hc}, \mathrm{hlr}/\mathrm{c}, \mathrm{l}/\mathrm{hcr}, \mathrm{r}/\mathrm{hlc}, \mathrm{hlc}/\mathrm{r}, \mathrm{hcr}/\mathrm{l}, \mathrm{c}/\mathrm{hlr}, \mathrm{hc}/\mathrm{lr}, \emptyset/\mathrm{hlcr}, \mathrm{x}\}$$

δ	s	l	c	r
$\mathrm{hlcr}/\emptyset$	x	x	lr/hc	x
lr/hc	hlr/c	x	$\mathrm{hlcr}/\emptyset$	x
hlr/c	lr/hc	r/hlc	x	l/hcr
l/hcr	x	x	hlc/r	hlr/c
r/hlc	x	hlr/c	hcr/l	x
hlc/r	x	c/hlr	l/hcr	x
hcr/l	x	x	r/hlc	c/hlr
c/hlr	hc/lr	hlc/r	x	hcr/l
hc/lr	c/hlr	x	$\emptyset/\mathrm{hlcr}$	x
$\emptyset/\mathrm{hlcr}$	x	x	hc/lr	x
x	x	x	x	x

▸ Diagrama de Estados

Uma convenção em diagramas de estados

Se não há transição de e sob a, então existe e' tal que:

- existe uma transição de e para e' sob a;
- e' não é estado final;
- existe uma transição de e' para e' sob cada símbolo do alfabeto.

$\Longrightarrow$ e' é um estado de erro

Uma convenção em diagramas de estados

Se não há transição de e sob a, então existe e' tal que:

- existe uma transição de e para e' sob a;
- e' não é estado final;
- existe uma transição de e' para e' sob cada símbolo do alfabeto.

$\Longrightarrow$ e' é um estado de erro

Diagrama de estados simplificado: aquele em que foram omitidos todos os estados de erro porventura existentes.

AFD para o probleminha de matemática

$$(\{0,1,2,3,4,5\},\{0,1\},\delta,0,\{0\})$$

δ	0	1
0	0	1
1	2	3
2	4	5
3	0	1
4	2	3
5	4	5

$\Longrightarrow$ Este AFD não tem estado de erro

▸ Diagrama de Estados

Função de transição estendida

Definição

Seja um AFD $M = (E, \Sigma, \delta, i, F)$. *A função de transição estendida para* M, $\hat{\delta}$: $E \times \Sigma^* \to E$, *é definida recursivamente assim:*

1. $\hat{\delta}(e, \lambda) = e$
2. $\hat{\delta}(e, ay) = \hat{\delta}(\delta(e, a), y)$, *para todo* $a \in \Sigma$ *e* $y \in \Sigma^*$

Exemplo

$\hat{\delta}(0, 1010) = \ldots$ *(para o probleminha de matemática)*

▸ Diagrama de Estados

Linguagem reconhecida por um AFD

Definição

A linguagem reconhecida por um AFD $M = (E, \Sigma, \delta, i, F)$ *é:*

$$L(M) = \{w \in \Sigma^* \mid \hat{\delta}(i, w) \in F\}$$

Linguagem reconhecida por um AFD

Definição

A linguagem reconhecida por um AFD $M = (E, \Sigma, \delta, i, F)$ *é:*

$$L(M) = \{w \in \Sigma^* \mid \hat{\delta}(i, w) \in F\}$$

$w \in \Sigma^*$ *é uma palavra reconhecida por* M *se* $\hat{\delta}(i, w) \in F$.

A função de transição estendida em forma procedural

Algoritmo

Entrada: e *e* w *tais que* $e \in E$ *e* $w \in \Sigma^*$
Saída: $\hat{\delta}(e, w)$.

enquanto $w \neq \lambda$ **faça**
    sejam $a \in \Sigma$ *e* $y \in \Sigma^*$ *tais que* $w = ay$;
    $e \leftarrow \delta(e, a)$; $w \leftarrow y$
fimenquanto;
retorne e

A função de transição estendida em forma procedural

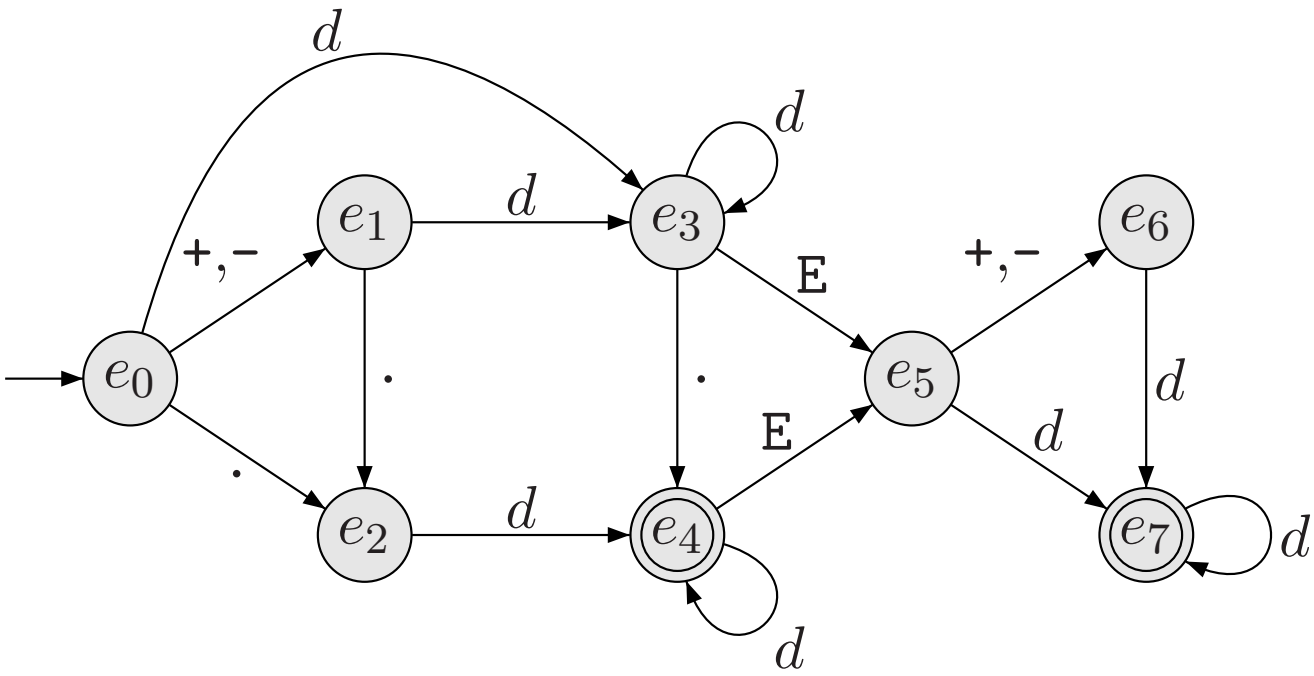
Algoritmo

Entrada: e *e* w *tais que* $e \in E$ *e* $w \in \Sigma^*$
Saída: $\hat{\delta}(e, w)$.

enquanto $w \neq \lambda$ **faça**
    sejam $a \in \Sigma$ *e* $y \in \Sigma^*$ *tais que* $w = ay$;
    $e \leftarrow \delta(e, a)$; $w \leftarrow y$
fimenquanto;
retorne e

Desempenho: $O(|w|)$.

Análise Léxica: diagrama de estados simplificado



Análise Léxica: constantes reais

$M = (E, \Sigma, \delta, e0, \{e3, e7\})$, em que:

- $E = \{e0, e1, e2, e3, e4, e5, e6, e7, x\}$ (x é um "estado de erro")
- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., +, -, \mathtt{E}\}$
- $F = \{e4, e7\}$
-

δ	d	.	+	-	E
$e0$	$e3$	$e2$	$e1$	$e1$	x
$e1$	$e3$	$e2$	x	x	x
$e2$	$e4$	x	x	x	x
$e3$	$e3$	$e4$	x	x	$e5$
$e4$	$e4$	x	x	x	$e5$
$e5$	$e7$	x	$e6$	$e6$	x
$e6$	$e7$	x	x	x	x
$e7$	$e7$	x	x	x	x
x	x	x	x	x	x

Mais exemplos

Exemplos

AFDs que reconheçam:

- $\{w \in \{0,1\}^* \mid w$ *tem número par de símbolos*$\}$.
- $\{w \in \{0,1\}^* \mid n_0(w)$ *é par e* $n_1(w)$ *é par*$\}$.

Mais exemplos

Exemplos

AFDs que reconheçam:

- $\{w \in \{0,1\}^* \mid w$ *tem número par de símbolos*$\}$.
- $\{w \in \{0,1\}^* \mid n_0(w)$ *é par e* $n_1(w)$ *é par*$\}$.

Definição

Dois AFDs, M_1 *e* M_2*, são ditos equivalentes sse* $L(M_1) = L(M_2)$.

Estados de erro

Definição

Um estado e de um AFD $(E, \Sigma, \delta, i, F)$ é um estado de erro sse não existe $w \in \Sigma^*$ tal que $\hat{\delta}(e, w) \in F$.

Estados de erro

Definição

Um estado e de um AFD $(E, \Sigma, \delta, i, F)$ é um estado de erro sse não existe $w \in \Sigma^*$ tal que $\hat{\delta}(e, w) \in F$.

Com base nessa definição e nas anteriores:

- Se e é um estado de erro, então $\hat{\delta}(e, w)$ é um estado de erro para toda $w \in \Sigma^*$.
- Se um estado de erro for atingido após a leitura de um prefixo, a palavra não é reconhecida.
- Nenhum estado final é estado de erro.
- i é um estado de erro em M sse $L(M) = \emptyset$.
- Se $X \subseteq E$ é um conjunto de estados de erro, obtém-se um AFD equivalente substituindo-se todos os estados de X por

Estados alcançáveis

Definição

Um estado e de um AFD $(E, \Sigma, \delta, i, F)$ é um estado alcançável sse existe $w \in \Sigma^*$ tal que $\hat{\delta}(i, w) = e$. Um estado não alcançável é dito inalcançável.

Com base nas definições anteriores:

- Se e é um estado alcançável, então $\hat{\delta}(e, w)$ é um estado alcançável para toda $w \in \Sigma^*$.
- Um estado alcançável em M pode ser estado de erro ou não.
- Um estado inalcançável em M também pode ser estado de erro ou não.
- $L(M) = \emptyset$ se e somente se $F = \emptyset$ ou todo estado final é inalcançável em M.

Duas perguntas

- Existe AFD mínimo?
- Há como construir um AFD, a partir de AFDs M_1 e M_2, que reconheça:
  - $L(M_1) \cup L(M_2)$?
  - $L(M_1) \cap L(M_2)$?
  - $L(M_1) - L(M_2)$?
  - $L(M_1)L(M_2)$?
  - $L(M_1)^*$?

$\Longrightarrow$ Resposta para ambas: SIM!

2. Autômatos Finitos Determinísticos
   - O que é autômato finito determinístico
   - Minimização de AFDs
   - Algumas propriedades dos AFDs

O que é AFD mínimo

Definição

Um AFD M é dito ser um AFD mínimo para a linguagem $L(M)$ se nenhum AFD para $L(M)$ contém menor número de estados que M.

O que é AFD mínimo

Definição

Um AFD M é dito ser um AFD mínimo para a linguagem $L(M)$ se nenhum AFD para $L(M)$ contém menor número de estados que M.

Para obter um AFD mínimo:

1. Eliminar estados não alcançáveis a partir do estado inicial.
2. Substituir cada grupo de estados equivalentes por um único estado.

Equivalência de estados

Definição

Seja um AFD $M = (E, \Sigma, \delta, i, F)$. *Então* $e, e' \in E$ *são ditos estados equivalentes*, $e \approx e'$, *sse:*

$$\text{para todo } y \in \Sigma^* \; \hat{\delta}(e, y) \in F \leftrightarrow \hat{\delta}(e', y) \in F.$$

Equivalência de estados

Definição

Seja um AFD $M = (E, \Sigma, \delta, i, F)$. *Então* $e, e' \in E$ *são ditos estados equivalentes*, $e \approx e'$, *sse:*

$$\text{para todo } y \in \Sigma^* \; \hat{\delta}(e, y) \in F \leftrightarrow \hat{\delta}(e', y) \in F.$$

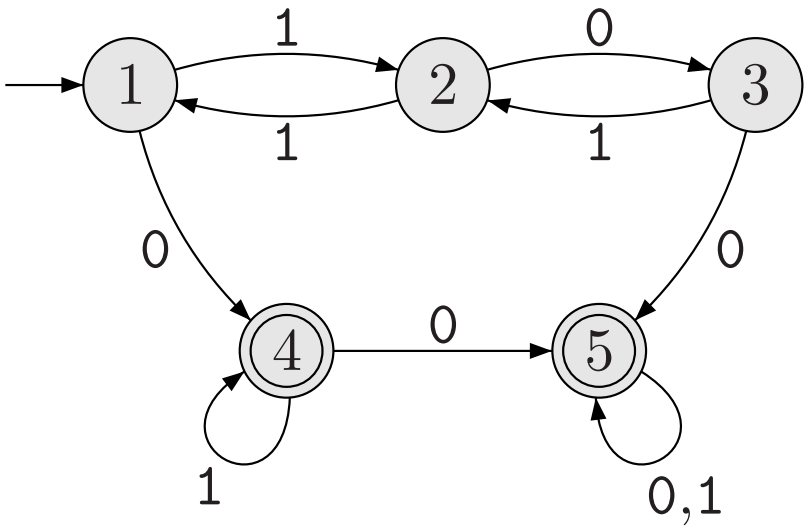
$\Longrightarrow \approx$ é relação de equivalência.

Por que reduzir estados equivalentes a um só?

Seja um AFD $M = (E, \Sigma, \delta, i, F)$.

1. se $e \approx e'$: um sufixo y é reconhecido passando-se por e sse ele é reconhecido passando-se por e'; logo e e e' podem se tornar um só!
2. se $e \not\approx e'$: se um sufixo y levar a estado de F, a palavra é aceita, caso contrário, não é. Logo, e e e' não podem se tornar um só!

Um AFD a minimizar

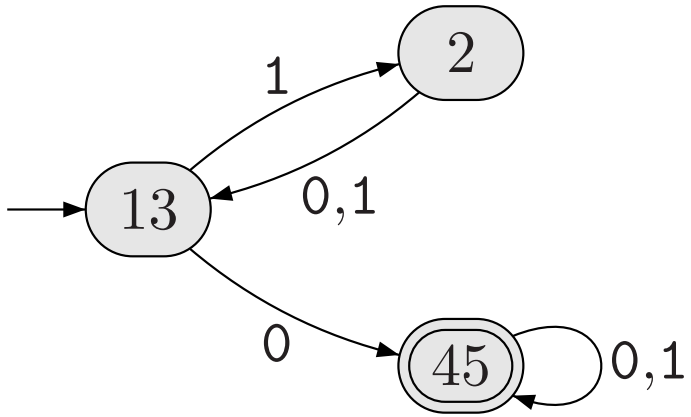


$4 \approx 5$

$1 \approx 3$

AFD mínimo equivalente

A partição induzida por $\approx$ é $\{\{1,3\},\{2\},\{4,5\}\}$.



Obtenção de $\approx$ passo a passo

Definição

Seja um AFD $M = (E, \Sigma, \delta, i, F)$ *e um estado* $e \in E$*. Definição de* $\approx_i$*:*

1. $e \approx_0 e'$ *sse* $(e \in F \leftrightarrow e' \in F)$
2. *para* $n \geq 0$*:* $e \approx_{n+1} e'$ *sse* $e \approx_n e'$ *e* $\delta(e, a) \approx_n \delta(e', a)$ *para todo* $a \in \Sigma$

Obtenção de $\approx$ passo a passo

Definição

Seja um AFD $M = (E, \Sigma, \delta, i, F)$ *e um estado* $e \in E$*. Definição de* $\approx_i$*:*

1. $e \approx_0 e'$ *sse* $(e \in F \leftrightarrow e' \in F)$
2. *para* $n \geq 0$*:* $e \approx_{n+1} e'$ *sse* $e \approx_n e'$ *e* $\delta(e, a) \approx_n \delta(e', a)$ *para todo* $a \in \Sigma$

- Como E é finito, existe k tal que $\approx_k = \approx_{k+1}$
- Se $\approx_k = \approx_{k+1}$, então $\approx_k = \approx$

Obtenção das partições $[e]_{\approx}$ passo a passo

$[e]_n$: classe de equivalência de e na partição induzida por $\approx_n$.

Obtenção de cada $[e]_n$:

1. $[e]_0 = \begin{cases} F & \text{se } e \in F \\ E - F & \text{se } e \in E - F; \end{cases}$
2. para $n \geq 0$, $[e]_{n+1} = \{e' \in [e]_n \mid [\delta(e', a)]_n = [\delta(e, a)]_n$ para todo $a \in \Sigma\}$.

Um exemplo de minimização

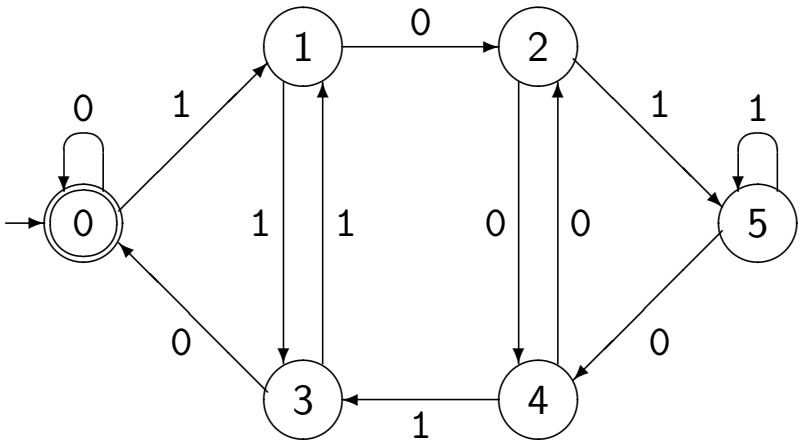
Evolução das partições

S_0: $\{1,2,3\},\{4,5\}$
S_1: $\{1,3\},\{2\},\{4,5\}$
S_2: $\{1,3\},\{2\},\{4,5\}$

▸ Diagrama de Estados

Outro exemplo de minimização

Autômato inicial



Evolução das partições

S_0: $\{0\}, \{1,2,3,4,5\}$

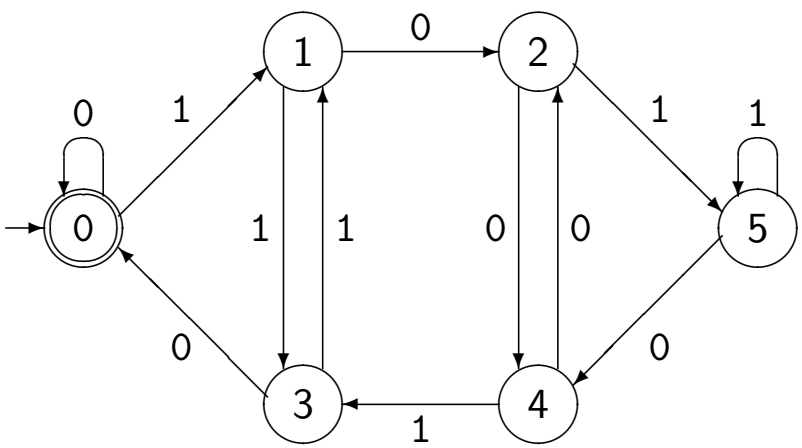
S_1: $\{0\}, \{1,2,4,5\}, \{3\}$

S_2: $\{0\}, \{1,4\}, \{2,5\}, \{3\}$

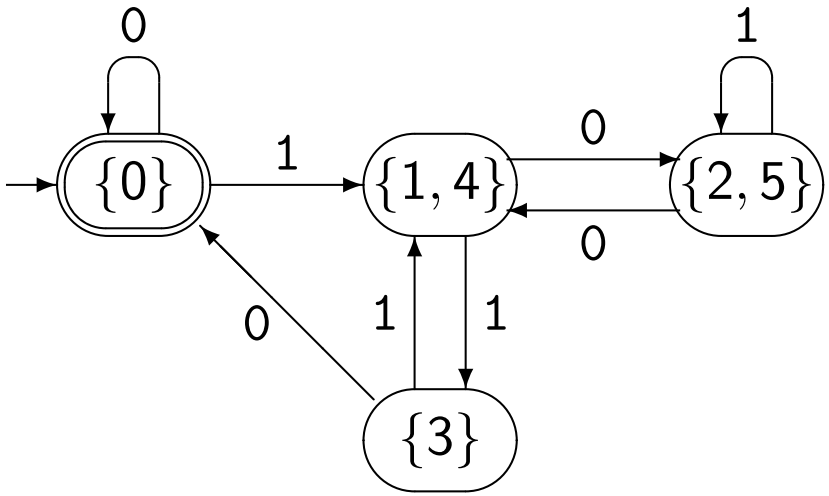
S_3: $\{0\}, \{1,4\}, \{2,5\}, \{3\}$

Outro exemplo de minimização

Autômato inicial



Autômato reduzido



2. Autômatos Finitos Determinísticos
   - O que é autômato finito determinístico
   - Minimização de AFDs
   - Algumas propriedades dos AFDs

Produto de autômatos

Sejam $M_1 = (E_1, \Sigma, \delta_1, i_1, F_1)$ e $M_2 = (E_2, \Sigma, \delta_2, i_2, F_2)$.

