Capítulo 2: Máquinas de Estados Finitos

Newton José Vieira Isabel Gomes Barbosa

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

19 de agosto de 2010

Sumário

Quebra-cabeças

O Leão, o coelho e o 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.

Modelagem do problema

Na fase de modelagem:

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

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

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

Modelagem do quebra-cabeças

Informações relevantes abstraídas para o quebra-cabeças:

a) em um dado instante, em que margem do rio estão o homem, o leão, o coelho, e o repolho; representação;

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

Representação das informações:

a) $A \subseteq \{h, l, c, r\}$ representa os elementos em A estão na margem esquerda e os em $\{h, l, c, r\} - A$ estão na direita;

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.

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-cabeças:

a) Estado: um subconjunto de $\{h, l, c, r\}$
inicial: $\{h, l, c, r\}$;
final: $\{\}$.

b) Transição: provocada por uma das ações: `s`, `l`, `c` ou `r`.

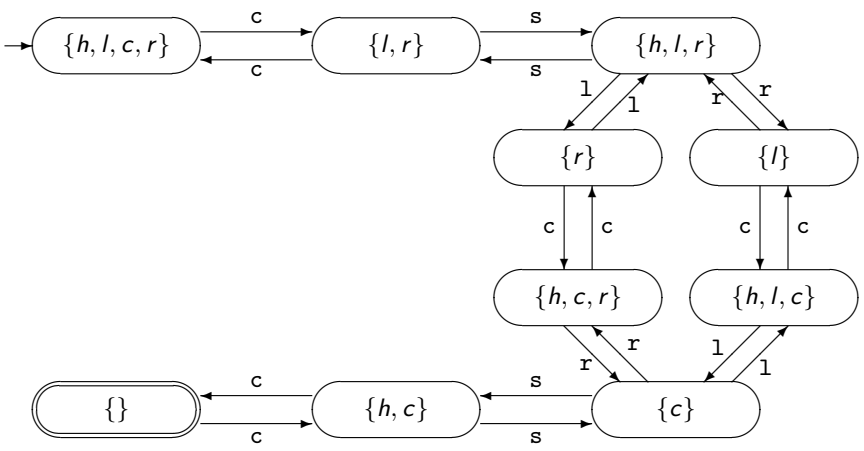
c) Solução: uma palavra no alfabeto $\{\mathtt{s}, \mathtt{l}, \mathtt{c}, \mathtt{r}\}$

Diagrama de estados

Representação do espaço de 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 `s`: aresta de e para e' com rótulo `s`.

Diagrama de estados para Leão-coelho-repolho



▸ Formalização

Computação

Dada $w \in \{\mathtt{s}, \mathtt{l}, \mathtt{c}, \mathtt{r}\}^*$:

- w é **reconhecida** se o "caminho correspondente" vai do estado inicial ao final.
- w é **rejeitada**, caso contrário.

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, $w = xy$).

Relação $\vdash$: $[e_1, w] \vdash [e_2, y]$ se existe uma transição de e_1 para e_2 sob a e $w = ay$.
Exemplo de **computação**:

$$[\{l, r\}, \mathtt{sllr}] \vdash [\{h, l, r\}, \mathtt{llr}] \vdash [\{r\}, \mathtt{lr}] \vdash [\{h, l, r\}, \mathtt{r}] \vdash [\{l\}, \lambda].$$

Características do autômato do quebra-cabeças

- 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.

Probleminha de matemática

Binário divisível por 6

Seja o problema de 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.

Projetando o autômato

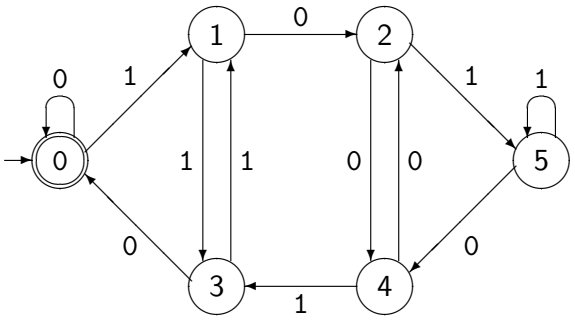
Se $\eta(x) \bmod 6 = r$, então:

- $\eta(x0) \bmod 6 = 2r \bmod 6$; e
- $\eta(x1) \bmod 6 = (2r + 1) \bmod 6$.