M_3 simula M_1 e M_2 em paralelo

$M_3 = (E_3, \Sigma, \delta_3, i_3, F_3)$, em que:

- $E_3 = E_1 \times E_2$;
- $\delta_3([e_1, e_2], a) = [\delta_1(e_1, a), \delta_2(e_2, a)] \; \forall \, e_1 \in E_1, \; e_2 \in E_2, \; a \in \Sigma$;
- $i_3 = [i_1, i_2]$.
- F_3: depende do que se quer para M_3.

Produto de autômatos

Sejam $M_1 = (E_1, \Sigma, \delta_1, i_1, F_1)$ e $M_2 = (E_2, \Sigma, \delta_2, i_2, F_2)$.

M_3 simula M_1 e M_2 em paralelo

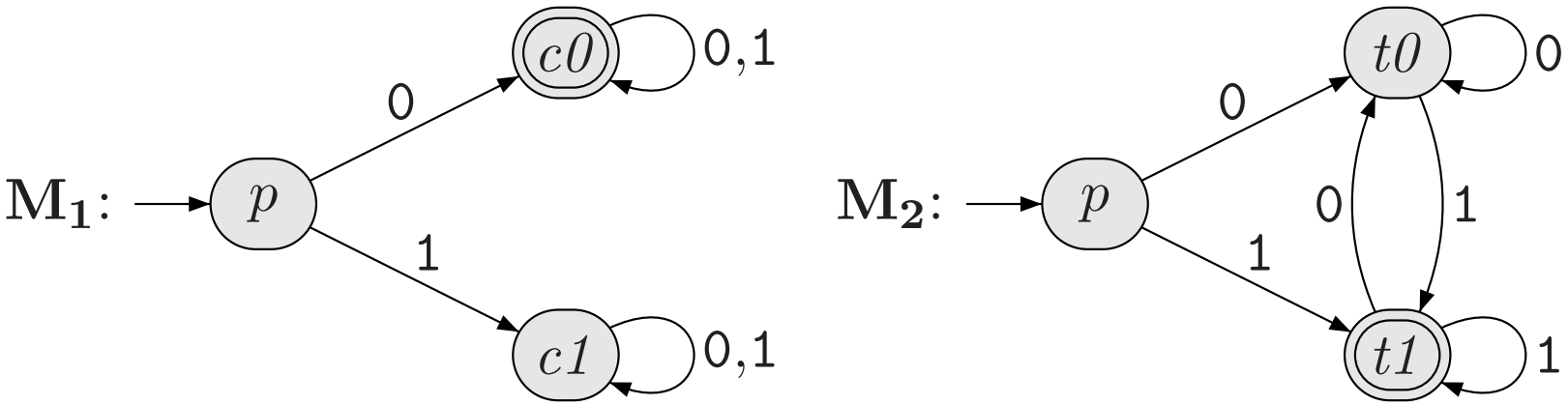
$M_3 = (E_3, \Sigma, \delta_3, i_3, F_3)$, em que:

- $E_3 = E_1 \times E_2$;
- $\delta_3([e_1, e_2], a) = [\delta_1(e_1, a), \delta_2(e_2, a)] \; \forall \, e_1 \in E_1, \; e_2 \in E_2, \; a \in \Sigma$;
- $i_3 = [i_1, i_2]$.
- F_3: depende do que se quer para M_3.

Processando uma palavra:

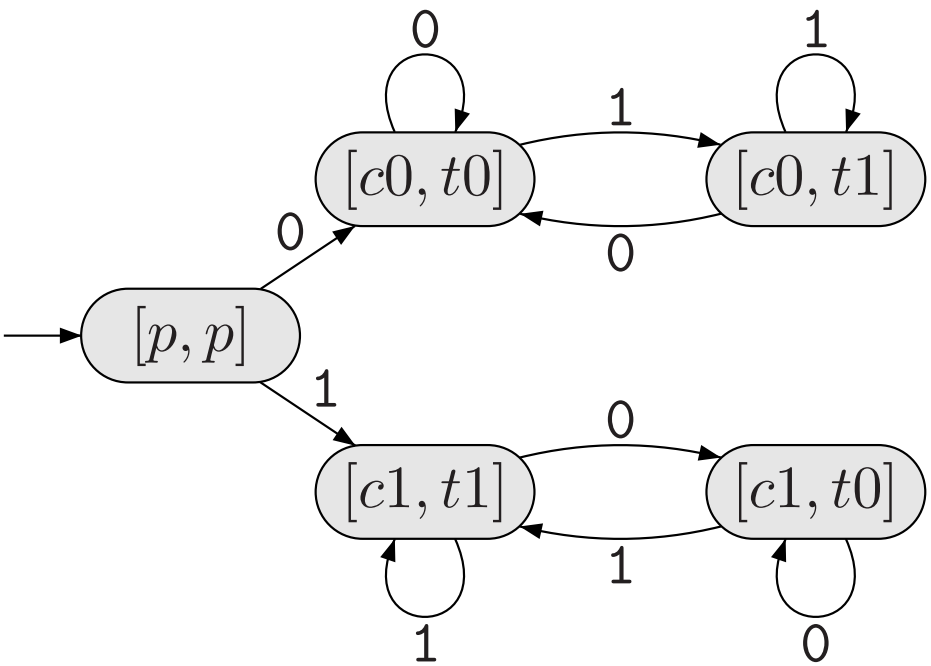
$$\hat{\delta}_3([e_1, e_2], w) = [\hat{\delta}_1(e_1, w), \hat{\delta}_2(e_2, w)], \text{ para todo } w \in \Sigma^*.$$

Dois AFDs



(a) Reconhecendo $\{0\}\{0,1\}^*$ **(b) Reconhecendo $\{0,1\}^*\{1\}$**

O produto



Complemento, interseção e união

Sejam $M_1 = (E_1, \Sigma, \delta_1, i_1, F_1)$ e $M_2 = (E_2, \Sigma, \delta_2, i_2, F_2)$.

- AFD para $\overline{L(M_1)}$:

$$(E_1, \Sigma, \delta_1, i_1, E_1 - F_1)$$

Complemento, interseção e união

Sejam $M_1 = (E_1, \Sigma, \delta_1, i_1, F_1)$ e $M_2 = (E_2, \Sigma, \delta_2, i_2, F_2)$.

- AFD para $\overline{L(M_1)}$:

  $(E_1, \Sigma, \delta_1, i_1, E_1 - F_1)$

- AFD para $L(M_1) \cap L(M_2)$:

  produto de M_1 e M_2 com $F_3 = F_1 \times F_2$

Complemento, interseção e união

Sejam $M_1 = (E_1, \Sigma, \delta_1, i_1, F_1)$ e $M_2 = (E_2, \Sigma, \delta_2, i_2, F_2)$.

- AFD para $\overline{L(M_1)}$:

  $(E_1, \Sigma, \delta_1, i_1, E_1 - F_1)$

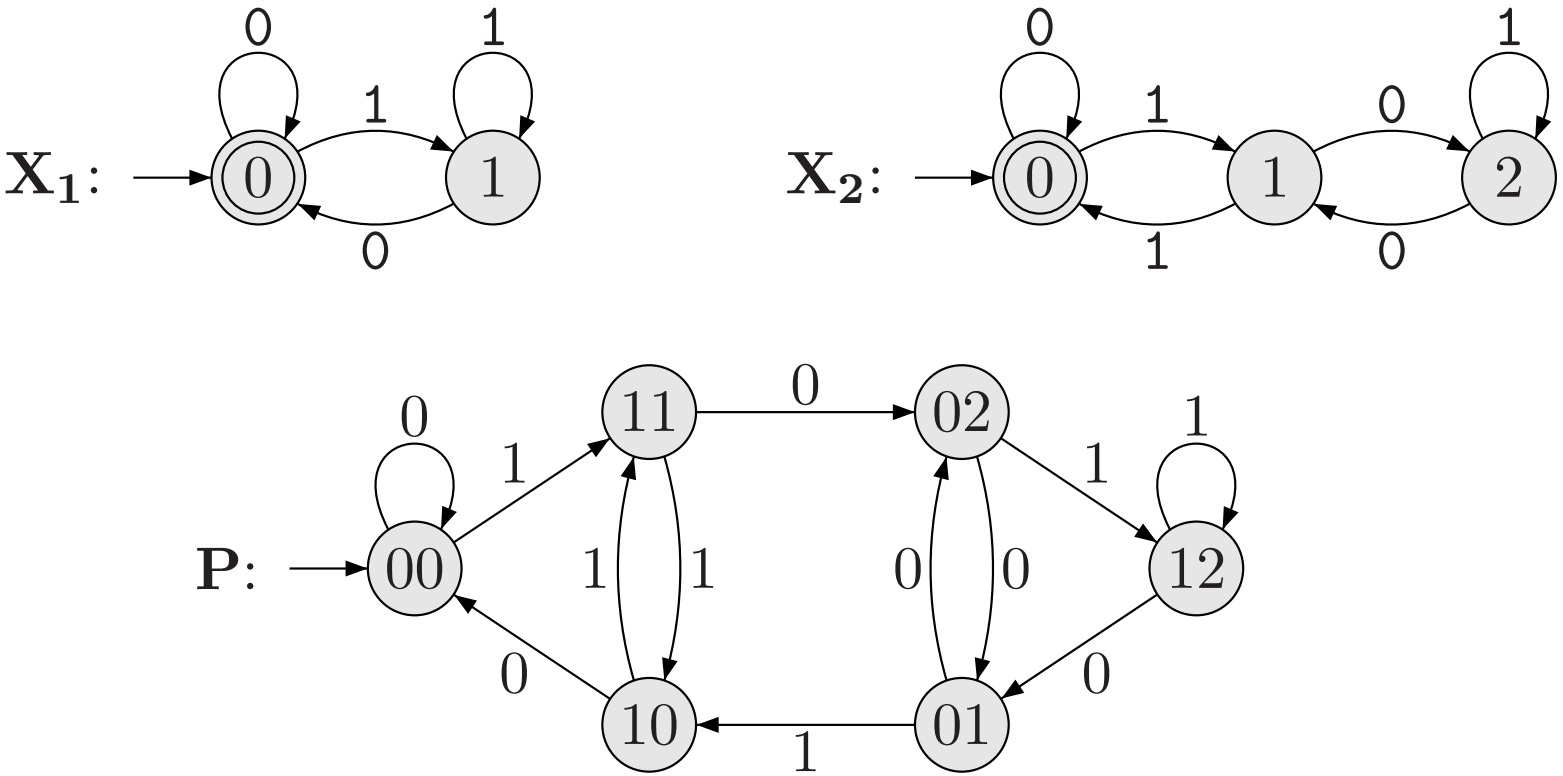
- AFD para $L(M_1) \cap L(M_2)$:

  produto de M_1 e M_2 com $F_3 = F_1 \times F_2$

- AFD para $L(M_1) \cup L(M_2)$:

  produto de M_1 e M_2 com $F_3 = (F_1 \times E_2) \cup (E_1 \times F_2)$

Outro exemplo



Linguagens finitas

Para toda linguagem finita existe AFD.

AFD com diagrama de estados simplificado sem ciclos:

é uma árvore em que o estado inicial é a raiz.

Linguagens finitas

Para toda linguagem finita existe AFD.

AFD com diagrama de estados simplificado sem ciclos:

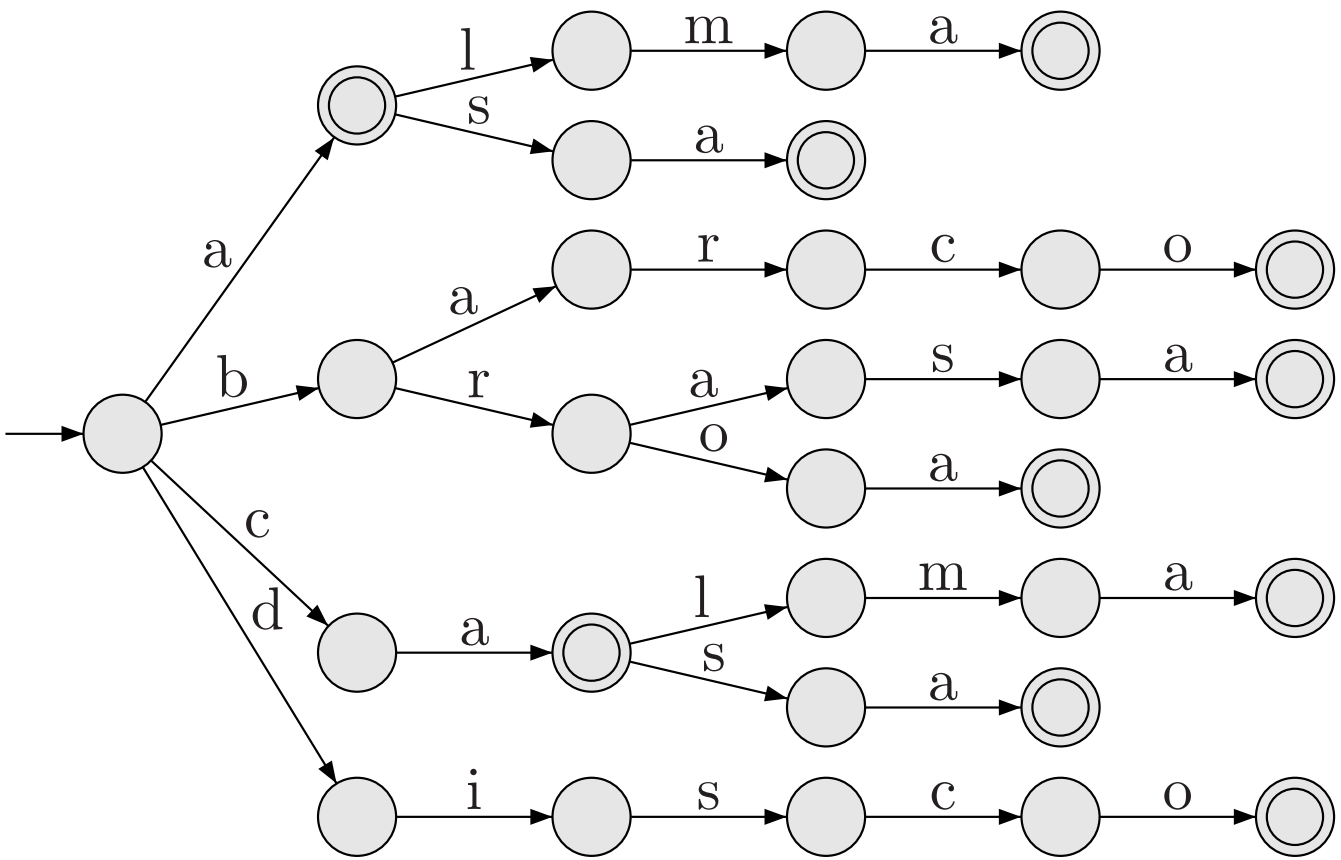
é uma árvore em que o estado inicial é a raiz.

Exemplo

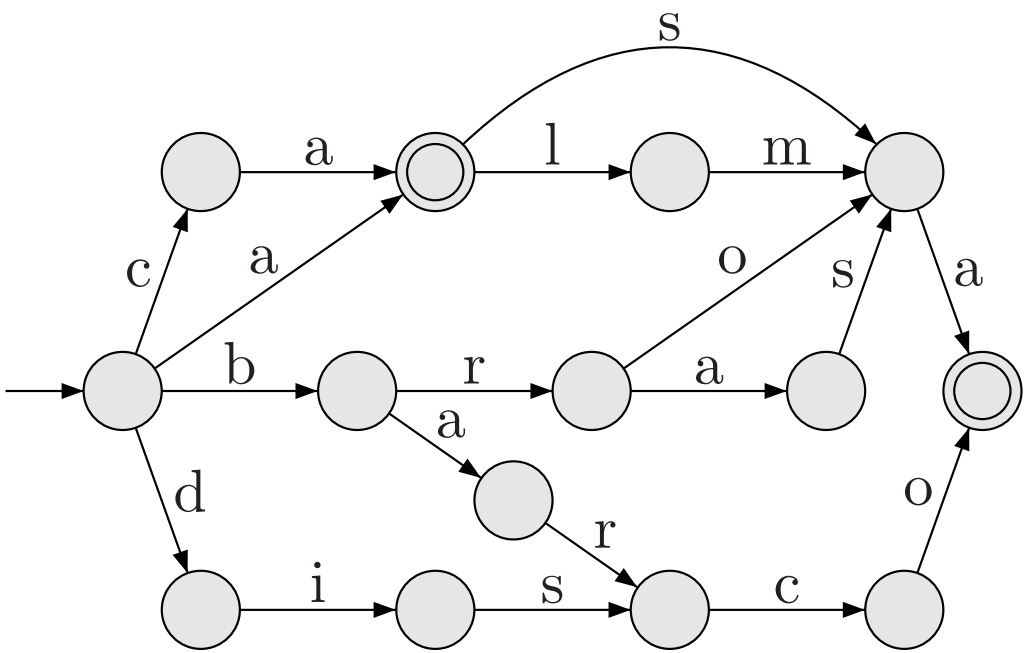
Um pequeno dicionário:

$$K = \{\texttt{a}, \texttt{alma}, \texttt{asa}, \texttt{barco}, \texttt{brasa}, \texttt{broa}, \texttt{ca}, \texttt{calma}, \texttt{casa}, \texttt{disco}\}$$

AFD para o pequeno dicionário



AFD minimizado para o pequeno dicionário



Uma liguagem não reconhecível por AFD

$$\{a^n b^n \mid n \geq 0\}$$

Um teorema útil para provar que não há AFD

Teorema

Seja um AFD M de k estados, e $z \in L(M)$ tal que $|z| \geq k$. Existem u, v e w tais que:

- $z = uvw$
- $v \neq \lambda$
- $uv^i w \in L(M)$ *para todo* $i \geq 0$

Um teorema útil para provar que não há AFD

Teorema

Seja um AFD M de k estados, e $z \in L(M)$ tal que $|z| \geq k$. Existem u, v e w tais que:

- $z = uvw$
- $v \neq \lambda$
- $uv^i w \in L(M)$ *para todo* $i \geq 0$

Demonstração

Se $z \in L(M)$ e $|z| \geq k$, a computação para z percorre um ciclo. Basta tomar $z = uvw$, em que v é a subpalavra consumida ao percorrer o ciclo.

Alguns problemas de decisão envolvendo AFDs

Existem procedimentos de decisão para determinar se:

1. $L(M) = \emptyset$
2. $L(M) = \Sigma^*$
3. $L(M)$ é finita

Alguns problemas de decisão envolvendo AFDs

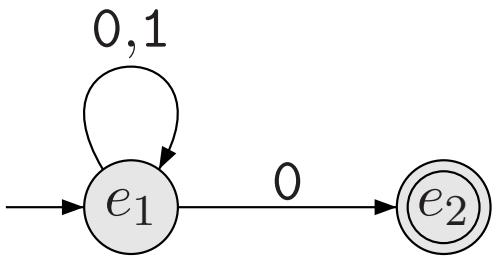
Existem procedimentos de decisão para determinar se:

1. $L(M) = \emptyset$
2. $L(M) = \Sigma^*$
3. $L(M)$ é finita

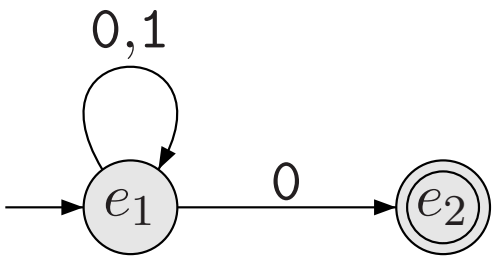
Seja M' um AFD mínimo equivalente a M. Então:

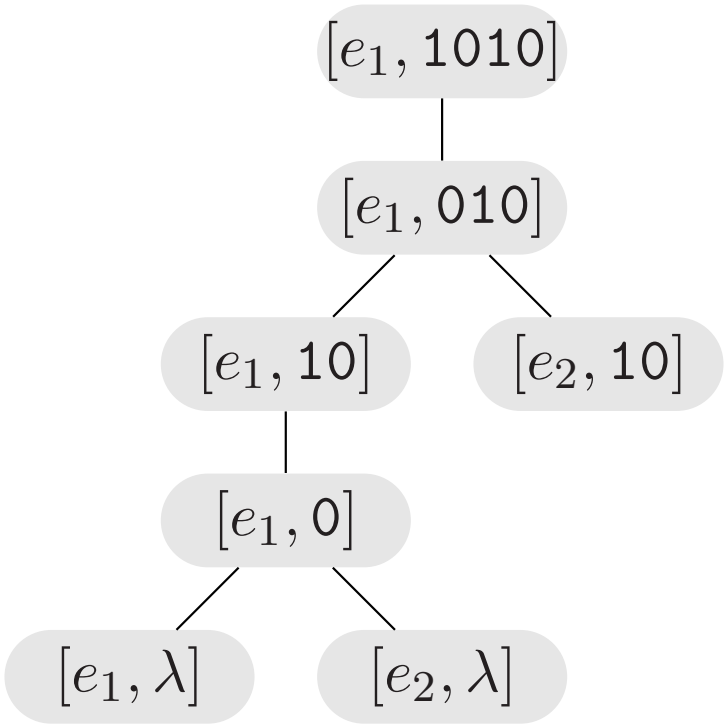
1. $L(M) = \emptyset$ sse M' tem um único estado e este não é final
2. $L(M) = \Sigma^*$ sse M' tem um único estado e este é final
3. $L(M)$ é finita sse o diagrama de estados simplificado de M' não possui ciclo, a não ser em estado de erro (se ele existir)

Exemplo de autômato finito não determinístico



Árvore de computações possíveis para 1010





Reconhecimento para AFN

- Uma palavra é reconhecida sse existe uma computação que a consome e termina em estado final.
- Em todo ponto de indecisão, a máquina adivinha qual escolha (se houver alguma) leva a uma computação que resulta em sucesso.

3. Autômatos Finitos Não Determinísticos
   - O que é autômato finito não determinístico
   - Equivalência entre AFDs e AFNs
   - AFN com transições λ

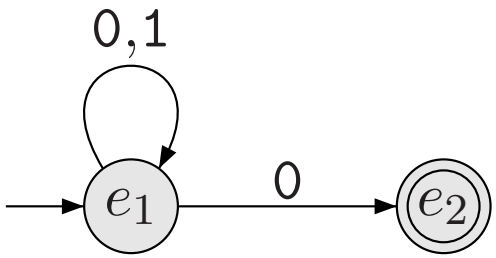
O que é AFN

Definição

Um AFN é uma quíntupla $(E, \Sigma, \delta, I, F)$, em que:

- *E é um conjunto finito não vazio de estados;*
- *Σ é um alfabeto;*
- *$I \subseteq E$ é um conjunto não vazio de estados iniciais;*
- *$F \subseteq E$, é um conjunto de estados finais;*
- *$\delta : E \times \Sigma \to \mathcal{P}(E)$ é a função de transição, uma função total.*

Exemplo de AFN



$$(\{e_1, e_2\}, \{0, 1\}, \delta, \{e_1\}, \{e_2\})$$

δ	0	1
e_1	$\{e_1, e_2\}$	$\{e_1\}$
e_2	$\emptyset$	$\emptyset$

Linguagem reconhecida por um AFN

Definição

A função de transição estendida para um AFN $M = (E, \Sigma, \delta, I, F)$, $\hat{\delta} : \mathcal{P}(E) \times \Sigma^* \to \mathcal{P}(E)$, *é definida assim:*

- $\hat{\delta}(\emptyset, w) = \emptyset$, *para todo* $w \in \Sigma^*$
- $\hat{\delta}(X, \lambda) = A$, *para todo* $X \subseteq E$
- $\hat{\delta}(X, ay) = \hat{\delta}(\bigcup_{e \in X} \delta(e, a), y)$, *para* $X \subseteq E$, $X \neq \emptyset$, $a \in \Sigma$ *e* $y \in \Sigma^*$

Linguagem reconhecida por um AFN

Definição

A função de transição estendida para um AFN $M = (E, \Sigma, \delta, I, F)$, $\hat{\delta} : \mathcal{P}(E) \times \Sigma^* \to \mathcal{P}(E)$, *é definida assim:*

- $\hat{\delta}(\emptyset, w) = \emptyset$, *para todo* $w \in \Sigma^*$
- $\hat{\delta}(X, \lambda) = A$, *para todo* $X \subseteq E$
- $\hat{\delta}(X, ay) = \hat{\delta}(\bigcup_{e \in X} \delta(e, a), y)$, *para* $X \subseteq E$, $X \neq \emptyset$, $a \in \Sigma$ *e* $y \in \Sigma^*$

A linguagem reconhecida por M é

$$L(M) = \{w \in \Sigma^* \mid \hat{\delta}(I, w) \cap F \neq \emptyset\}.$$

$\hat{\delta}$ em forma procedural

Entrada: $X \subseteq E$ e $w \in \Sigma^*$
Saída: $\hat{\delta}(X, w)$.
enquanto $w \neq \lambda$ e $X \neq \emptyset$ **faça**
    sejam $a \in \Sigma$ e $y \in \Sigma^*$ tais que $w = ay$;
    $X \leftarrow \cup_{e \in X} \delta(e, a)$; $w \leftarrow y$
fimenquanto;
retorne X

Para que servem AFNs?

Dado que:

- Um AFN não pode ser implementado tão diretamente quanto um AFD.
- Para todo AFN existe um AFD equivalente.

por que o conceito de AFN é importante?

Para que servem AFNs?

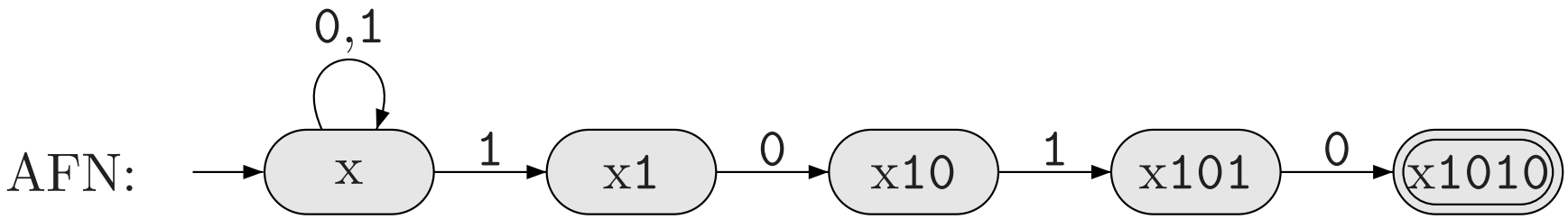
Dado que:

- Um AFN não pode ser implementado tão diretamente quanto um AFD.
- Para todo AFN existe um AFD equivalente.

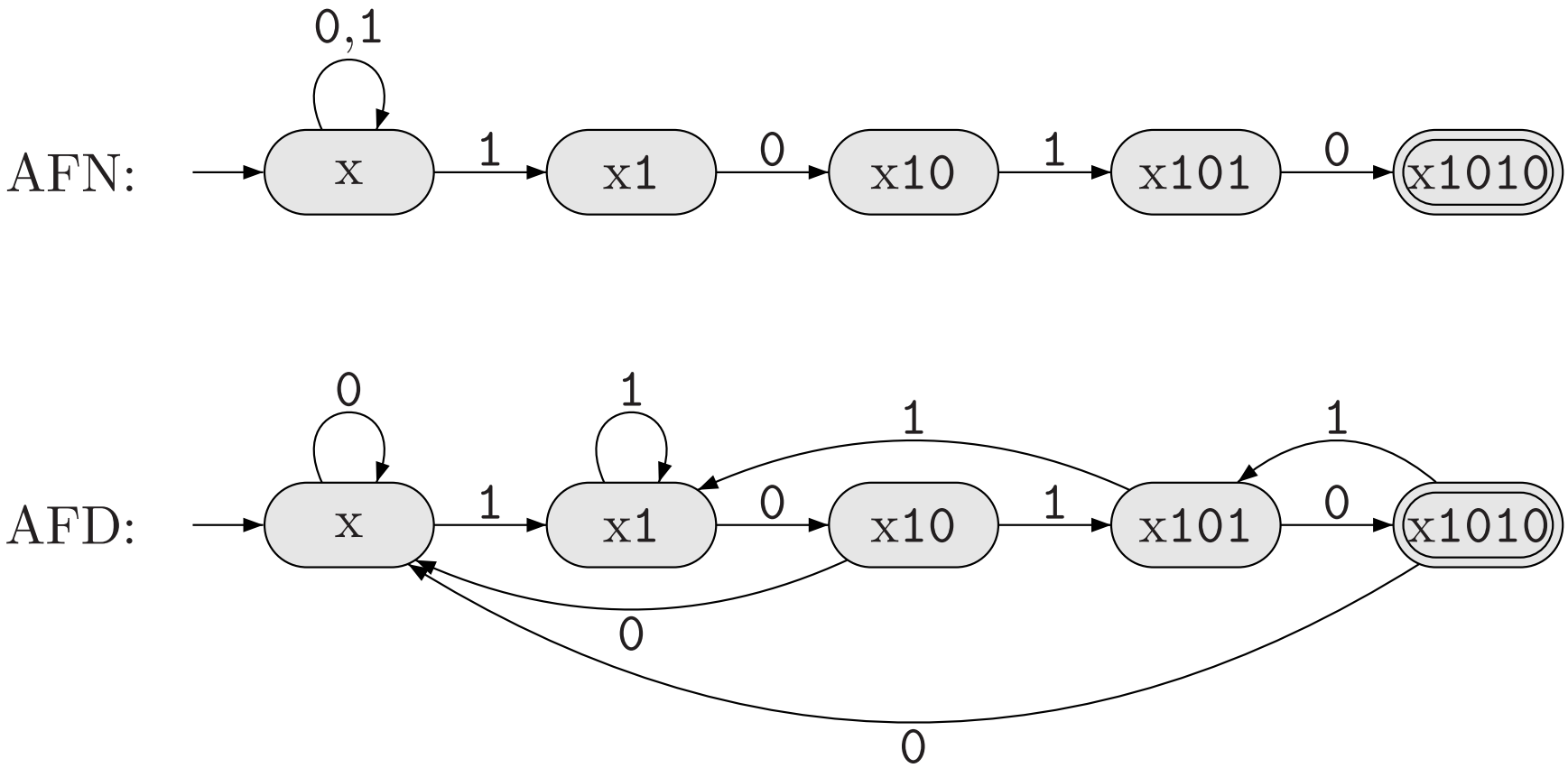
por que o conceito de AFN é importante?

1. Pode ser mais fácil construir um AFN que um AFD.
2. Um AFN pode ser mais claro que um AFD, dando mais confiança quanto à sua correção.
3. Um AFD pode ser exponencialmente maior que um AFN.
4. O conceito de não determinismo é importante em outras áreas da computação.

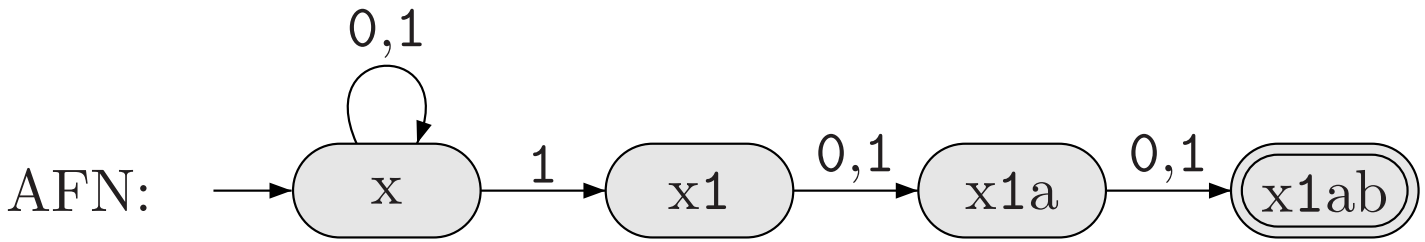
AFN para $\{0,1\}^*\{1010\}$



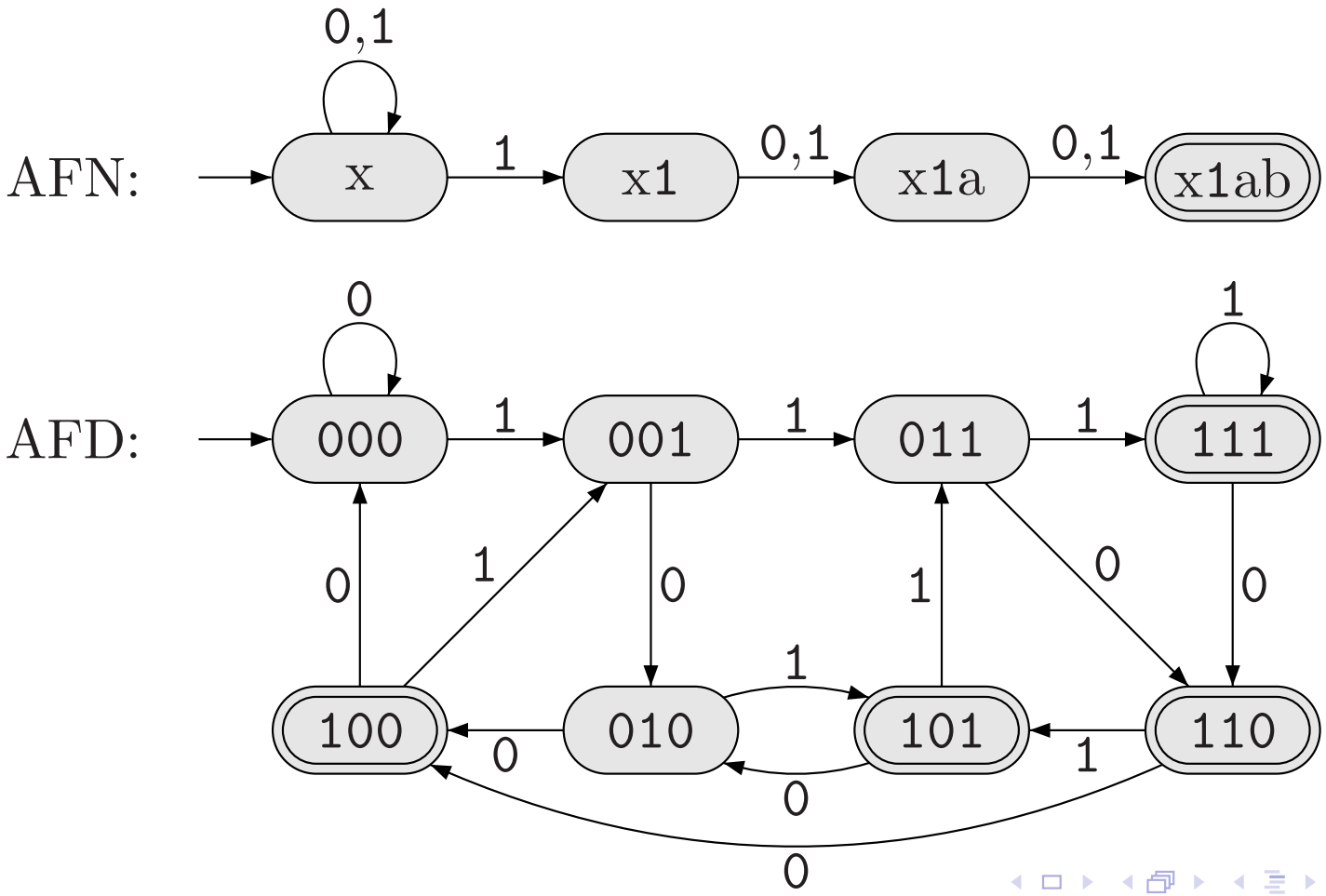
AFN e AFD para $\{0,1\}^*\{1010\}$



AFN para $\{0,1\}^*\{1\}\{0,1\}\{0,1\}$



AFN e AFD para $\{0,1\}^*\{1\}\{0,1\}\{0,1\}$



3. Autômatos Finitos Não Determinísticos
   - O que é autômato finito não determinístico
   - Equivalência entre AFDs e AFNs
   - AFN com transições λ

Para todo AFN existe um AFD equivalente

Idéia: Um estado do AFD será o conjunto dos estados do AFN atingidos por todas as computações possíveis para a mesma palavra.

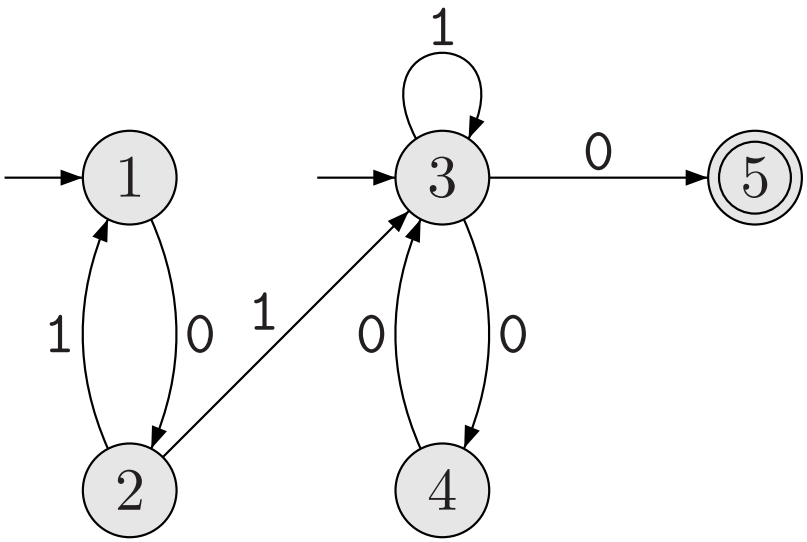
Para todo AFN existe um AFD equivalente

Idéia: Um estado do AFD será o conjunto dos estados do AFN atingidos por todas as computações possíveis para a mesma palavra.

Um AFD equivalente a um AFN $M = (E, \Sigma, \delta, I, F)$ é $M' = (\mathcal{P}(E), \Sigma, \delta', I, F')$, em que:

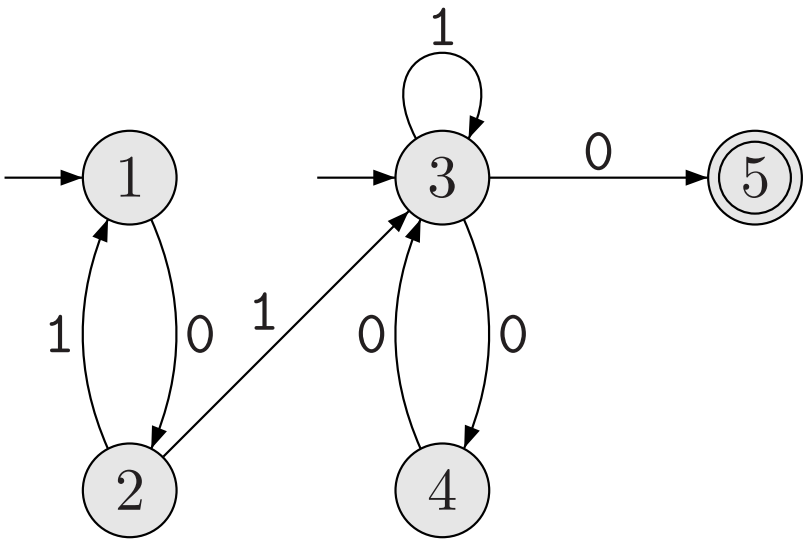
- para cada $X \subseteq E$ e $a \in \Sigma$, $\delta'(X, a) = \bigcup_{e \in X} \delta(e, a)$;
- $F' = \{X \subseteq E \mid X \cap F \neq \emptyset\}$.