Segue-se um autômato com as características:

- um estado para cada resto de 0 a 5;
- do estado correspondente ao resto r emanam duas transições:
  - transição sob 0 para o estado correspondente a $2r \bmod 6$, e
  - transição sob 1 para o estado correspondente a $(2r+1) \bmod 6$;
- o estado correspondente a resto 0 é o estado inicial;
- o estado correspondente a resto 0 é estado final.

Diagrama de estados



▸ Formalização

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 para 3 andares

As 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 de 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.

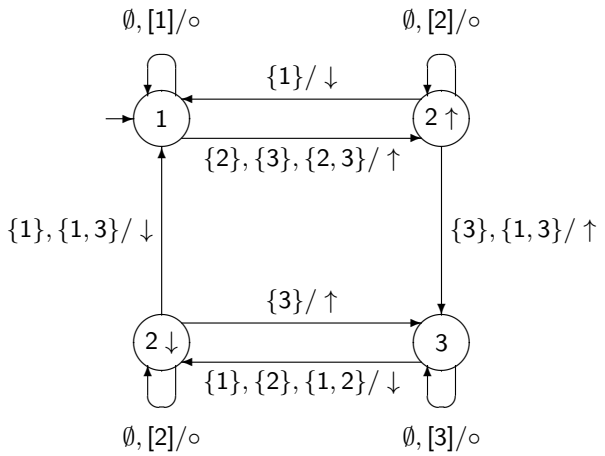
Componentes do autômato

Alguns 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 o 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.

Exemplo de computação:

$$[1, \{2,3\}\{1,3\}\{1\}, \lambda] \vdash [2\uparrow, \{1,3\}\{1\}, \uparrow] \vdash [3, \{1\}, \uparrow\uparrow] \vdash [2\downarrow, \lambda, \uparrow\uparrow\downarrow].$$

(Apesar do último conjunto de chamadas ser $\{1\}$, o elevador não vai até o primeiro andar, porque "alguém" desligou manualmente o chamado depois que o elevador partiu do terceiro em direção ao segundo andar.)

O que é autômato finito determinístico

AFD

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

$$M = (E, \{\mathtt{s}, \mathtt{l}, \mathtt{c}, \mathtt{r}\}, \delta, \{h, l, c, r\}, \{\{\}\})$$

$$E = \{\{h, l, c, r\}, \{l, r\}, \{h, l, r\}, \{l\}, \{r\}, \{h, l, c\}, \{h, c, r\}, \{c\}, \{h, c\}, \{\}, t\}$$

δ	s	l	c	r
$\{h, l, c, r\}$	t	t	$\{l, r\}$	t
$\{l, r\}$	$\{h, l, r\}$	t	$\{h, l, c, r\}$	t
$\{h, l, r\}$	$\{l, r\}$	$\{r\}$	t	$\{l\}$
$\{l\}$	t	t	$\{h, l, c\}$	$\{h, l, r\}$
$\{r\}$	t	$\{h, l, r\}$	$\{h, c, r\}$	t
$\{h, l, c\}$	t	$\{c\}$	$\{l\}$	t
$\{h, c, r\}$	t	t	$\{r\}$	$\{c\}$
$\{c\}$	$\{h, c\}$	$\{h, l, c\}$	t	$\{h, c, r\}$
$\{h, c\}$	$\{c\}$	t	$\{\}$	t
$\{\}$	t	t	$\{h, c\}$	t
t	t	t	t	t

▸ Diagrama de Estados

Uma convenção em diagramas de estados

Se não há transição de e sob a no diagrama de estados, então existe e' (estado de **erro**) 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.

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

AFD para o probleminha de matemática

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

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

▸ Diagrama de Estados

Função de transição estendida

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 como segue:

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)

Linguagem reconhecida por um AFD

A linguagem reconhecida por um AFD

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

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

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

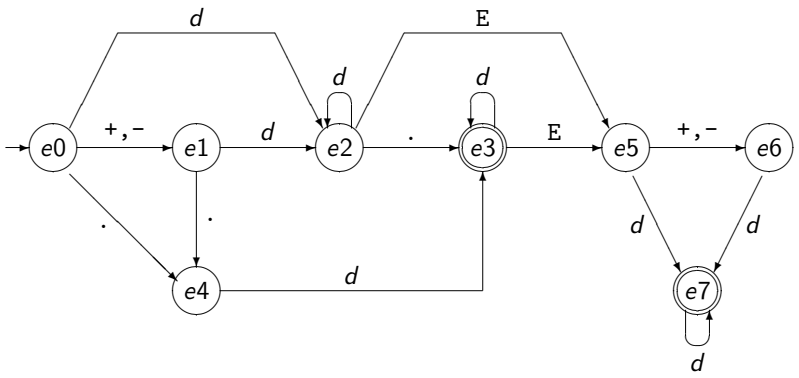
Algoritmo para simular AFDs

Algoritmo

Entrada: 1. o AFD, dado por i, F e D, e
2. a palavra de entrada, dada por *prox*.
Saída: *sim* ou *não*.
$e \leftarrow i$; $s \leftarrow prox()$;
enquanto $s \neq fs$ **faça**
  $e \leftarrow D[e, s]$; $s \leftarrow prox()$;
fimenquanto;
se $e \in F$ **então**
  retorne *sim*
senão
  retorne *não*
fimse

Desempenho: $O(n)$ (n: número de símbolos da palavra).

Uma aplicação: Análise Léxica



Análise Léxica

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

- $E = \{e0, e1, e2, e3, e4, e5, e6, e7, ee\}$
- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., +, -, \mathtt{E}\}$

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

Mais exemplos

Exemplos: AFDs que reconheçam:

- $\{w \in \{0,1\}^* \mid w$ tem número par de símbolos$\}$.
- $\{w \in \{0,1\}^* \mid w$ tem número par de 0s e número par de 1s$\}$.

AFDs equivalentes

Dois AFDs, M_1 e M_2, são ditos equivalentes se, e somente se, $L(M_1) = L(M_2)$.

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!!!

O que é AFD mínimo

AFD mínimo

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

Estados equivalentes

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

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

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 $e \not\approx e'$ (existe $y \in \Sigma^*$ $\hat{\delta}(e, y) \in F$ e $\hat{\delta}(e', y) \notin F$ ou vice-versa): 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!

O conceito de autômato reduzido

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

AFD reduzido

Seja um AFD $M = (E, \Sigma, \delta, i, F)$. Um autômato reduzido correspondente a M é o AFD $M' = (E', \Sigma, \delta', i', F')$, em que:

- $E' = \{[e] \mid e \in E\}$;
- $\delta'([e], a) = [\delta(e, a)]$ para todo $e \in E$ e $a \in \Sigma$;
- $i' = [i]$;
- $F' = \{[e] \mid e \in F\}$.

Obtenção de $\approx$ passo a passo

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

1. $e \approx_0 e'$ se, e somente se, $(e, e' \in F$ ou $e, e' \in E - F)$;
2. para $n \geq 0$: $e \approx_{n+1} e'$ se, e somente se, $e \approx_n e'$ e $\delta(e, a) \approx_n \delta(e', a)$ para todo $a \in \Sigma$.

Teorema que justifica $\approx_n$:

Seja um AFD $M = (E, \Sigma, \delta, i, F)$. Então $e \approx_n e'$ se, e somente se, para todo $w \in \Sigma^*$ tal que $|w| \leq n$, $\hat{\delta}(e, w) \in F \leftrightarrow \hat{\delta}(e', w) \in F$.