Exemplo de obtenção de AFD a partir de AFN

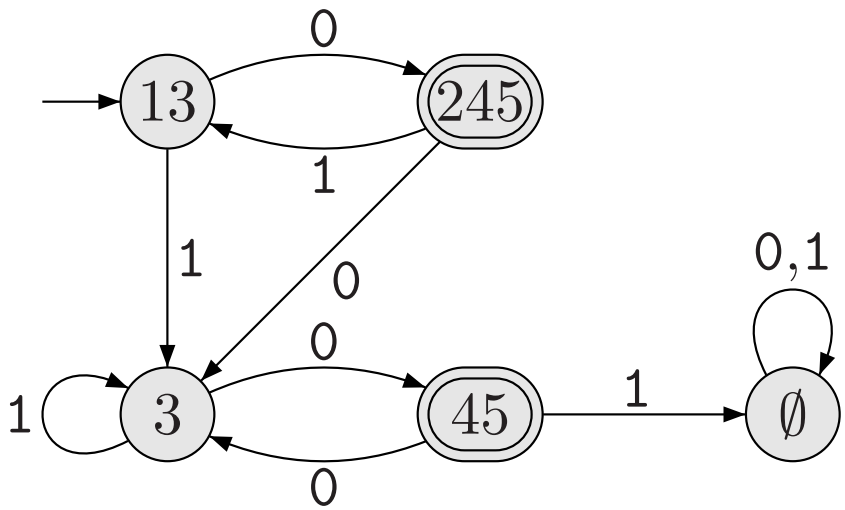


(a) AFN p/ $\{01\}^*\{00,1\}^*\{0\}$

Exemplo de obtenção de AFD a partir de AFN



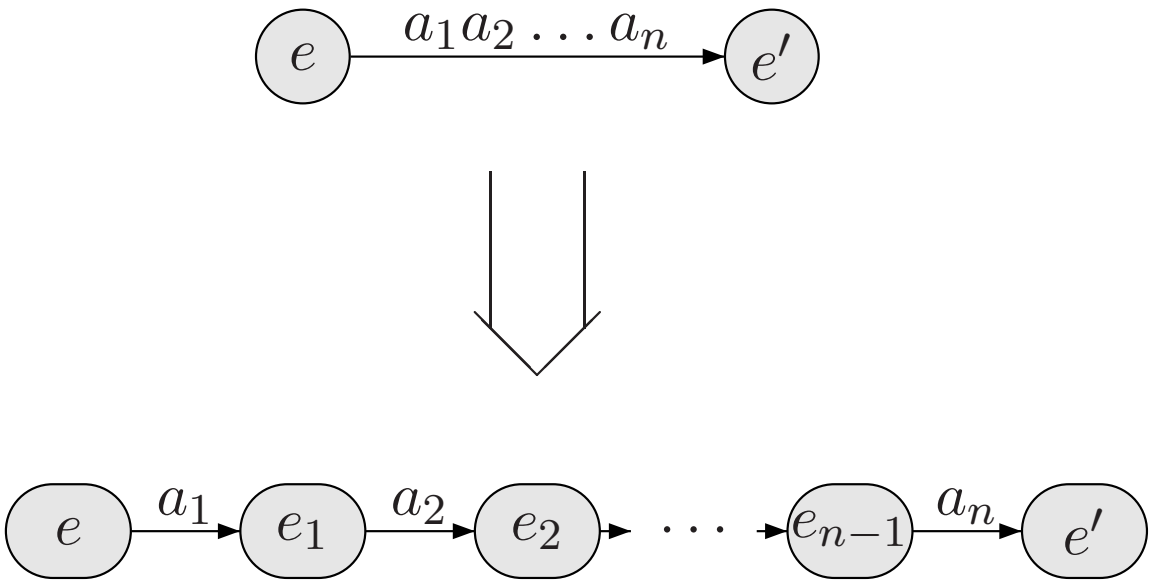
(a) AFN p/ $\{01\}^*\{00,1\}^*\{0\}$



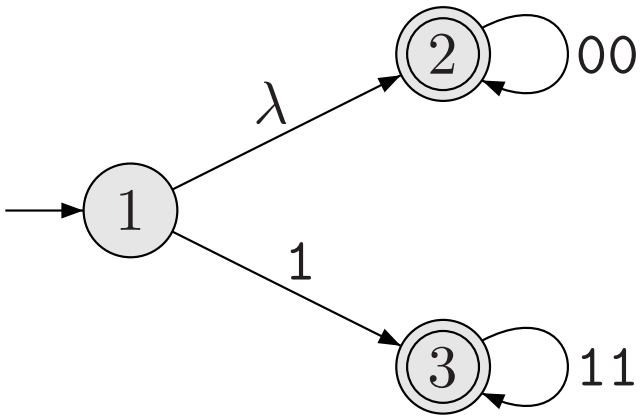
(b) AFD correspondente

3. Autômatos Finitos Não Determinísticos
   - O que é autômato finito não determinístico
   - Equivalência entre AFDs e AFNs
   - AFN com transições λ

Transições sob palavras com mais de um símbolo

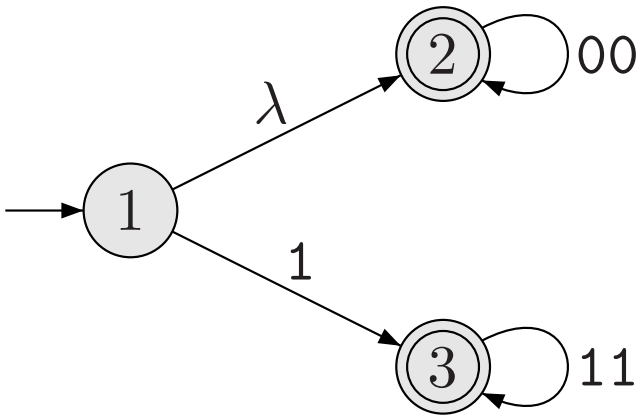


Um exemplo de AFN com transições sob palavras

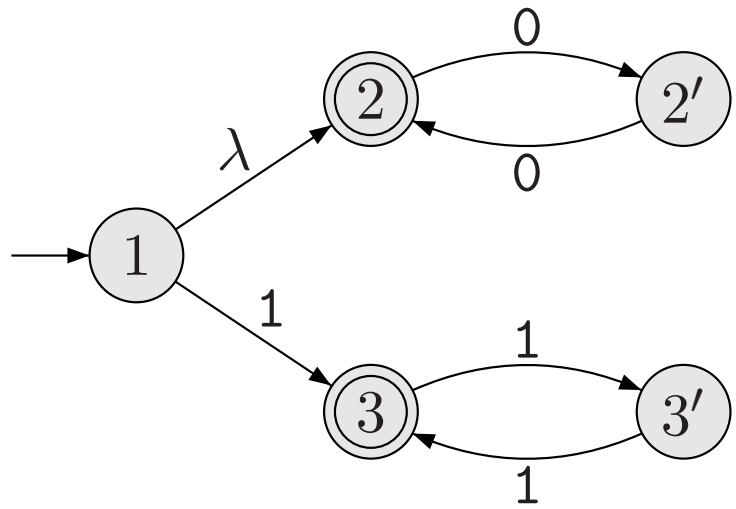


(a) AFN com palavras

Um exemplo de AFN com transições sob palavras



(a) AFN com palavras



(b) AFN com transição λ

AFN com transições λ

Notação

Σ_λ *denota o conjunto* $\Sigma \cup \{\lambda\}$

AFN com transições λ

Notação

Σ_λ *denota o conjunto* $\Sigma \cup \{\lambda\}$

Definição

Um autômato finito não determinístico com transições λ *(AFN*λ*) é uma quíntupla* $(E, \Sigma, \delta, I, F)$*, em que:*

- E, Σ, I *e* F *são como em AFNs; e*
- δ *é uma função total de* $E \times \Sigma_\lambda$ *para* $\mathcal{P}(E)$*.*

A função fecho λ

Seja um AFNλ $M = (E, \Sigma, \delta, I, F)$.

Definição

A função fecho λ para M, $f\lambda$: $\mathcal{P}(E) \rightarrow \mathcal{P}(E)$, é definida assim:

a) $X \subseteq f\lambda(X)$

b) *se* $e \in f\lambda(X)$, *então* $\delta(e, \lambda) \subseteq f\lambda(X)$

A função de transição estendida

Seja um AFNλ $M = (E, \Sigma, \delta, I, F)$.

Definição

A função de transição estendida, $\hat{\delta}$: $\mathcal{P}(E) \times \Sigma^* \to \mathcal{P}(E)$, *é definida assim:*

a) $\hat{\delta}(\emptyset, w) = \emptyset$, *para todo* $w \in \Sigma^*$;

b) $\hat{\delta}(X, \lambda) = f\lambda(A)$, *para* $X \subseteq E$;

c) $\hat{\delta}(X, ay) = \hat{\delta}(\bigcup_{e \in f\lambda(X)} \delta(e, a), y)$, *para* $X \subseteq E$, $X \neq \emptyset$, $a \in \Sigma$ *e* $y \in \Sigma^*$.

A linguagem reconhecida por um AFNλ

Definição

Seja $M = (E, \Sigma, \delta, I, F)$. *A linguagem reconhecida por* M *é*

$$L(M) = \{w \in \Sigma^* \mid \hat{\delta}(I, w) \cap F \neq \emptyset\}$$

$f\lambda$ em forma procedural

Entrada: $X \subseteq E$
Saída: $f\lambda(X)$.

$A \leftarrow X$; /* estados em aberto */
enquanto $A \neq \emptyset$ **faça**
    selecione $e \in A$;
    $A \leftarrow (A - \{e\}) \cup (\delta(e, \lambda) - X)$; $X \leftarrow X \cup \delta(e, \lambda)$
fimenquanto;
retorne X

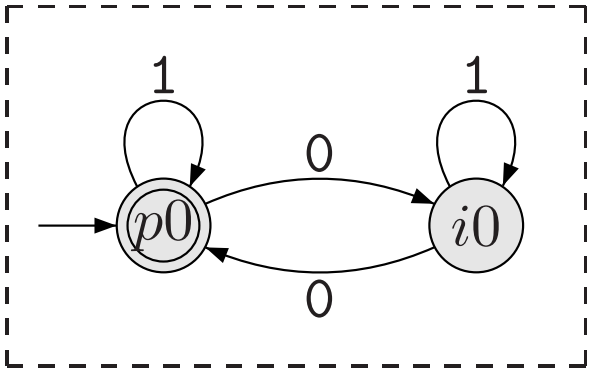
$\hat{\delta}$ em forma procedural

Entrada: $X \subseteq E$ e $w \in \Sigma^*$
Saída: $\hat{\delta}(X, w)$.

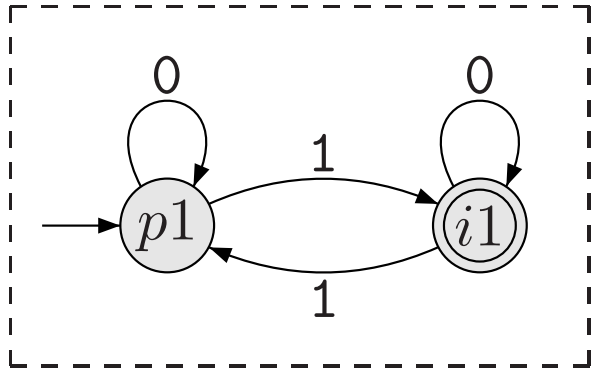
$X \leftarrow f\lambda(X)$;
enquanto $w \neq \lambda$ e $X \neq \emptyset$ **faça**
    sejam $a \in \Sigma$ e $y \in \Sigma^*$ tais que $w = ay$;
    $X \leftarrow \cup_{e \in f\lambda(X)} \delta(e, a)$; $w \leftarrow y$
fimenquanto;
retorne X

Exemplo: obtendo AFNλ para L_1L_2

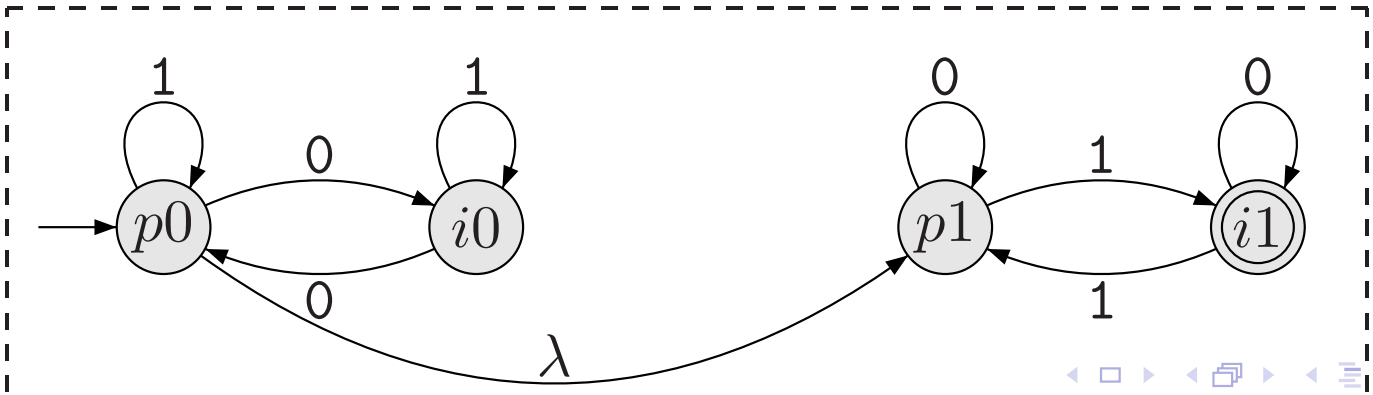
AFD para L_1



AFD para L_2



AFNλ para L_1L_2



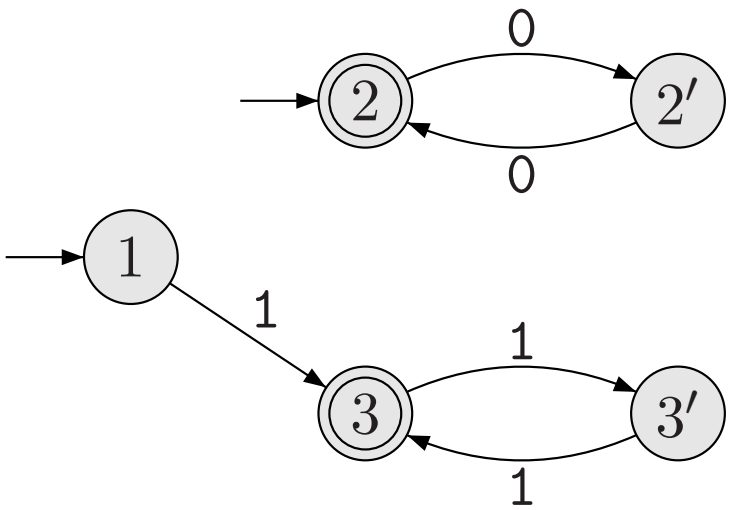
Obtenção de AFN equivalente a AFNλ

Seja um AFNλ $M = (E, \Sigma, \delta, I, F)$.
Um AFN equivalente a M é $M' = (E, \Sigma, \delta', I', F)$, em que:

- $I' = f\lambda(I)$; e
- $\delta'(e, a) = f\lambda(\delta(e, a))$, para cada $e \in E$ e $a \in \Sigma$.

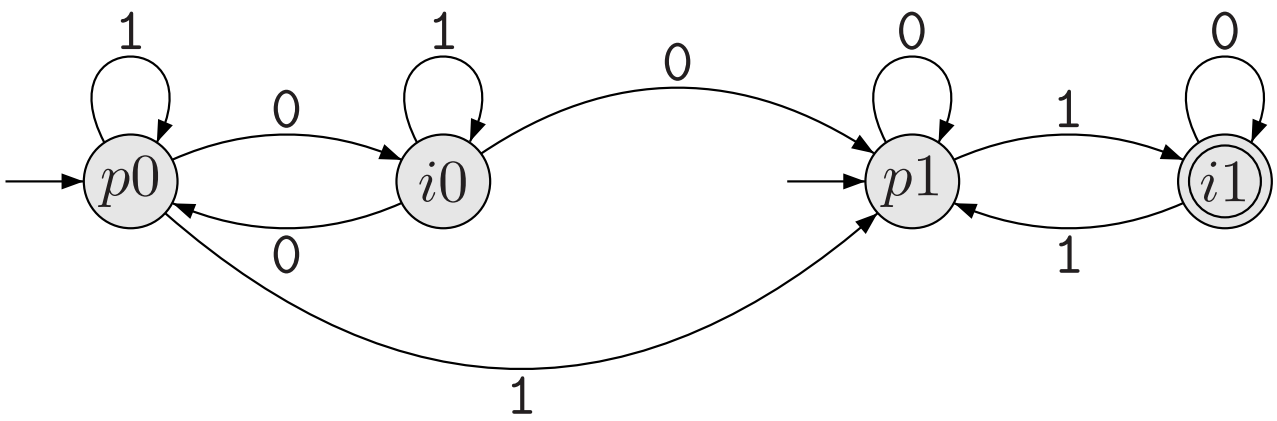
Um exemplo

▸ AFNλ



Outro exemplo

▸ AFNλ



Exemplo de composição de AFs para obter AFD

Exemplo

Construir um AFD que reconheça $(\{\mathtt{a}\}^* \cup \{\mathtt{b}\}^*)(\{\mathtt{c}\}^* \cup \{\mathtt{d}\}^*)$.

Exemplo de composição de AFs para obter AFD

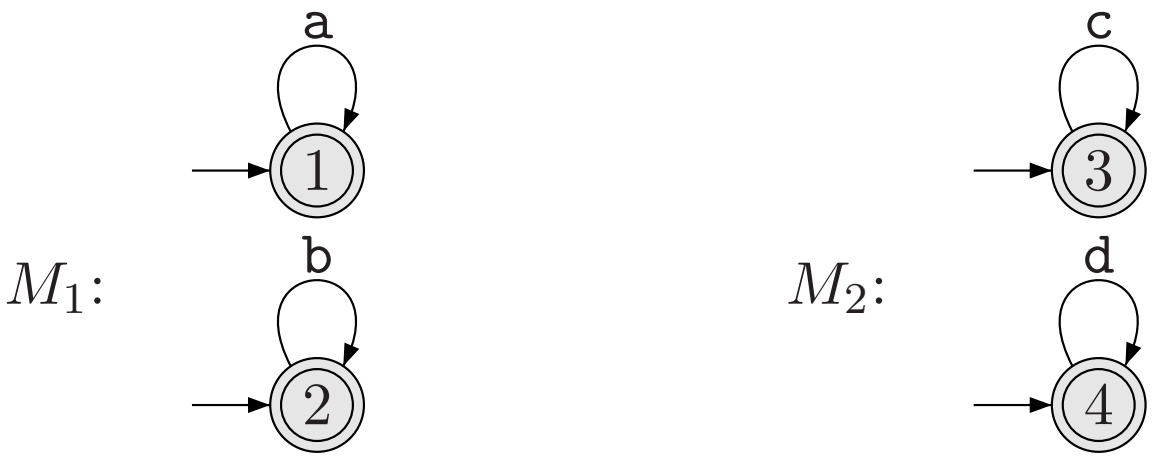
Exemplo

Construir um AFD que reconheça $(\{a\}^* \cup \{b\}^*)(\{c\}^* \cup \{d\}^*)$.

Passo 1: AFs para $\{a\}^*$, $\{b\}^*$, $\{c\}^*$ e $\{d\}^*$)

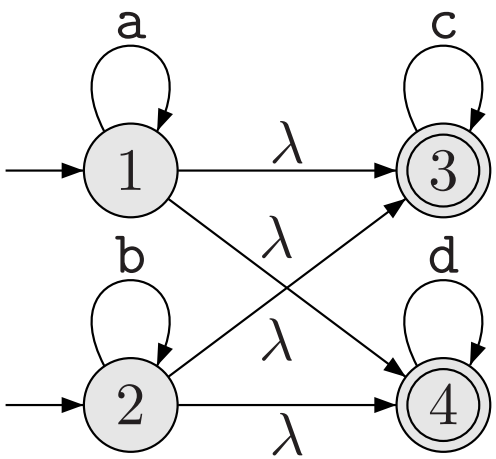
Exemplo de composição de AFs para obter AFD

Passo 2: AFs para $\{a\}^* \cup \{b\}^*$ e para $\{c\}^* \cup \{d\}^*$



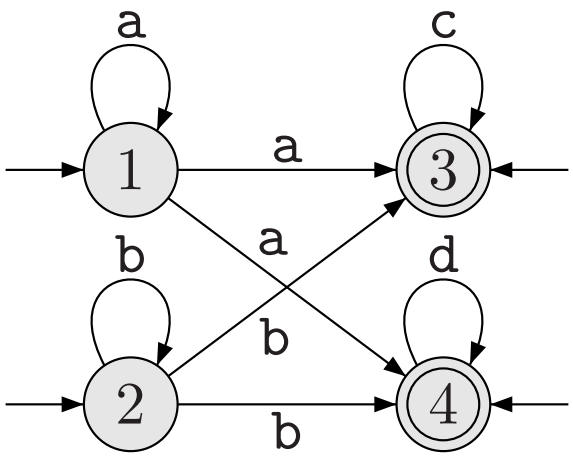
Exemplo de composição de AFs para obter AFD

Passo 3: AFNλ para $(\{a\}^* \cup \{b\}^*)(\{c\}^* \cup \{d\}^*)$



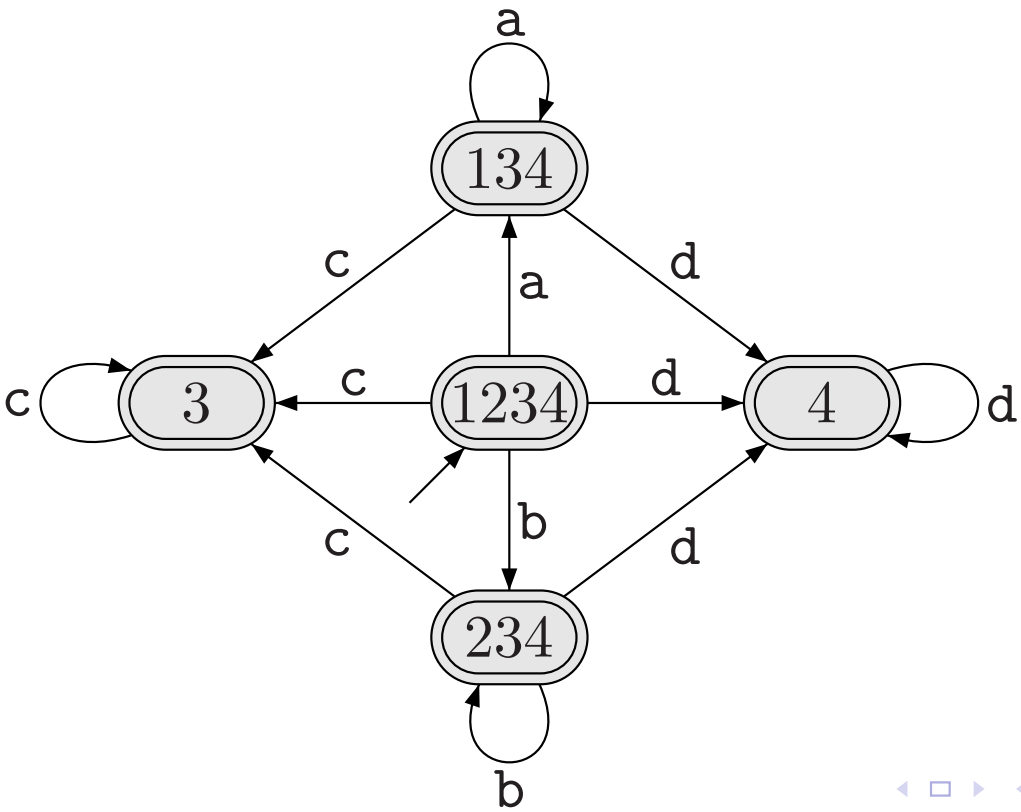
Exemplo de composição de AFs para obter AFD

Passo 4: de AFNλ para AFN



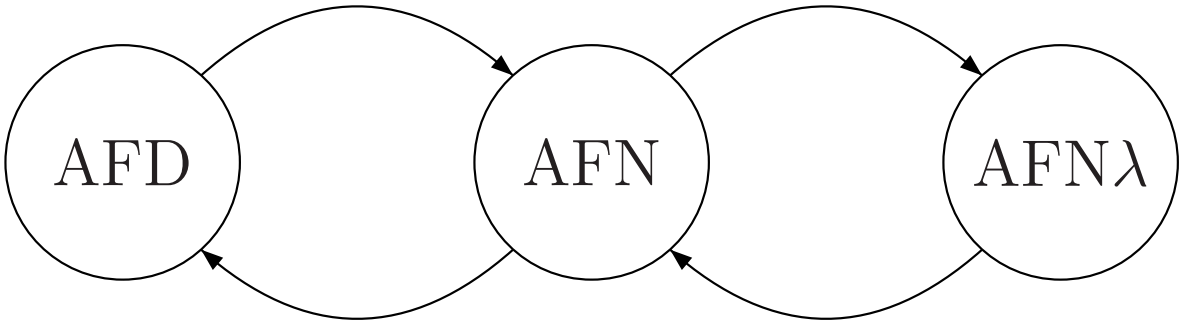
Exemplo de composição de AFs para obter AFD

Passo 5: de AFN para AFD



Relações entre tipos de autômatos finitos

é um caso especial de



pode ser transformado em

Linguagem Regular

Definição

Uma linguagem é dita ser uma linguagem regular (LR) se existe um autômato finito que a reconhece.

Caracterizações de linguagens regulares

Dada uma linguagem:

- É possível determinar se ela é ou não é regular?
- Se ela for regular, é possível facilitar a obtenção de um AF para L?

Caracterizações de linguagens regulares

Dada uma linguagem:

- É possível determinar se ela é ou não é regular?
- Se ela for regular, é possível facilitar a obtenção de um AF para L?

- Três formas de mostrar que uma linguagem não é regular:
  1. teorema da invariância à direita
  2. lema do bombeamento
  3. propriedades de fecho
- Para mostrar que é regular e facilitar a obtenção de AF:
  - propriedades de fecho

4. Linguages Regulares: Propriedades
   - Invariância à direita
   - Propriedades de fechamento
   - O lema do bombeamento

O teorema da invariância à direita

Teorema

Se L é uma linguagem regular de alfabeto Σ, então para qualquer $R \subseteq \Sigma^*$ infinito existem $x, y \in R$ tais que $x \neq y$ e para toda $z \in \Sigma^*$ $xz \in L \leftrightarrow yz \in L$.

O teorema da invariância à direita

Teorema

Se L é uma linguagem regular de alfabeto Σ, então para qualquer $R \subseteq \Sigma^*$ infinito existem $x, y \in R$ tais que $x \neq y$ e para toda $z \in \Sigma^*$ $xz \in L \leftrightarrow yz \in L$.

A contrapositiva

Existe um conjunto infinito R tal que para cada par $x, y \in R$, $x \neq y$, existe $z \in \Sigma^*$ tal que $xz \in L$ e $yz \notin L$, ou vice-versa, $yz \in L$ e $xz \notin L \rightarrow L$ não é regular.

Um exemplo

Teorema

A linguagem $L = \{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbf{N}\}$ *não é regular.*

Demonstração

Seja $R = \{\mathtt{a}\}^*$. *Sejam duas palavras arbitrárias* $\mathtt{a}^i, \mathtt{a}^j \in R$, $i \neq j$. *Como* $\mathtt{a}^i\mathtt{b}^i \in L$ *e* $\mathtt{a}^j\mathtt{b}^i \notin L$, *segue-se, pelo TID, que* L *não é regular.*

Um exemplo

Teorema

A linguagem $L = \{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbf{N}\}$ *não é regular.*

Demonstração

Seja $R = \{\mathtt{a}\}^*$. *Sejam duas palavras arbitrárias* $\mathtt{a}^i, \mathtt{a}^j \in R$, $i \neq j$. *Como* $\mathtt{a}^i\mathtt{b}^i \in L$ *e* $\mathtt{a}^j\mathtt{b}^i \notin L$, *segue-se, pelo TID, que* L *não é regular.*

Uso do teorema para mostrar que L não é regular:

1. encontrar um conjunto infinito de palavras R tal que
2. para quaisquer duas palavras diferentes x e y de R exista uma palavra z (possivelmente dependente de x ou y) tal que ou $xz \in L$ ou $yz \in L$, mas não ambos.

Mais um exemplo

Teorema

$L = \{0^m 1^n \mid m > n\}$ *não é regular.*

Demonstração

Tome $R = \{0\}^*$ *e sejam* $0^i, 0^j \in R$, $i < j$, *palavras arbitrárias de* R. *Tem-se que* $0^i 1^i \notin L$ *e* $0^j 1^i \in L$. *Logo, pelo teorema da invariância à direita,* L *não é regular.*

Ainda mais um exemplo

Teorema

$L = \{w \in \{0,1\}^* \mid w = w^R\}$ *não é regular.*

Demonstração

Tome $R = \{0\}^*\{1\}$ *e sejam* $0^i1, 0^j1 \in R$, $i < j$, *palavras arbitrárias de* R. *Tem-se que* $0^i10^i \in L$ *e* $0^j10^i \notin L$. *Assim, pelo teorema da invariância à direita,* L *não é regular.*

Outro exemplo

Teorema

$L = \{0^{n^2} \mid n \in \mathbb{N}\}$.

Demonstração

Seja o conjunto $R = L$. *Sejam* $0^{i^2}, 0^{j^2} \in R$, $j > i$, *quaisquer. Segue-se que* $0^{i^2}0^{2i+1} = 0^{(i+1)^2} \in L$, *mas* $0^{j^2}0^{2i+1} \notin L$, *pois* $j^2 < j^2 + 2i + 1 < (j+1)^2$). *Portanto, pelo teorema da invariância à direita,* L *não é regular.*

O problema de encontrar R adequado nem sempre é trivial

$\delta(k)$: menor inteiro positivo tal $k + \delta(k)$ é primo.

Teorema

$L = \{0^n \mid$ n é primo$\}$ não é regular.

Demonstração

Seja $R = \{0^{p_0}, 0^{p_1}, 0^{p_2}, \ldots\}$ em que p_0, p_1, p_2, ... é a sequência de números primos definida recursivamente assim:

- *$p_0 = 2$;*
- *para $n \geq 1$, p_n é o menor número primo maior ou igual a $p_{n-1} + \delta(p_{n-1})$ tal que $p_n + \delta(p_n) > p_{n-1} + \delta(p_{n-1})$.*

Sejam $0^{p_i}, 0^{p_j} \in R$, $i < j$, arbitrários. Tem-se que $p_i + \delta(p_i)$ é primo, mas $p_j + \delta(p_i)$ não é. Portanto, $0^{p_i} 0^{\delta(p_i)} \in L$ e $0^{p_j} 0^{\delta(p_i)} \notin L$. Pelo teorema da invariância à direita, L não é regular.

4. Linguages Regulares: Propriedades
   - Invariância à direita
   - Propriedades de fechamento
   - O lema do bombeamento

Algumas propriedades de fechamento

O que é fechamento

Seja uma classe de linguagens, $\mathcal{L}$, e uma operação sobre linguagens, O. Diz-se que $\mathcal{L}$ é *fechada* sob O se a aplicação de O a linguagens de $\mathcal{L}$ resulta sempre em uma linguagem de $\mathcal{L}$.

Algumas propriedades de fechamento

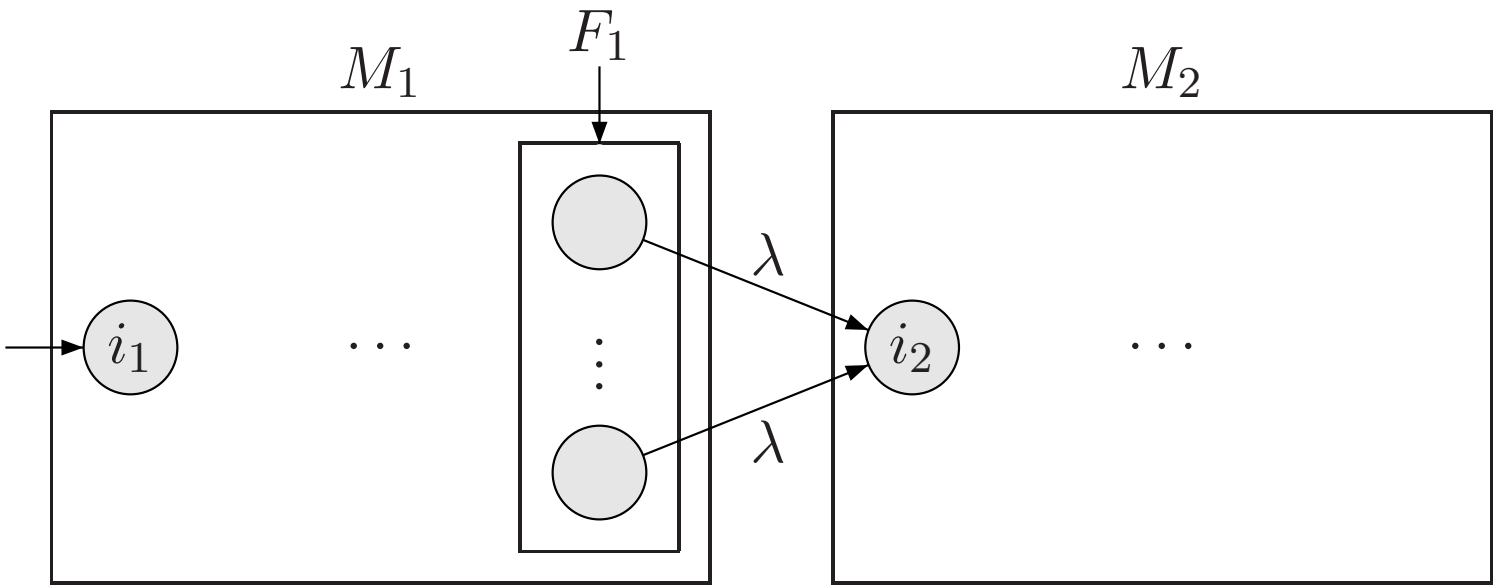
O que é fechamento

Seja uma classe de linguagens, $\mathcal{L}$, e uma operação sobre linguagens, O. Diz-se que $\mathcal{L}$ é *fechada* sob O se a aplicação de O a linguagens de $\mathcal{L}$ resulta sempre em uma linguagem de $\mathcal{L}$.

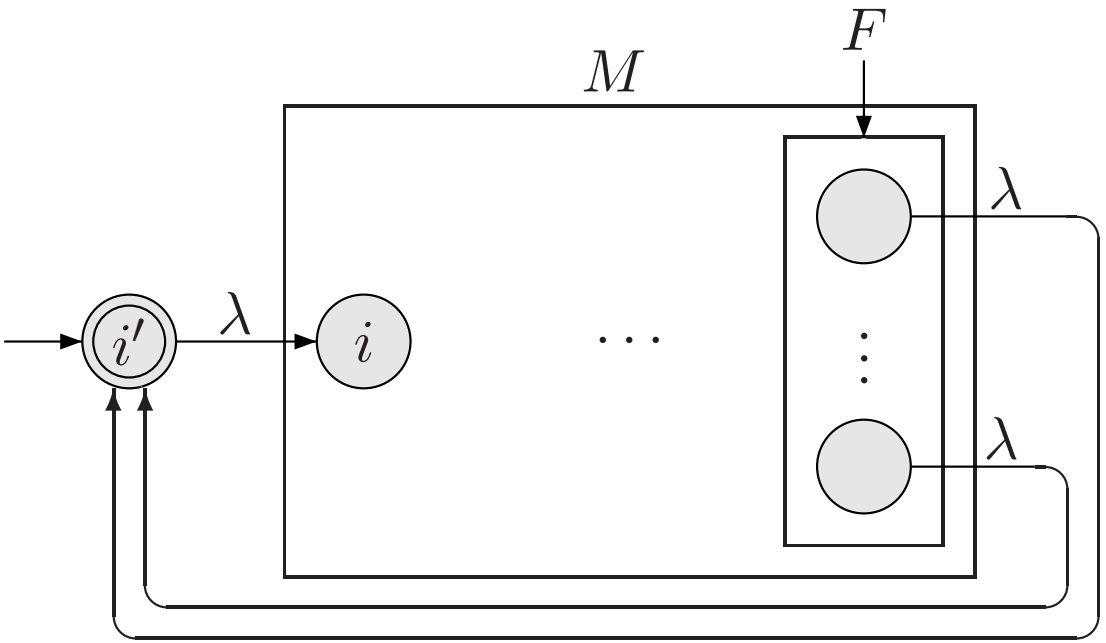
A classe das linguagens regulares é fechada sob:

1. complementação
2. união
3. interseção
4. concatenação
5. fecho de Kleene

Fechamento sob concatenação



Fechamento sob fecho de kleene



Aplicações das propriedades de fechamento

Três aplicações das propriedades de fecho das linguagens regulares:

1. provar que uma linguagem é regular;
2. provar que uma linguagem não é regular;
3. facilitar a obtenção de AF para linguagem regular.

Exemplo de aplicação do tipo 1

- $L_1 = \{w \in \{0,1\}^* \mid w \text{ representa número divisível por } 6\}$
- $L_2 = \{0,1\}^*\{1\}\{0,1\}^2$

$L_1 - L_2$ é regular?

Exemplo de aplicação do tipo 3

- $L_1 = \{w \in \{0,1\}^* \mid w \text{ representa número divisível por } 6\}$
- $L_2 = \{0,1\}^*\{1\}\{0,1\}^2$

Como construir um AFD para $L_1 - L_2$?

Exemplo de aplicação do tipo 2

Teorema

$L = \{\mathtt{a}^k\mathtt{b}^m\mathtt{c}^n \mid k = m + n\}$ *não é regular.*

Demonstração

Suponha que L seja uma linguagem regular. Como $\{\mathtt{a}\}^*\{\mathtt{b}\}^*$ *é linguagem regular e a classe das linguagens regulares é fechada sob interseção, segue-se que* $L \cap \{\mathtt{a}\}^*\{\mathtt{b}\}^*$ *deve ser uma linguagem regular. Mas,* $L \cap \{\mathtt{a}\}^*\{\mathtt{b}\}^* = \{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$*, que não é linguagem regular. Logo, L não é linguagem regular.*

4. Linguages Regulares: Propriedades
   - Invariância à direita
   - Propriedades de fechamento
   - O lema do bombeamento

Lema do bombeamento para linguagens regulares

Lema do bombeamento

Seja L uma linguagem regular. Então existe uma constante $k > 0$ tal que para qualquer palavra $z \in L$ com $|z| \geq k$ existem u, v e w que satisfazem as seguintes condições:

- $z = uvw$;
- $|uv| \leq k$;
- $v \neq \lambda$; e
- $uv^i w \in L$ para todo $i \geq 0$.

Uma aplicação do Lema do Bombeamento

O LB pode ser usado para provar que uma linguagem L não é regular assim:

1. supõe-se que L seja linguagem regular
2. escolhe-se $\boxed{\text{uma palavra } z \text{ tal que } |z| \geq k}$, sendo k a constante do LB
3. mostra-se que para toda decomposição de z em u v e w $\boxed{\text{existe } i \text{ tal que } uv^i w \notin L}$

Exemplo de uso do LB

Teorema

$L = \{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbf{N}\}$ *não é regular.*

Demonstração

Suponha que L seja regular. Seja k a constante do LB e $z = \mathtt{a}^k\mathtt{b}^k$. *Como* $|z| > k$, *o lema diz que existem u, v e w tais que:*

- $z = uvw$;
- $|uv| \leq k$;
- $v \neq \lambda$; *e*
- $uv^iw \in L$ *para todo* $i \geq 0$.

Neste caso, v só tem a*s, pois* $uvw = \mathtt{a}^k\mathtt{b}^k$ *e* $|uv| \leq k$, *e v tem pelo menos um* a, *porque* $v \neq \lambda$. *Isso implica que* $uv^2w = \mathtt{a}^{k+|v|}\mathtt{b}^k \notin L$, *o que contradiz o LB. Portanto, L não é linguagem regular.*

Outro exemplo

Teorema

$L = \{0^m 1^n \mid m > n\}$ *não é regular*

Demonstração

Suponha que L seja uma linguagem regular. Seja k a constante do LB, e seja $z = 0^{k+1}1^k$. *Como* $|z| > k$, *o lema diz que existem u, v e w tais que:*

- $z = uvw$;
- $|uv| \leq k$;
- $v \neq \lambda$; *e*
- $uv^i w \in L$ *para todo* $i \geq 0$.

Como $uvw = 0^{k+1}1^k$ *e* $0 < |v| \leq k$, *v só tem 0s e possui no mínimo um 0. Logo,* $uv^0 w = 0^{k+1-|v|}1^k \notin L$, *contradizendo o LB. Portanto,L não é regular.*

Mais um exemplo de uso do lema do bombeamento

Teorema

$L = \{0^n \mid \ n \text{ é primo}\}$ *não é regular*

Demonstração

Suponha que L seja regular. Seja k a constante do LB, e seja $z = 0^n$, *em que n é um número primo maior que k. Como* $|z| > k$, *para provar que L não é regular, basta mostrar um i tal que* $uv^i w \notin L$ *supondo que* $z = uvw$, $|uv| \leq k$ *e* $v \neq \lambda$. *Como* $z = 0^n$, $uv^i w = 0^{n+(i-1)|v|}$. *Assim, i deve ser tal que* $n + (i-1)|v|$ *não seja um número primo. Ora, para isso, basta fazer* $i = n + 1$, *obtendo-se* $n + (i-1)|v| = n + n|v| = n(1 + |v|)$, *que não é primo (pois* $|v| > 0$*). Desse modo,* $uv^{n+1} w \notin L$, *contradizendo o LB. Logo, L não é linguagem regular.*

Associando saída aos estados: máquina de Moore

Definição

Uma máquina de Moore é uma sêxtupla $(E, \Sigma, \Delta, \delta, \sigma, i)$, em que:

- *E (o conjunto de estados), Σ (o alfabeto de entrada), δ (a função de transição) e i (o estado inicial) são como em AFDs*
- *Δ é o alfabeto de saída*
- *$\sigma : E \to \Delta$ é a função de saída, uma função total*

A saída computada por uma máquina de Moore

Seja uma máquina de Moore $M = (E, \Sigma, \Delta, \delta, \sigma, i)$. A função de saída estendida para M, $r : E \times \Sigma^* \to \Delta^*$ é definida recursivamente assim:

a) $r(e, \lambda) = \sigma(e)$

b) $r(e, ay) = \sigma(e)r(\delta(e, a), y)$, para todo $a \in \Sigma$ e $y \in \Sigma^*$

A saída computada por uma máquina de Moore

Seja uma máquina de Moore $M = (E, \Sigma, \Delta, \delta, \sigma, i)$. A função de saída estendida para M, $r : E \times \Sigma^* \to \Delta^*$ é definida recursivamente assim:

a) $r(e, \lambda) = \sigma(e)$

b) $r(e, ay) = \sigma(e)r(\delta(e, a), y)$, para todo $a \in \Sigma$ e $y \in \Sigma^*$

A saída computada por uma máquina de Moore de estado inicial i para a palavra $w \in \Sigma^*$ é $r(i, w)$.