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

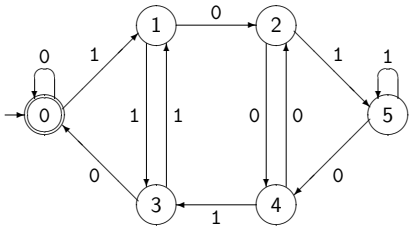
$[e]_n$: classe de equivalência de e na partição induzida por $\approx_n$.

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

Autômato inicial

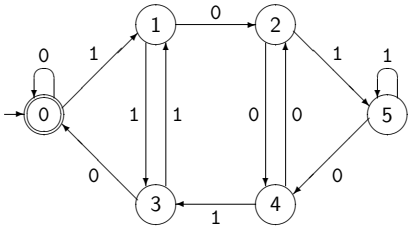


Evolução das partições

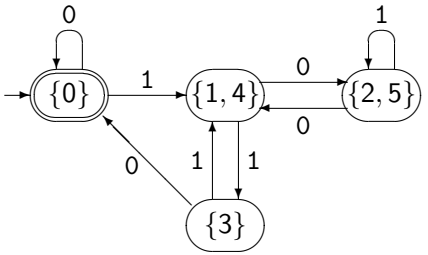
S_0: $\{0\}, \{1,2,3,4,5\}$
S_1: $\{0\}, \{1,2,4,5\}, \{3\}$
S_2: $\{0\}, \{1,4\}, \{2,5\}, \{3\}$
S_3: $\{0\}, \{1,4\}, \{2,5\}, \{3\}$

Um exemplo de minimização

Autômato inicial



Autômato reduzido



Produto de AFDs

Simulação do funcionamento **em paralelo** de dois AFDs:

- Sejam $M_1 = (E_1, \Sigma, \delta_1, i_1, F_1)$ e $M_2 = (E_2, \Sigma, \delta_2, i_2, F_2)$.
- $M_3 = (E_3, \Sigma, \delta_3, i_3, F_3)$, 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.

Sejam $e_1 \in E_1$ e $e_2 \in E_2$ de M_3. Então,

$$\hat{\delta_3}([e_1, e_2], w) = [\hat{\delta_1}(e_1, w), \hat{\delta_2}(e_2, w)], \text{ para todo } w \in \Sigma^*.$$

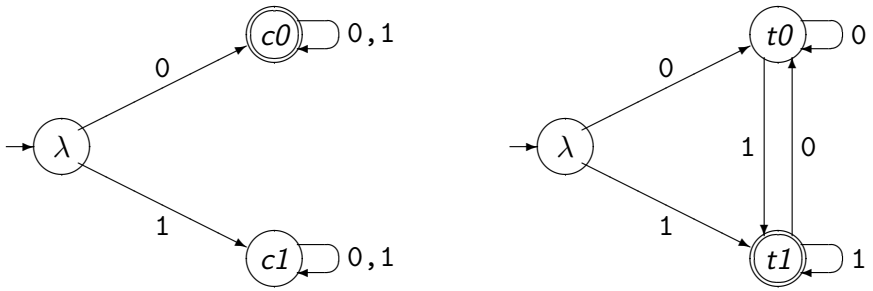
Complemento, interseção e união

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

1. AFD para $\overline{L(M_1)}$:
   $(E_1, \Sigma, \delta_1, i_1, E_1 - F_1)$.
2. $L(M_1) \cap L(M_2)$:
   produto de M_1 e M_2 com $F_3 = F_1 \times F_2$.
3. $L(M_1) \cup L(M_2)$:
   produto de M_1 e M_2 com $F_3 = (F_1 \times E_2) \cup (E_1 \times F_2)$

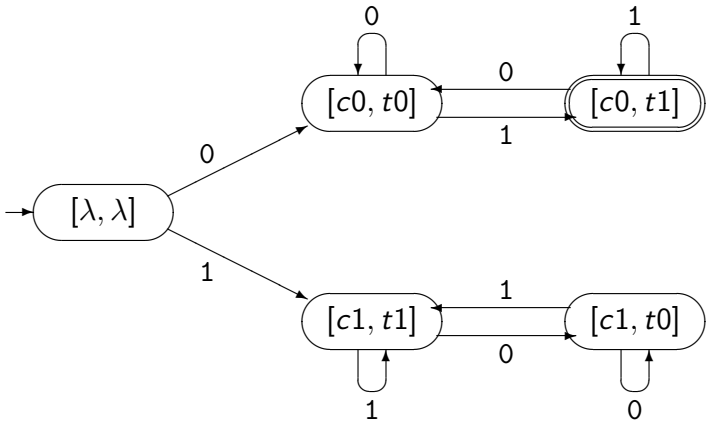
Exemplo de produto/interseção e união

Reconhecendo $\{0\}\{0,1\}^*$ e $\{0,1\}^*\{1\}$:



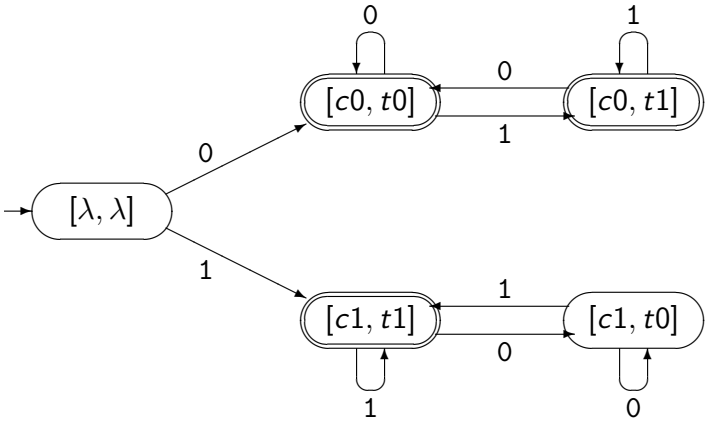
Exemplo de produto/interseção

Reconhecendo $\{0\}\{0,1\}^* \cap \{0,1\}^*\{1\}$:



Exemplo de produto/união

Reconhecendo $\{0\}\{0,1\}^* \cup \{0,1\}^*\{1\}$:



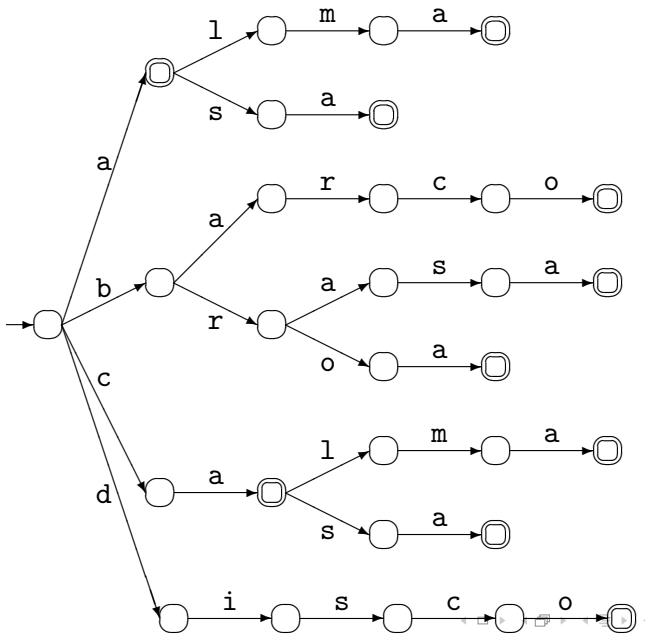
Linguagens finitas

- Para toda linguagem finita existe um AFD.
- AFD com diagrama de estados simplificado **sem ciclos**:
  árvore em que o estado inicial é a raiz.