Interpretação da saída

- A interpretação de $r(i, w)$ é dependente da aplicação.
- Em algumas aplicações só interessa o último símbolo de $r(i, w)$.
- Em outras aplicações toda a palavra $r(i, w)$ é importante.

Interpretação da saída

- A interpretação de $r(i, w)$ é dependente da aplicação.
- Em algumas aplicações só interessa o último símbolo de $r(i, w)$.
- Em outras aplicações toda a palavra $r(i, w)$ é importante.

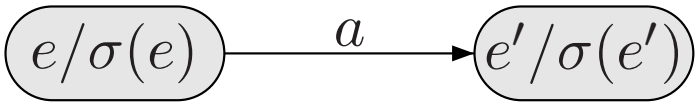
Dado um AFD $M = (E, \Sigma, \delta, i, F)$, uma máquina de Moore $M' = (E, \Sigma, \Delta, \delta, \sigma, i)$ que "simula" M é tal que:

- $\Delta = \{0, 1\}$
- $\sigma(e) = 1$ sse $e \in F$

Com isso: $w \in L(M)$ sse $r(i, w)$ termina em 1.

Diagrama de estados para máquinas de Moore

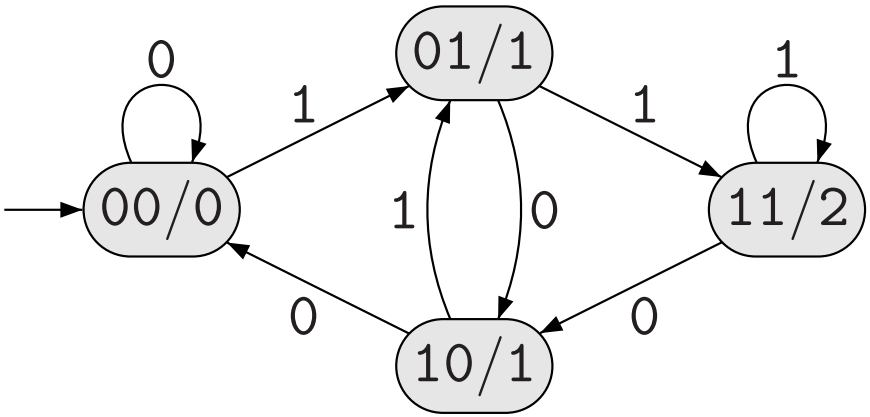
A transição $\delta(e, a) = e'$ é representada juntamente com $\sigma(e)$ e $\sigma(e')$ assim:



Exemplo de máquina de Moore

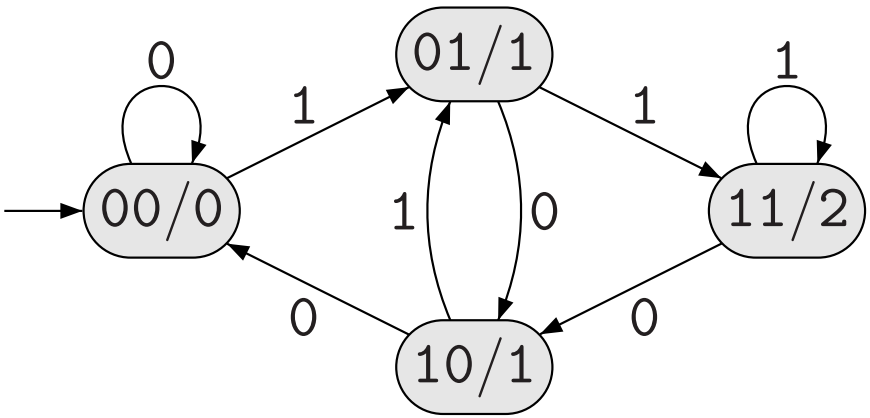
▸ Mealy

Último símbolo de $r(00, w)$: número de 1s nos dois últimos símbolos de w.



δ e σ no formato tabular

δ	0	1	σ
00	00	01	0
01	10	11	1
10	00	01	1
11	10	11	2



Contando número de 001s

O número de 001s é dado pelo número de 1s na palavra de saída.

Contando número de 001s

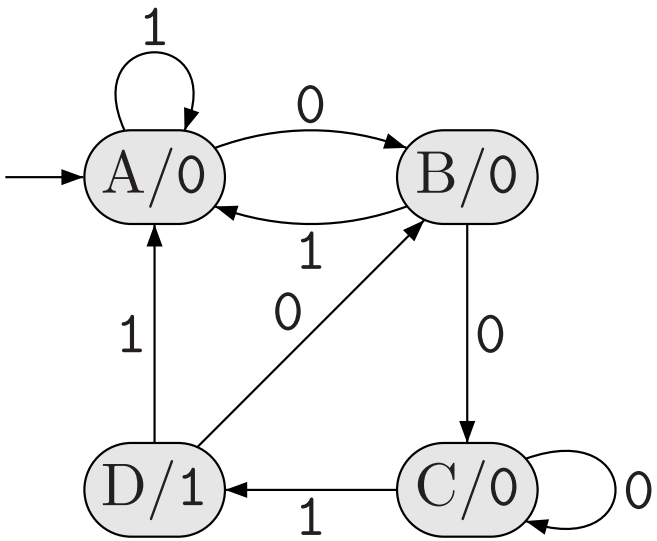
O número de 001s é dado pelo número de 1s na palavra de saída.

Exemplo

entrada: 10011010011010
saída: 00010000010000

A saída tem dois 1*s; logo, a de entrada tem duas subpalavras* 001*.*

Máquina de Moore que conta o número de 001s



Associando saída às transições: máquina de Mealy

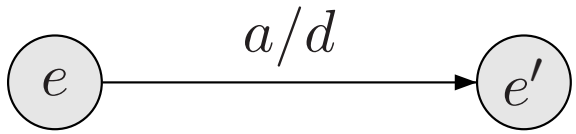
Definição

Uma máquina de Mealy é uma sêxtupla $(E, \Sigma, \Delta, \delta, \mu, i)$*, em que:*

- E *(o conjunto de estados),* Σ *(o alfabeto de entrada),* δ *(a função de transição) e* i *(o estado inicial) são como em AFDs*
- Δ *é o alfabeto de saída*
- $\mu : E \times \Sigma \to \Delta$ *é a função de saída, uma função total*

Diagrama de estados para máquina de Mealy

A transição $\delta(e, a) = e'$ é representada juntamente com $\sigma(e, a) = d$ assim:



A saída computada por uma máquina de Mealy

Seja uma máquina de Mealy $M = (E, \Sigma, \Delta, \delta, \sigma, i)$. A função de saída estendida para M, $s : E \times \Sigma^* \to \Delta^*$, é definida recursivamente assim:

a) $s(e, \lambda) = \lambda$

b) $s(e, ay) = \sigma(e, a)s(\delta(e, a), y)$, para todo $a \in \Sigma$ e $y \in \Sigma^*$

A saída computada por uma máquina de Mealy

Seja uma máquina de Mealy $M = (E, \Sigma, \Delta, \delta, \sigma, i)$. A função de saída estendida para M, $s : E \times \Sigma^* \to \Delta^*$, é definida recursivamente assim:

a) $s(e, \lambda) = \lambda$

b) $s(e, ay) = \sigma(e, a)s(\delta(e, a), y)$, para todo $a \in \Sigma$ e $y \in \Sigma^*$

A saída computada por uma máquina de Mealy $M = (E, \Sigma, \Delta, \delta, \sigma, i)$ para a palavra $w \in \Sigma^*$ é $s(i, w)$.

Quociente da divisão por 6 de número em binário

Sejam $\eta(x) = 6q_1 + r_1 \ (0 \leq r_1 < 6)$,
$\eta(xa) = 6q_2 + r_2 \ (0 \leq r_2 < 6)$.

Quociente da divisão por 6 de número em binário

Sejam $\eta(x) = 6q_1 + r_1 \ (0 \leq r_1 < 6)$,
$\eta(xa) = 6q_2 + r_2 \ (0 \leq r_2 < 6)$.

Dois casos:

$\underline{a = 0}$: Como $\eta(x0) = 2\eta(x)$, $6q_2 + r_2 = 2(6q_1 + r_1)$.
Logo, $q_2 = 2q_1 + (2r_1 - r_2)/6$.

$\underline{a = 1}$: Como $\eta(x1) = 2\eta(x)+1$, $6q_2+r_2 = 2(6q_1+r_1)+1$.
Logo, $q_2 = 2q_1 + (2r_1 + 1 - r_2)/6$.

Quociente da divisão por 6 de número em binário

Sejam $\eta(x) = 6q_1 + r_1 \ (0 \leq r_1 < 6)$,
$\eta(xa) = 6q_2 + r_2 \ (0 \leq r_2 < 6)$.

Dois casos:

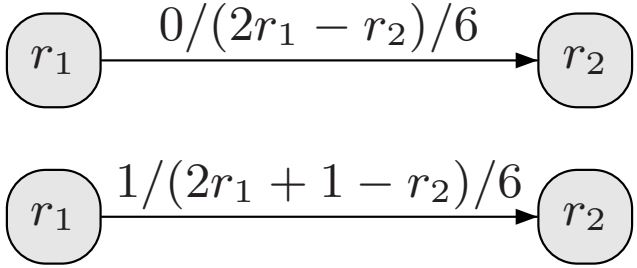
$\underline{a = 0}$: Como $\eta(x0) = 2\eta(x)$, $6q_2 + r_2 = 2(6q_1 + r_1)$.
Logo, $q_2 = 2q_1 + (2r_1 - r_2)/6$.

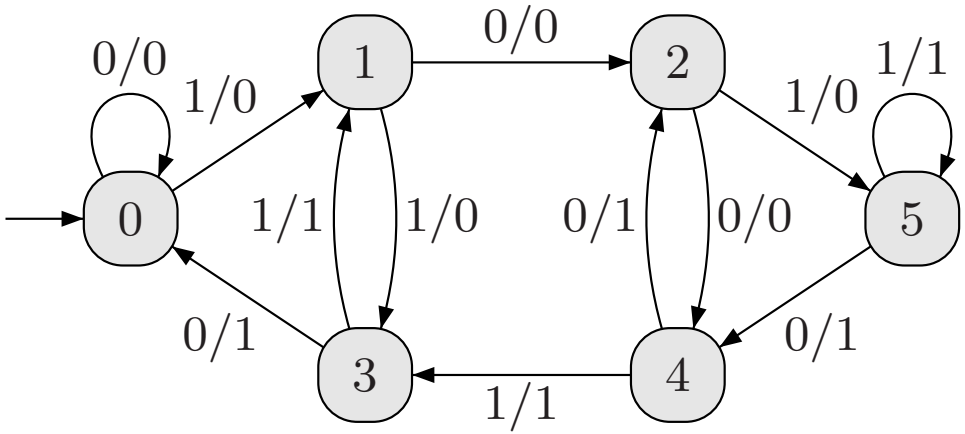
$\underline{a = 1}$: Como $\eta(x1) = 2\eta(x)+1$, $6q_2+r_2 = 2(6q_1+r_1)+1$.
Logo, $q_2 = 2q_1 + (2r_1 + 1 - r_2)/6$.

Portanto, se o próximo símbolo for:

- 0: o próximo dígito do quociente é $(2r_1 - r_2)/6$.
- 1: o próximo dígito do quociente é $(2r_1 - r_2 + 1)/6$.

Máquina Mealy para quociente da divisão por 6





Equivalência de máquinas de Moore e de Mealy

Definição

Uma máquina de Moore $(E_1, \Sigma, \Delta, \delta_1, \sigma_1, i_1)$ *e uma máquina de Mealy* $(E_2, \Sigma, \Delta, \delta_2, \sigma_2, i_2)$ *são ditas equivalentes se para todo* $w \in \Sigma^*$, $r(i_1, w) = \sigma_1(i_1)s(i_2, w)$.

Obtendo máquina de Mealy a partir de máquina de Moore

Seja uma máquina de Moore $M = (E, \Sigma, \Delta, \delta, \sigma, i)$.

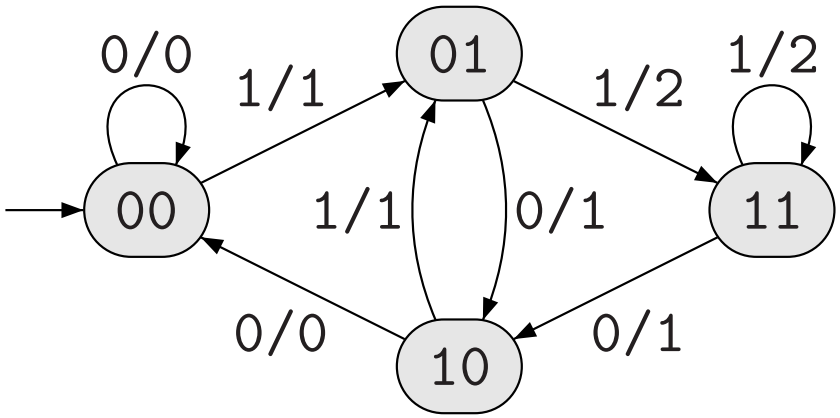
Máquina de Mealy equivalente:

$M' = (E, \Sigma, \Delta, \delta, \mu, i)$, em que:

$\mu(e, a) = \sigma(\delta(e, a))$, $\forall (e, a) \in E \times \Sigma$.

Um exemplo

Máquina de Moore ▸ Moore ⇒ Máquina de Mealy:



Obtendo máquina de Moore a partir de máquina de Mealy

Seja uma máquina de Mealy $M = (E, \Sigma, \Delta, \delta, \mu, i)$.

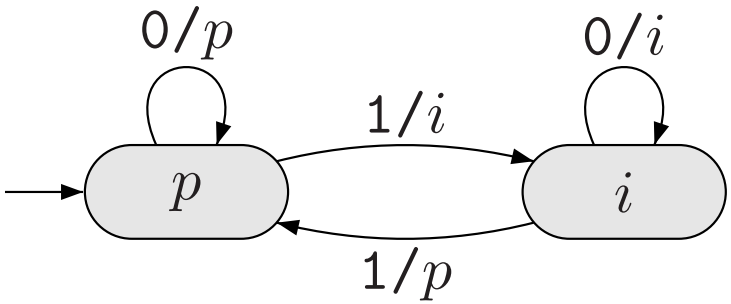
Máquina de Moore equivalente:

$M' = (E', \Sigma, \Delta, \delta', \sigma, i')$, em que:

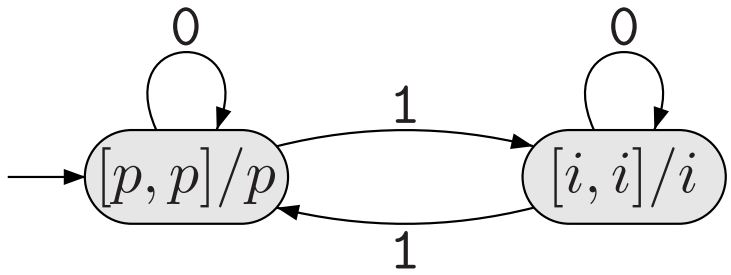
- $i' = [i, d_0]$ para um certo $d_0 \in \Delta$ (qualquer um serve);
- $E' = \{[\delta(e, a), \mu(e, a)] \mid e \in E \text{ e } a \in \Sigma\} \cup \{i'\}$;
- $\delta'([e, d], a) = [\delta(e, a), \mu(e, a)]$ para cada $[e, d] \in E'$ e $a \in \Sigma$;
- $\sigma([e, d]) = d$ para cada $e \in E$ e $d \in \Delta$.

Um exemplo

Máquina de Mealy $\Rightarrow$ Máquina de Moore:



(a) Máquina de Mealy



(a) Máquina de Moore

Denotação e geração de linguagens regulares

Duas novas formas de especificar as linguagens regulares:

- Expressão Regular: especifica uma linguagem por meio de uma expressão que a denota.
- Gramática Regular: especifica uma linguagem por meio de uma gramática que a gera.

O que é expressão regular

Definição

Uma expressão regular (ER) sobre um alfabeto Σ é definida recursivamente assim:

1. $\emptyset$, λ, *e* a *para qualquer* $a \in \Sigma$ *são expressões regulares; elas denotam* $\emptyset$, $\{\lambda\}$ *e* $\{a\}$;
2. *se* r *e* s *são expressões regulares, então são expressões regulares:* $(r + s)$, (rs), *e* r^*; *elas denotam* $L(r) \cup L(s)$, $L(r)L(s)$ *e* $L(r)^*$.

Exemplos de expressões regulares

ER sobre $\{0,1\}$	*Linguagem denotada*
$\emptyset$	$\emptyset$
λ	$\{\lambda\}$
(01)	$\{0\}\{1\} = \{01\}$
$(0+1)$	$\{0\} \cup \{1\} = \{0,1\}$
$((0+1)(01))$	$\{0,1\}\{01\} = \{001, 101\}$
0^*	$\{0\}^* = \{0^n \mid n \geq 0\}$
$(0+1)^*$	$\{0,1\}^* = \Sigma^*$
$(((0+1)^*1)(0+1))$	$\{0,1\}^*\{1\}\{0,1\}$

Prioridades dos operadores

Regras para omissão de parênteses:

a) Como a união é associativa, pode-se escrever $(r_1 + r_2 + \cdots + r_n)$, omitindo-se os parênteses internos.

b) Idem, para a concatenação.

c) Os parênteses externos podem ser omitidos.

d) Fecho de Kleene tem precedência sobre união e concatenação; concatenação tem precedência sobre união.

Algumas equivalências

1.	$r + s = s + r$	11.	$r^{**} = r^*$
2.	$r + \emptyset = r$	12.	$r^* = (rr)^*(\lambda + r)$
3.	$r + r = r$	13.	$\emptyset^* = \lambda$
4.	$r\lambda = \lambda r = r$	14.	$\lambda^* = \lambda$
5.	$r\emptyset = \emptyset r = \emptyset$	15.	$r^* r^* = r^*$
6.	$(r + s)t = rt + st$	16.	$rr^* = r^* r$
7.	$r(s + t) = rs + rt$	17.	$(r^* + s)^* = (r + s)^*$
8.	$(r + s)^* = (r^* s)^* r^*$	18.	$(r^* s^*)^* = (r + s)^*$
9.	$(r + s)^* = r^*(sr^*)^*$	19.	$r^*(r + s)^* = (r + s)^*$
10.	$(rs)^* = \lambda + r(sr)^* s$	20.	$(r + s)^* r^* = (r + s)^*$

Algumas observações sobre a tabela

- Qualquer equivalência que não envolva fecho de Kleene pode ser derivada a partir de 1 a 7 mais as propriedades de associatividade da união e da concatenação.

Algumas observações sobre a tabela

- Qualquer equivalência que não envolva fecho de Kleene pode ser derivada a partir de 1 a 7 mais as propriedades de associatividade da união e da concatenação.
- Com o fecho de Kleene, não há um conjunto finito de equivalências a partir das quais se possa derivar qualquer outra.

Algumas observações sobre a tabela

- Qualquer equivalência que não envolva fecho de Kleene pode ser derivada a partir de 1 a 7 mais as propriedades de associatividade da união e da concatenação.
- Com o fecho de Kleene, não há um conjunto finito de equivalências a partir das quais se possa derivar qualquer outra.
- Algumas equivalências são redundantes. Por exemplo, a 13 pode ser obtida de 2, 5 e 10:

$$\begin{aligned} \emptyset^* &= (r\emptyset)^* && \text{por 5} \\ &= \lambda + r(\emptyset r)^*\emptyset && \text{por 10} \\ &= \lambda + \emptyset && \text{por 5} \\ &= \lambda && \text{por 2} \end{aligned}$$

Simplificação de expressões regulares

$$
\begin{aligned}
\underline{(00^* + 10^*)}0^*(1^* + 0)^* &= (0 + 1)\underline{0^*0^*}(1^* + 0)^* && \text{por 6} \\
&= (0 + 1)0^*\underline{(1^* + 0)^*} && \text{por 15} \\
&= (0 + 1)0^*\underline{(1 + 0)^*} && \text{por 17} \\
&= (0 + 1)\underline{0^*(0 + 1)^*} && \text{por 1} \\
&= (0 + 1)(0 + 1)^* && \text{por 19}
\end{aligned}
$$

Simplificação de expressões regulares

$$
\begin{aligned}
\underline{(00^* + 10^*)}0^*(1^* + 0)^* &= (0 + 1)\underline{0^*0^*}(1^* + 0)^* && \text{por 6} \\
&= (0 + 1)0^*\underline{(1^* + 0)^*} && \text{por 15} \\
&= (0 + 1)0^*\underline{(1 + 0)^*} && \text{por 17} \\
&= (0 + 1)\underline{0^*(0 + 1)^*} && \text{por 1} \\
&= (0 + 1)(0 + 1)^* && \text{por 19}
\end{aligned}
$$

Diga por que:

- $(r + rr + rrr + rrrr)^* = r^*$
- $((0(0 + 1)1 + 11)0^*(00 + 11))^*(0 + 1)^* = (0 + 1)^*$
- $r^*(r + s^*) = r^*s^*$

Notações úteis

Notação

- r^+ *significa* (rr^*)
- r^n, $n \geq 0$ *é assim definida, recursivamente:*
  - *a)* $r^0 = \lambda$;
  - *b)* $r^n = rr^{n-1}$, *para* $n \geq 1$.