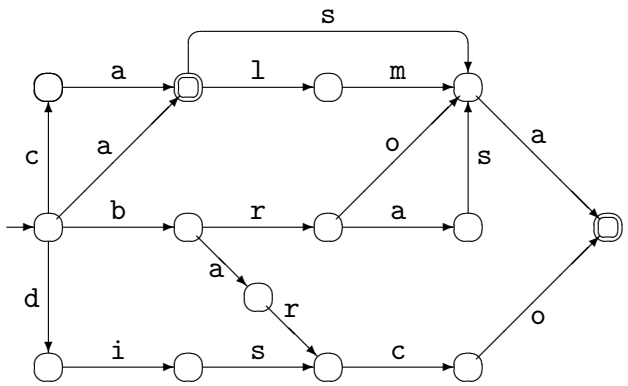
Exemplo: Um pequeno dicionário:

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

AFD para o pequeno dicionário



O AFD minimizado para o dicionário



Uma liguagem não reconhecível por AFD

$L = \{a^n b^n \mid n \geq 0\}$. Seja um AFD M para L com k estados. M passa por um ciclo quando lê o prefixo a^k de $a^k b^k$.
Seja v a subpalavra consumida ao se percorrer o ciclo. Nesse caso, $a^k b^k = uvw$, onde:

- $u = a^{n_1}$, $n_1 \geq 0$;
- $v = a^i$, $i \geq 1$;
- $w = a^{n_2} b^k$, $n_2 \geq 0$;

Como o ciclo pode ser percorrido quantas vezes se queira, $uv^i w \in L$ para todo $i \geq 0$. Mas $uv^2 w \notin L$. Contradição!
Logo, não existe AFD para L.

Um teorema útil para provar que não há AFD

A técnica empregada no Exemplo leva a um teorema:

Teorema para provar que não há AFD

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

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

Alguns problemas de decisão envolvendo AFDs

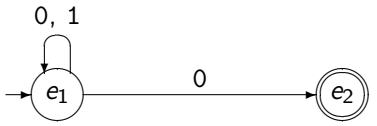
Existem procedimentos de decisão para determinar, para qualquer AFD M, se:

1. $L(M) = \emptyset$; e
2. $L(M)$ é finita.

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

1. $L(M) = \emptyset$ se, e somente se, M' não tiver estados finais;
2. $L(M)$ é finita se, e somente se, o diagrama de estados simplificado de M' não possui ciclo.

Exemplo de autômato finito não determinístico (AFN)



- Não determinismo: no estado e_1 existem **duas** transições possíveis sob o símbolo 0.
- Computações possíveis para a palavra 1010:

$$[e_1, 1010] \vdash [e_1, 010] \vdash \begin{cases} [e_1, 10] \vdash [e_1, 0] \vdash \begin{cases} [e_1, \lambda] \\ [e_2, \lambda] \end{cases} \\ [e_2, 10] \end{cases}$$

Reconhecimento para AFN

- Uma palavra é reconhecida se, e somente se, **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 no reconhecimento.

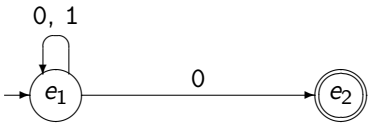
O que é AFN

Definição de AFN

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

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 recursivamente como segue:

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

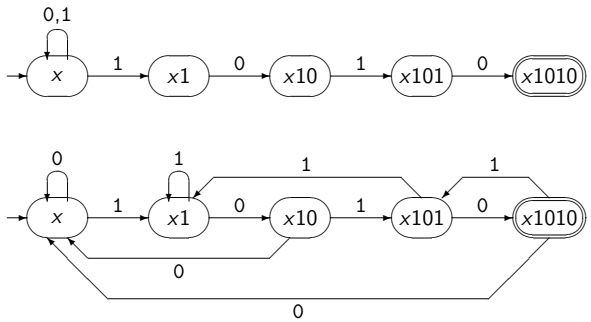
Linguagem reconhecida por um AFN

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

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

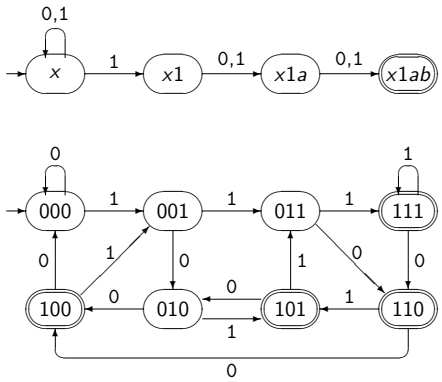
Exemplo 1: AFN $\times$ AFD

AFN e AFD que aceitam $\{0, 1\}^*\{1010\}$:



Exemplo 2: AFN $\times$ AFD

AFN e AFD que aceitam $\{w \in \{0, 1\}^* \mid |w| \geq 3$ e o terceiro símbolo da direita para a esquerda é $1\}$:



Equivalência entre AFDs e AFNs

Para qualquer AFN existe um AFD equivalente.

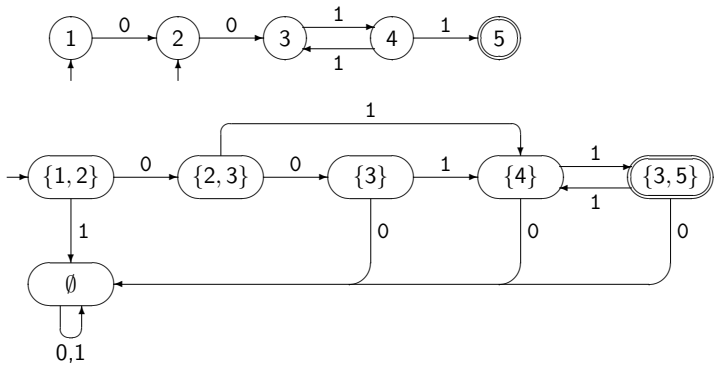
Idéia: Um estado será um conjunto, significando todos os estados do AFN atingidos por todas as computações possíveis para a mesma palavra.

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

Equivalência entre AFDs e AFNs

Diagrama de estados de um AFN e do AFD equivalente.



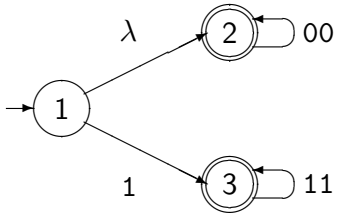
O que é AFN estendido

AFN estendido

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

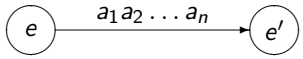
- E, Σ, I e F são como em AFNs; e
- δ é uma função parcial de $E \times D$ para $\mathcal{P}(E)$, em que D é algum subconjunto finito de Σ^*.

Exemplo: AFNE para
$\{w \in \{0\}^* \mid |w| \text{ é par}\} \cup \{w \in \{1\}^* \mid |w| \text{ é ímpar}\}$:

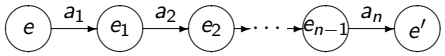


Tirando transições sob palavras de mais de um símbolo

Uma transição da forma



pode ser substituída por n transições:



AFN com transições λ

Definição de AFNλ

Um AFNλ 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 $E \times (\Sigma \cup \{\lambda\})$ para $\mathcal{P}(E)$.

A função fecho λ

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

A função **fecho** λ para M, $f\lambda$: $\mathcal{P}(E) \to \mathcal{P}(E)$, é definida recursivamente como:

- $X \subseteq f\lambda(X)$;
- se $e \in f\lambda(X)$, então $\delta(e, \lambda) \subseteq f\lambda(X)$.

A função de transição estendida

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

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

1. $\hat{\delta}(\emptyset, w) = \emptyset$, para $w \in \Sigma^*$;
2. $\hat{\delta}(A, \lambda) = f\lambda(A)$, para $A \subseteq E$;
3. $\hat{\delta}(A, ay) = \hat{\delta}(\bigcup_{e \in f\lambda(A)} \delta(e, a), y)$, para $A \subseteq E$, $a \in \Sigma$ e $y \in \Sigma^*$.