Notações úteis

Notação

- r^+ *significa* (rr^*)
- r^n, $n \geq 0$ *é assim definida, recursivamente:*
  - *a)* $r^0 = \lambda$;
  - *b)* $r^n = rr^{n-1}$, *para* $n \geq 1$.

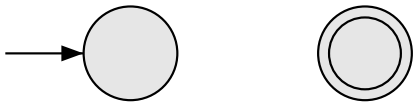
Exemplos

$(0+1)^{10}$

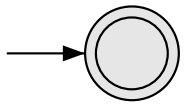
$r^* = (r^n)^*(\lambda + r + r^2 + \cdots + r^{n-1})$ *para* $n > 1$

Obtendo AF a partir de expressão regular

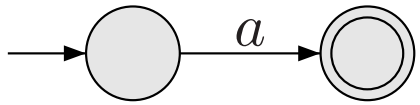
Toda expressão regular denota uma linguagem regular.



(a) AF para $\emptyset$



(b) AF para $\{\lambda\}$



(c) AF para $\{a\}$

$\Rightarrow$ Como seriam AFs para $L_1 \cup L_2$, $L_1 \cap L_2$ e L^*?

Diagrama ER

Definição

Um diagrama ER sobre Σ é um diagrama de estados cujas arestas, em vez de serem rotuladas com símbolos do alfabeto Σ, são rotuladas com ERs sobre Σ. Além disso, há no máximo uma aresta de um estado para outro em um diagrama ER.

Diagrama ER

Definição

Um diagrama ER sobre Σ é um diagrama de estados cujas arestas, em vez de serem rotuladas com símbolos do alfabeto Σ, são rotuladas com ERs sobre Σ. Além disso, há no máximo uma aresta de um estado para outro em um diagrama ER.

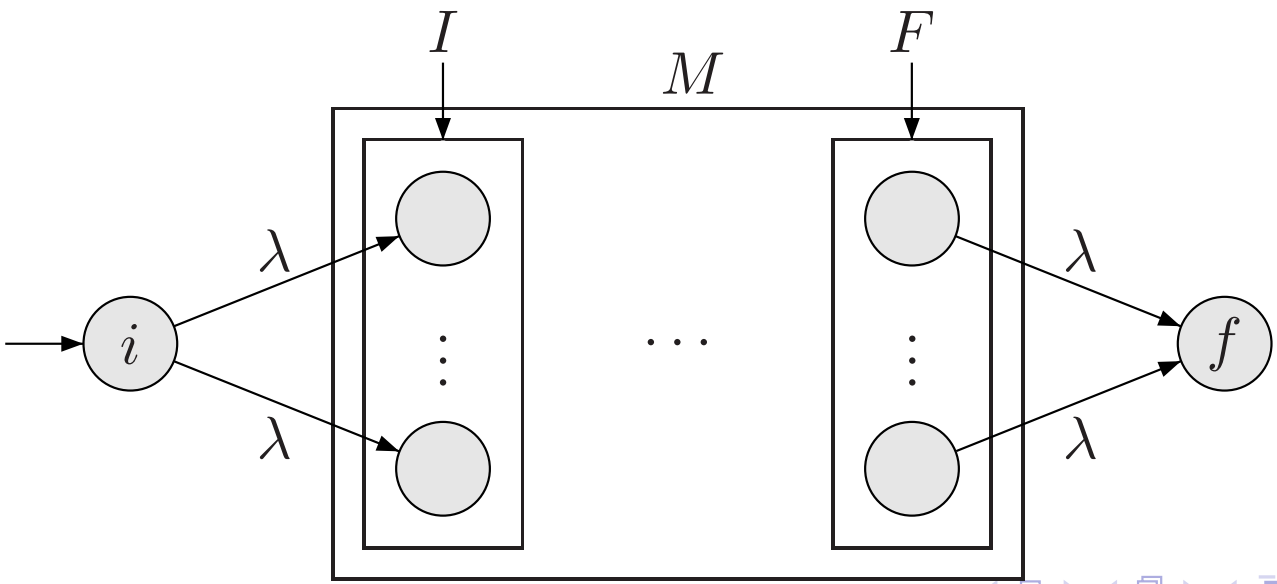
$\Rightarrow$ Uma transição sob ER r representa uma submáquina que lê qualquer palavra de $L(r)$.

Obtendo expressões regulares a partir de AFs

Toda linguagem regular é denotada por alguma expressão regular.

Seja um AFN $M = (E, \Sigma, \delta, I, F)$.

1. Obtenha AFNλ $M' = (E', \Sigma, \delta, \{i\}, \{f\})$ equivalente a M:

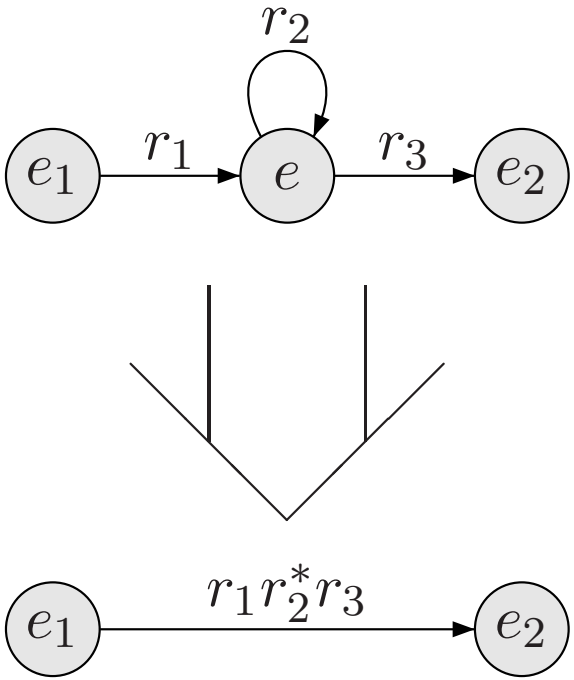


Obtendo expressões regulares a partir de AFs

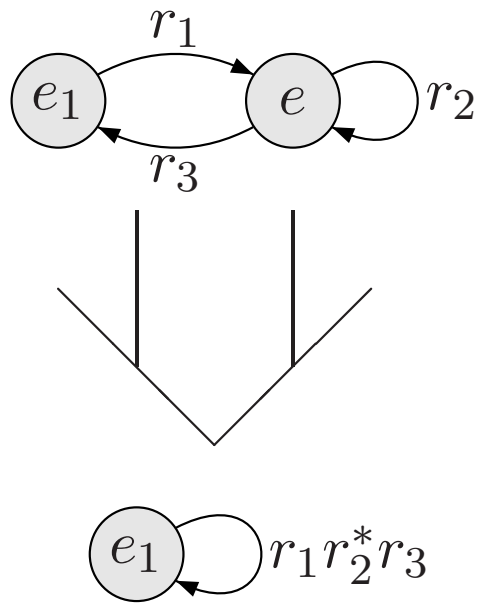
2. Construa um diagrama ER assim:
   - os estados de M' são os estados no diagrama ER;
   - i é o (único) estado inicial e f o (único) estado final no diagrama ER;
   - para cada par de estados $[e_1, e_2]$ em M', sejam $a_1, \ldots, a_n$ todos os elementos em Σ_λ tais que de e_1 há transição para e_2 sob a_i $(1 \leq i \leq n)$; há uma única transição de e_1 para e_2 no diagrama, sob a ER $a_1 + \cdots + a_n$.

Obtendo expressões regulares a partir de AFs

3. Elimine um a um os estados do diagrama ER, menos i e f:



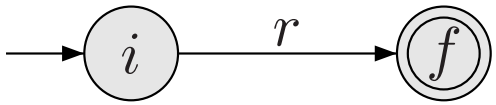
(a) $e_2 \neq e_1$



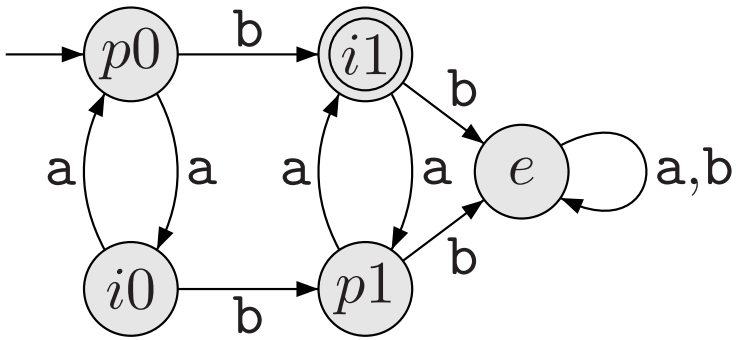
(b) $e_2 = e_1$

Obtendo expressões regulares a partir de AFs

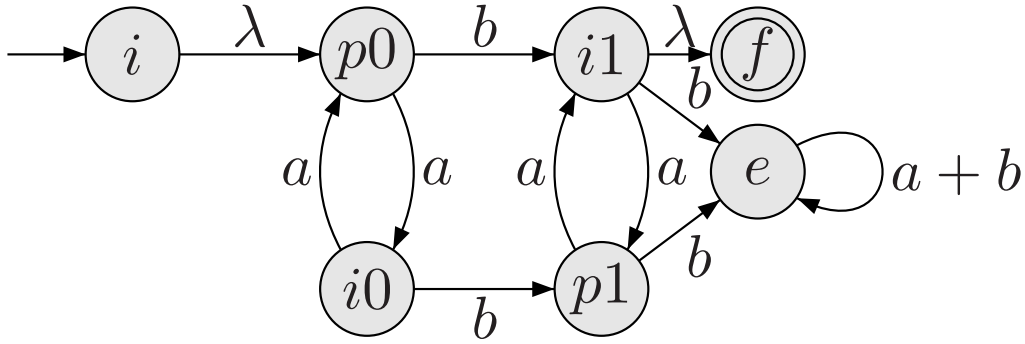
4. Após eliminados os estados em $E - \{i, f\}$, chega-se ao diagrama ER a seguir. A ER procurada é, então, r.



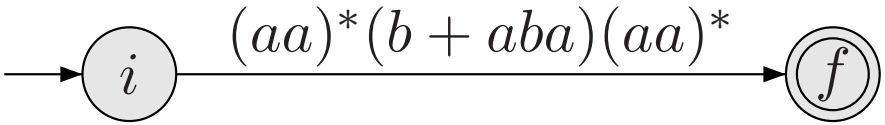
Exemplo



(a) Diagrama de estados



(b) Diagrama ER



Resultado final

Como obter ERs mais simples

$\Rightarrow$ A simplicidade da ER obtida depende da ordem de eliminação dos estados do diagrama ER.

Como obter ERs mais simples

$\Rightarrow$ A simplicidade da ER obtida depende da ordem de eliminação dos estados do diagrama ER.

Definição

$\#p(e)$ é o número de pares $[e_1, e_2]$, ambos diferentes de e, tais que há transição de e_1 para e e transição de e para e_2.

Como obter ERs mais simples

$\Rightarrow$ A simplicidade da ER obtida depende da ordem de eliminação dos estados do diagrama ER.

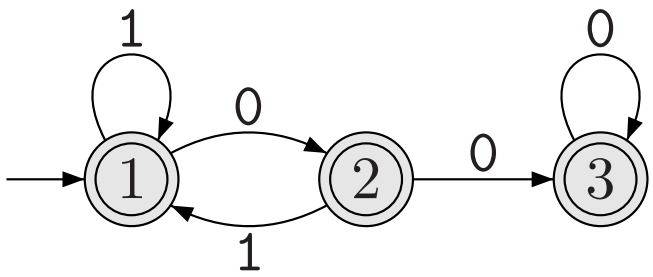
Definição

$\#p(e)$ é o número de pares $[e_1, e_2]$, ambos diferentes de e, tais que há transição de e_1 para e e transição de e para e_2.

Heurística

Eliminar um estado e com menor $\#p(e)$.