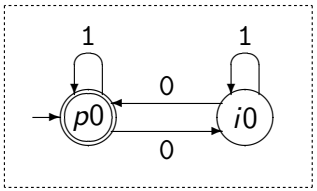
Linguagem reconhecida por um AFNλ

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

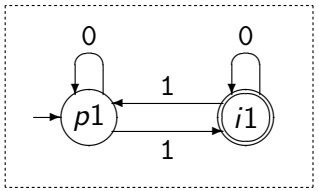
$$L(M) = \{w \in \Sigma^* \mid \hat{\delta}(I, w) \cap F \neq \emptyset\}.$$

Um exemplo de AFNλ

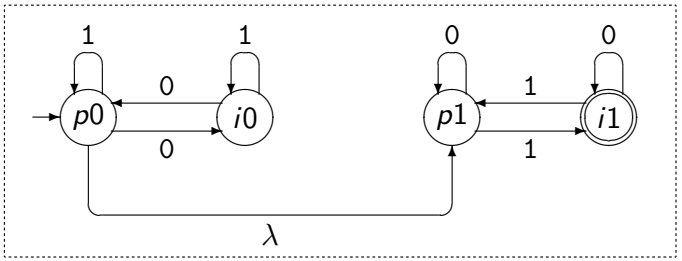
AFD para L_1



AFD para L_2



AFNλ para L_1L_2

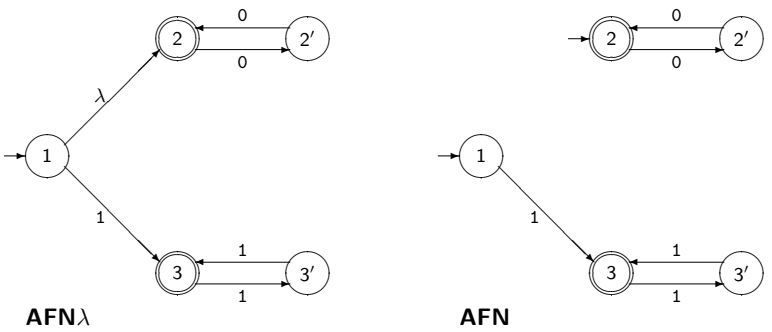


Obtenção de AFN equivalente a AFNλ

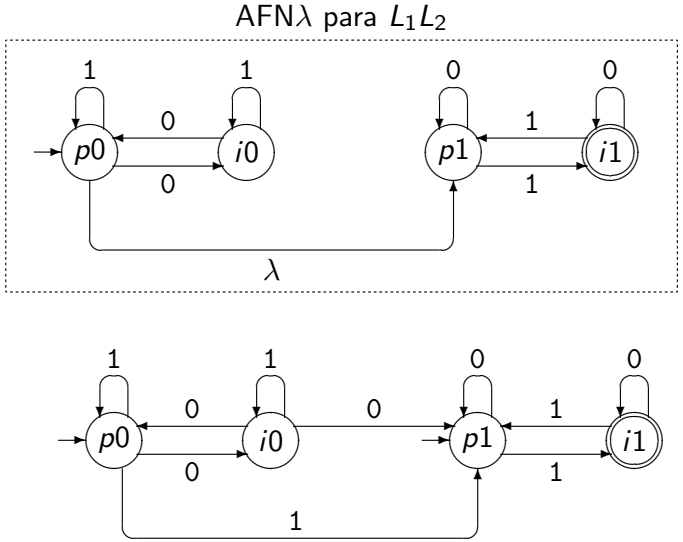
Seja um AFNλ $M = (E, \Sigma, \delta, I, F)$.
Um AFN equivalente a M é $M' = (E, \Sigma, \delta', I', F)$, em que:

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

Um exemplo



Outro exemplo



Relações entre automâtos finitos