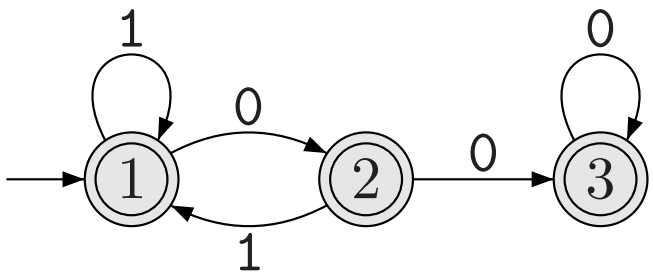
Exemplo



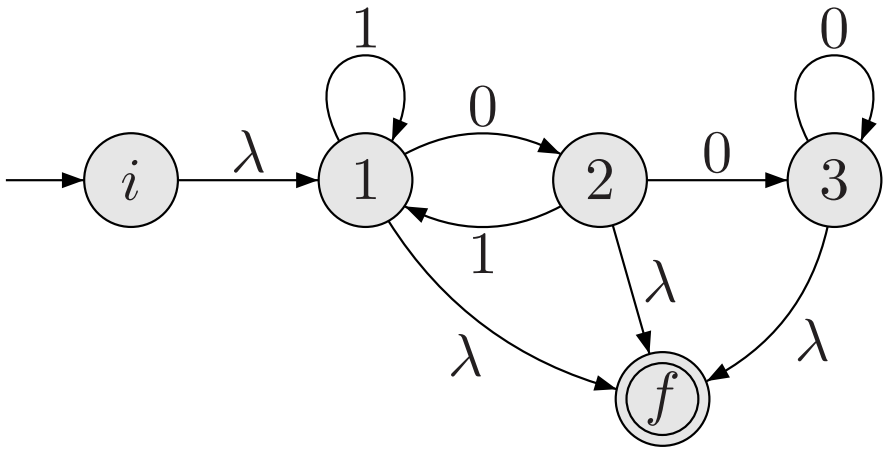
(a) Diagrama de estados

(b) Diagrama ER

Exemplo



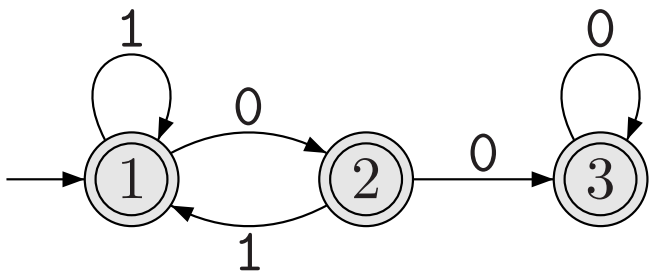
(a) Diagrama de estados



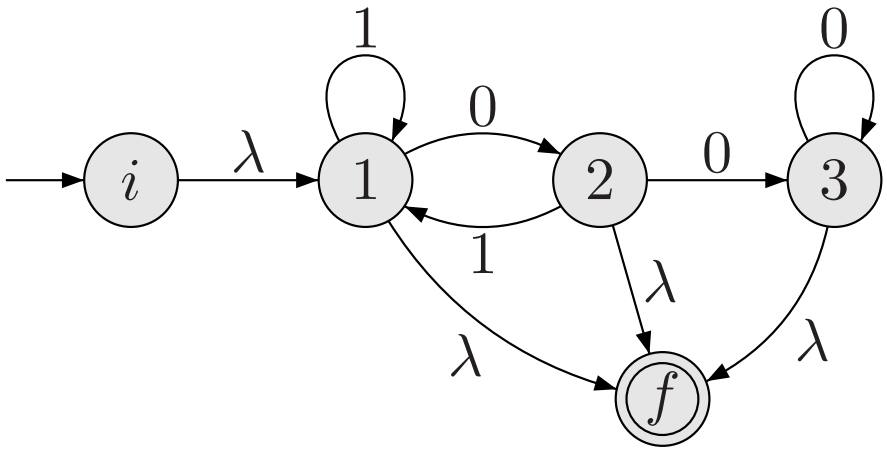
(b) Diagrama ER

(c) Eliminando 3

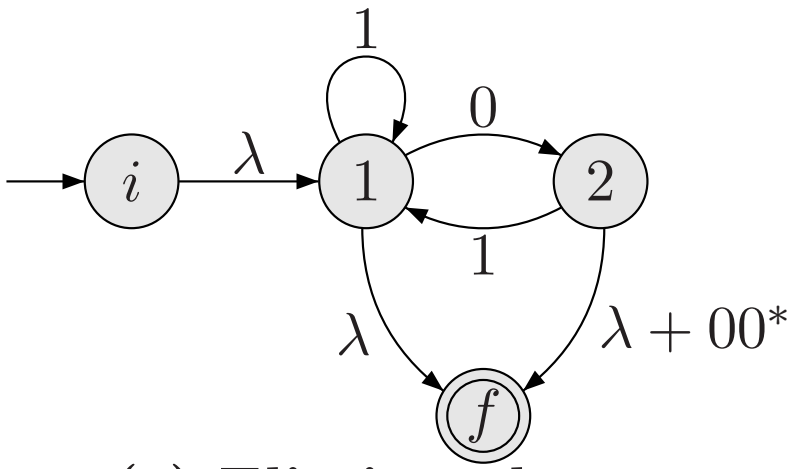
Exemplo



(a) Diagrama de estados



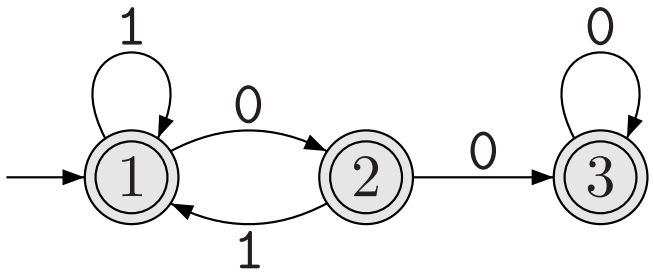
(b) Diagrama ER



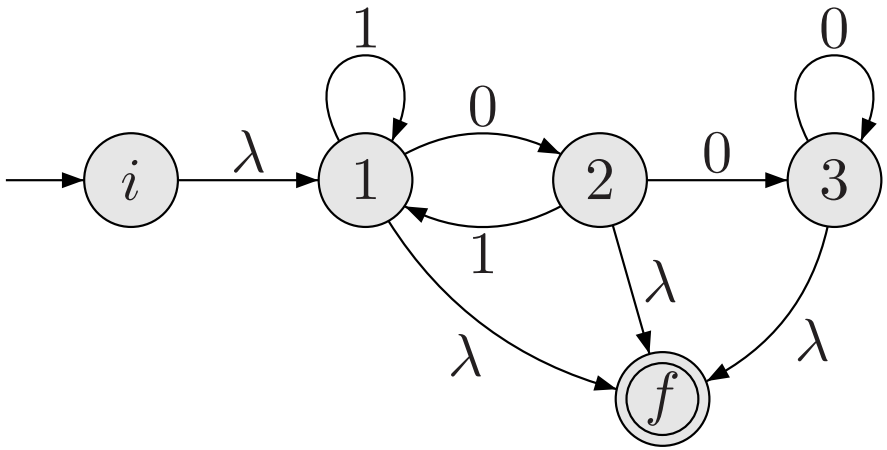
(c) Eliminando 3

(d) Eliminando 2

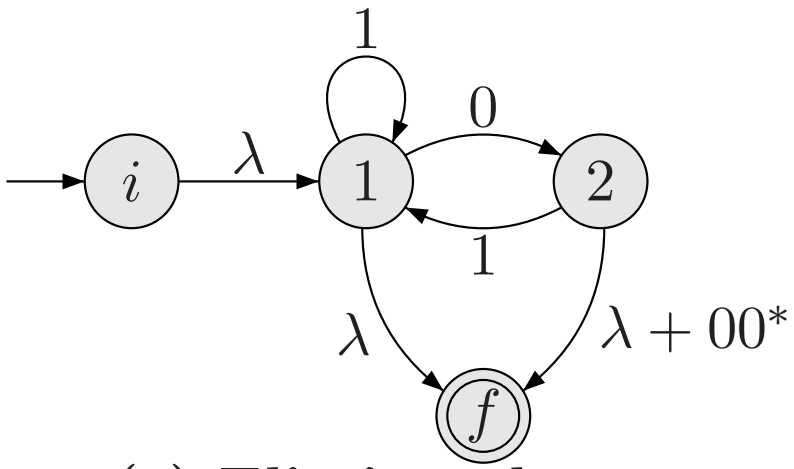
Exemplo



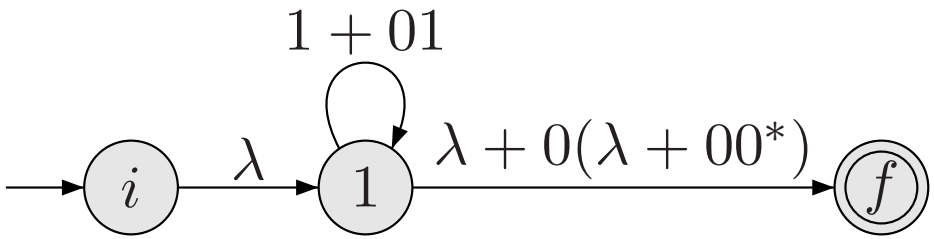
(a) Diagrama de estados



(b) Diagrama ER



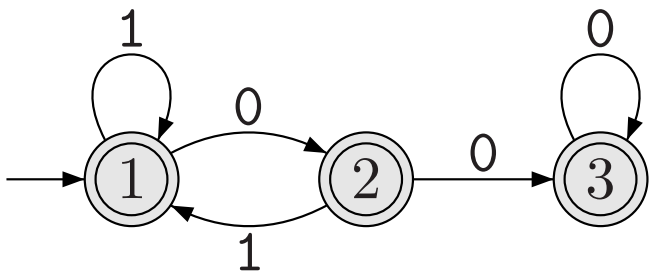
(c) Eliminando 3



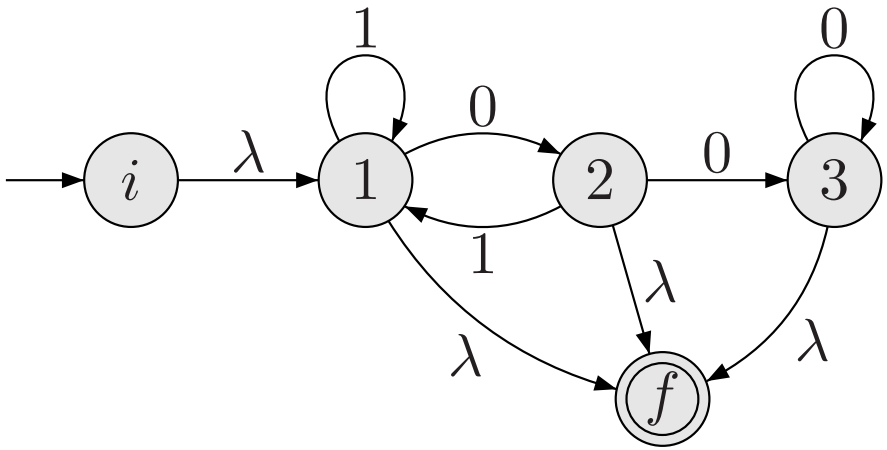
(d) Eliminando 2

e) Eliminando 1

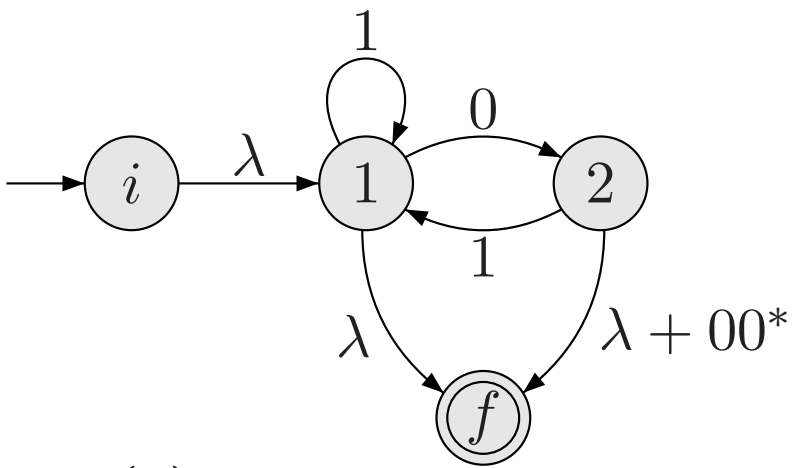
Exemplo



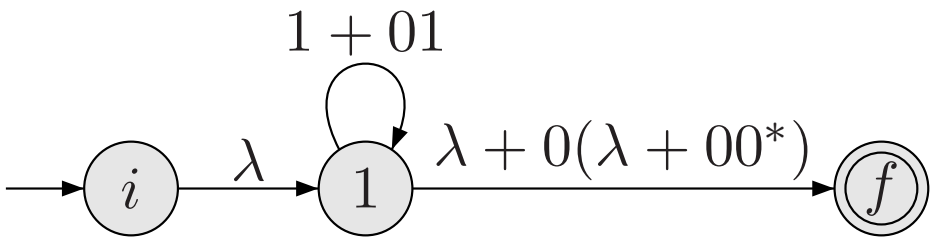
(a) Diagrama de estados



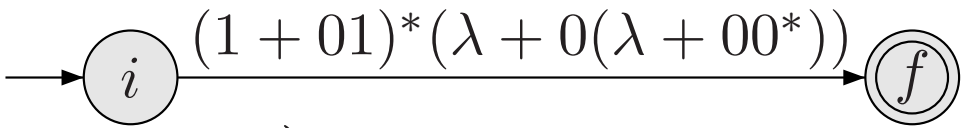
(b) Diagrama ER



(c) Eliminando 3



(d) Eliminando 2



e) Eliminando 1

O que é uma gramática regular

Definição

Uma gramática regular (GR) é uma gramática (V, Σ, R, P)*, em que cada regra tem uma das formas:*

- $X \to a$
- $X \to aY$
- $X \to \lambda$

em que $X, Y \in V$ *e* $a \in \Sigma$.

O que é uma gramática regular

Definição

Uma gramática regular (GR) é uma gramática (V, Σ, R, P)*, em que cada regra tem uma das formas:*

- $X \to a$
- $X \to aY$
- $X \to \lambda$

em que $X, Y \in V$ *e* $a \in \Sigma$.

$\Rightarrow$ Formato das formas sentenciais: wA, $w \in \Sigma^+$, $A \in V$.

Exemplo

$$L = \{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid w \text{ não contém } \mathtt{abc}\}$$

Exemplo

$L = \{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid w \text{ não contém } \mathtt{abc}\}$

Uma GR que gera L: $(\{A, B, C\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, R, A)$, em que R contém:

$$A \rightarrow \mathtt{a}B \mid \mathtt{b}A \mid \mathtt{c}A \mid \lambda$$
$$B \rightarrow \mathtt{a}B \mid \mathtt{b}C \mid \mathtt{c}A \mid \lambda$$
$$C \rightarrow \mathtt{a}B \mid \mathtt{b}A \mid \lambda$$

Exemplo

$L = \{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid w \text{ não contém } \mathtt{abc}\}$

Uma GR que gera L: $(\{A, B, C\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, R, A)$, em que R contém:

$$A \to \mathtt{a}B \mid \mathtt{b}A \mid \mathtt{c}A \mid \lambda$$
$$B \to \mathtt{a}B \mid \mathtt{b}C \mid \mathtt{c}A \mid \lambda$$
$$C \to \mathtt{a}B \mid \mathtt{b}A \mid \lambda$$

O i-ésimo símbolo é gerado no i-ésimo passo:

$$A \Rightarrow \mathtt{a}B \Rightarrow \mathtt{aa}B \Rightarrow \mathtt{aab}C \Rightarrow \mathtt{aabb}A \Rightarrow \mathtt{aabbc}A \Rightarrow \mathtt{aabbc}$$

AF a partir de GR

Toda gramática regular gera uma linguagem regular.

AF a partir de GR

Toda gramática regular gera uma linguagem regular.

Seja $G = (V, \Sigma, R, P)$ e $Z \notin V$.
Um AFN que reconhece L(G): $M = (E, \Sigma, \delta, \{P\}, F)$, em que

- $E = \begin{cases} V \cup \{Z\} & \text{se } R \text{ contém regra da forma } X \to a \\ V & \text{caso contrário.} \end{cases}$
- Para toda regra da forma:
  - $X \to aY$ faça $Y \in \delta(X, a)$
  - $X \to a$ faça $Z \in \delta(X, a)$
- $F = \begin{cases} \{X | X \to \lambda \in R\} \cup \{Z\} & \text{se } Z \in E \\ \{X | X \to \lambda \in R\} & \text{caso contrário.} \end{cases}$

Exemplo

GR G:

$$A \to 0A \mid 1B \mid 0$$
$$B \to 1B \mid \lambda$$

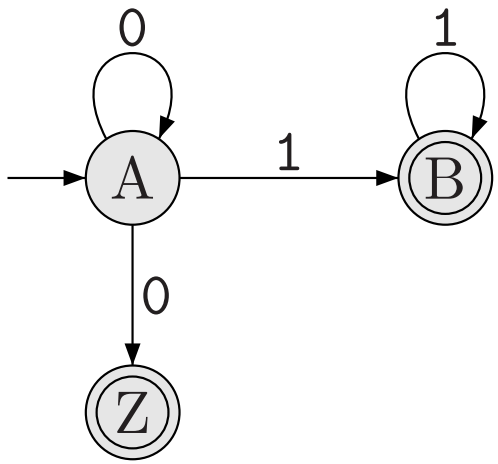
$L(G) = 0^*(0 + 1^+)$. AFN para $L(G)$:

Exemplo

GR G:

$$A \to 0A \mid 1B \mid 0$$
$$B \to 1B \mid \lambda$$

$L(G) = 0^*(0 + 1^+)$. AFN para $L(G)$:



GR a partir de AF

Toda linguagem regular é gerada por gramática regular.

Seja um AFN $M = (E, \Sigma, \delta, \{i\}, F)$.
A GR (E, Σ, R, i) gera $L(M)$, em que:

$$R = \{e \to ae' \mid e' \in \delta(e, a)\} \cup \{e \to \lambda \mid e \in F\}.$$

Exemplo

A GR

$$A \to 0A \mid 0Z \mid 1B$$
$$B \to 1B \mid \lambda$$
$$Z \to \lambda$$

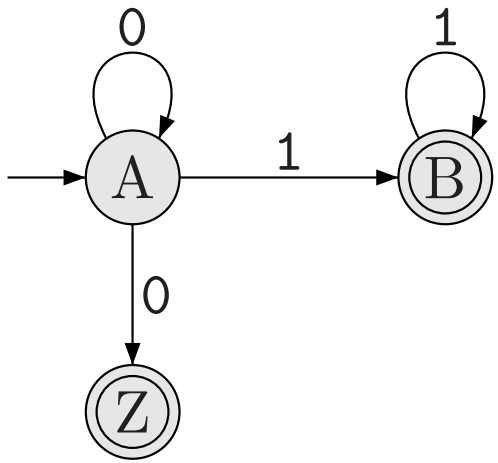
é obtida de:

Exemplo

A GR

$$A \to 0A \mid 0Z \mid 1B$$
$$B \to 1B \mid \lambda$$
$$Z \to \lambda$$

é obtida de:



Recapitulando

Reconhecedores de linguagens regulares: AFDs, AFNs e AFNλ.

- Cada um pode ser mais conveniente que os outros em situações específicas.
- Eles têm o mesmo poder computacional (não determinismo não aumenta o poder computacional).
- Existe algoritmo para obter reconhecedor de um tipo a partir de reconhecedor de outro.

Recapitulando

Denotação de linguagens regulares: expressões regulares.

- Existe algoritmo para obter AF a partir de ER, e vice-versa, para obter ER a partir de AF.

Recapitulando

Denotação de linguagens regulares: expressões regulares.

- Existe algoritmo para obter AF a partir de ER, e vice-versa, para obter ER a partir de AF.

Geração de linguagens regulares: gramáticas regulares.

- Existe algoritmo para obter AF a partir de GR, e vice-versa, para obter GR a partir de AF.

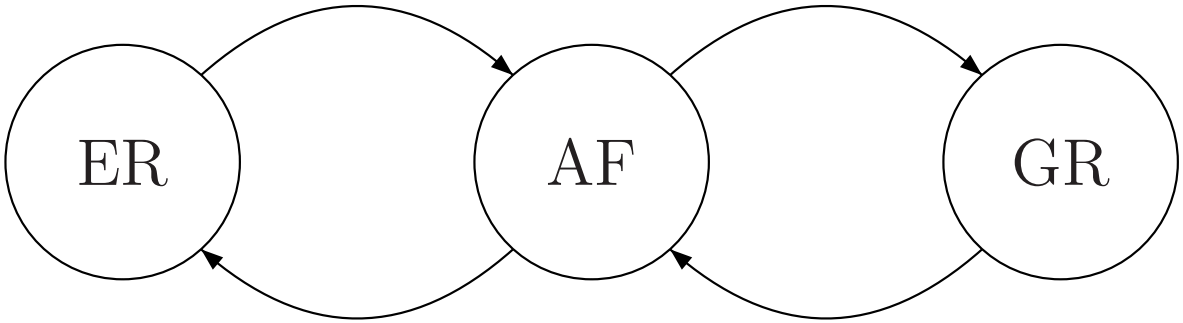
Uma síntese

$\Rightarrow$ ERs, GRs e AFs são formalismos alternativos para linguagens regulares.

Uma síntese

$\Rightarrow$ ERs, GRs e AFs são formalismos alternativos para linguagens regulares.

Transformações entre formalismos:



Linguagens regulares são fechadas sob muitas operações.

- Exemplos: união, interseção, complemento, concatenação e fecho de Kleene. Nos exercícios: reverso, homorfismo, substituição, quociente etc.
- Arsenal importante para provar que linguagens são ou não regulares.
- Importantes na obtenção de AFs, ERs e/ou GRs por meio de decomposição.

Linguagens regulares são fechadas sob muitas operações.

- Exemplos: união, interseção, complemento, concatenação e fecho de Kleene. Nos exercícios: reverso, homorfismo, substituição, quociente etc.
- Arsenal importante para provar que linguagens são ou não regulares.
- Importantes na obtenção de AFs, ERs e/ou GRs por meio de decomposição.

Problemas associados são normalmente decidíveis.

- Exemplos: determinar se L é vazia, se $L = \Sigma^*$, se L é finita etc., dada L em qualquer dos formalismos.

Linguagens regulares são fechadas sob muitas operações.

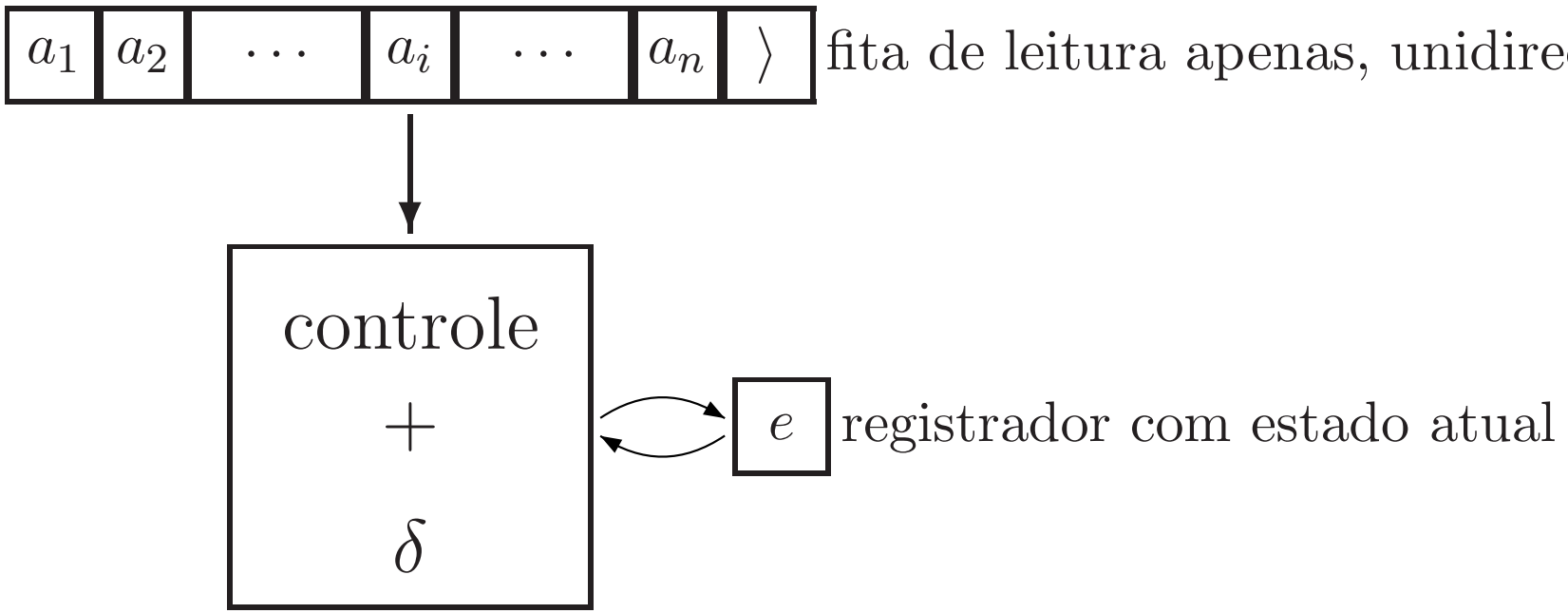
- Exemplos: união, interseção, complemento, concatenação e fecho de Kleene. Nos exercícios: reverso, homorfismo, substituição, quociente etc.
- Arsenal importante para provar que linguagens são ou não regulares.
- Importantes na obtenção de AFs, ERs e/ou GRs por meio de decomposição.

Problemas associados são normalmente decidíveis.

- Exemplos: determinar se L é vazia, se $L = \Sigma^*$, se L é finita etc., dada L em qualquer dos formalismos.

Há enorme diversidade de aplicações para linguagens regulares.

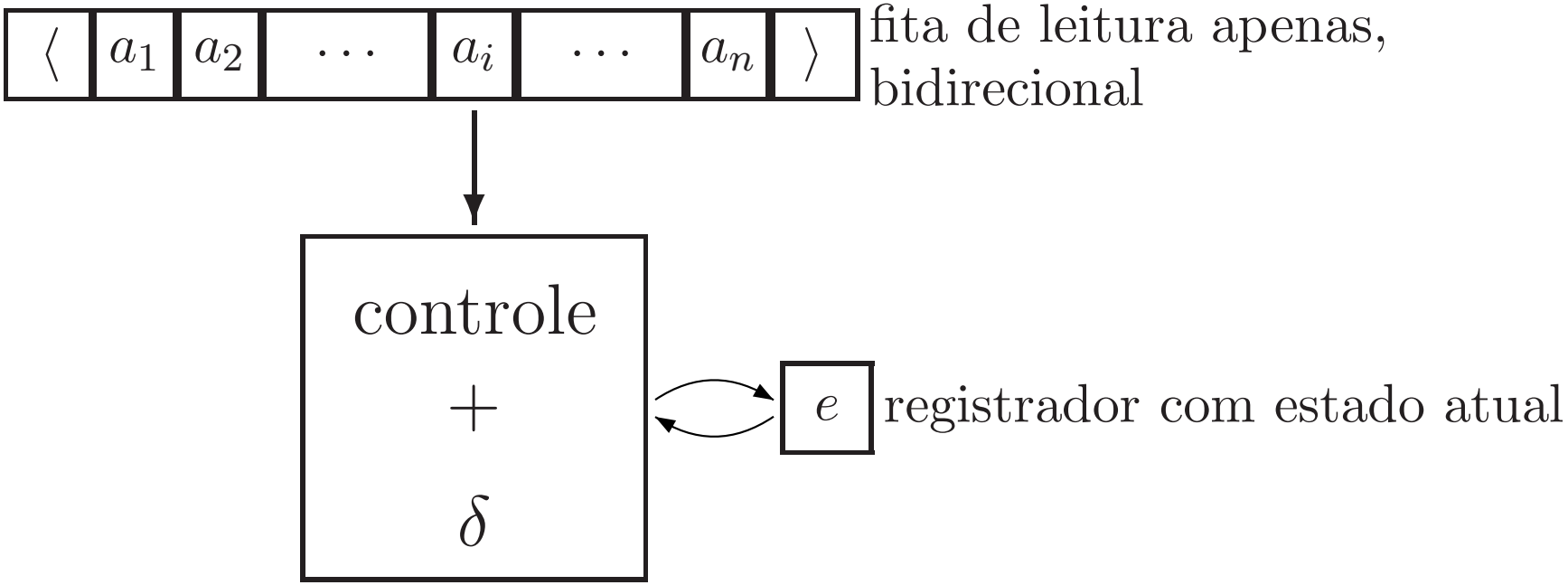
AFD visto como um computador



AFD com fita bidirecional

Função de transição parcial $\delta : E \times \Sigma \to E \times \{\mathtt{e}, \mathtt{d}\}$.

$L(M)$: toda $w \in \Sigma^*$ para a qual M para em um estado final.



Expressividade de AFD com fita bidirecional

- AFDs bidirecionais reconhecem exatamente as linguagens regulares.
- Mesmo com não determinismo, um AF bidirecional reconhece apenas linguagens regulares.
- Que incrementos poderiam aumentar o poder computacional?