MAC 414
Autômatos, Computabilidade e Complexidade

aula 4 — 23/09/2020

Transições λ

Transições λ

Funcionam como uma adivinhação para mudar de estado.

Transições λ

Funcionam como uma adivinhação para mudar de estado.

Notação: $\hat{\Sigma} = \Sigma \cup \{\lambda\}$

Autômatos

Definição

Um autômato não determinístico consiste de 5 componentes,

$$\mathscr{A} = (K, \Sigma, \Delta, S, F),$$

onde:

- K *é um conjunto finito, de estados.*
- Σ *é um alfabeto*
- $\Delta \subseteq K \times \hat{\Sigma} \times K$ *é o cojunto de ~~função de~~ transições*
- $S \in K$ *é o conjunto de estados iniciais*
- $F \subseteq K$ *são os estados finais*

O grafo de um AND

O grafo de um AND

$G_{\mathscr{A}}$ é um grafo dirigido, com rótulos nas arestas.

O grafo de um AND

$G_{\mathscr{A}}$ é um grafo dirigido, com rótulos nas arestas.

$V(G_{\mathscr{A}}) = K$

O grafo de um AND

$G_{\mathscr{A}}$ é um grafo dirigido, com rótulos nas arestas.

$V(G_{\mathscr{A}}) = K$

Arestas: para cada $(p, \sigma, q) \in \Delta$, existe uma aresta α de p a q, cujo rótulo $\rho(\alpha)$ é σ.

O grafo de um AND

$G_{\mathscr{A}}$ é um grafo dirigido, com rótulos nas arestas.

$V(G_{\mathscr{A}}) = K$

Arestas: para cada $(p, \sigma, q) \in \Delta$, existe uma aresta α de p a q, cujo rótulo $\rho(\alpha)$ é σ.

A definição de rótulo de um passeio estende a vista para AD.

O grafo de um AND

$G_{\mathscr{A}}$ é um grafo dirigido, com rótulos nas arestas.

$V(G_{\mathscr{A}}) = K$

Arestas: para cada $(p, \sigma, q) \in \Delta$, existe uma aresta α de p a q, cujo rótulo $\rho(\alpha)$ é σ.

A definição de rótulo de um passeio estende a vista para AD.

Um passeio é vencedor se vai de S a F.

A linguagem de um AND

A linguagem de um AND

$$L(\mathscr{A}) = \{\rho(P) \mid P \text{ é um passeio vencedor}\}$$

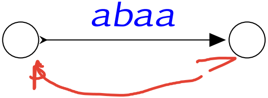
A linguagem de um AND

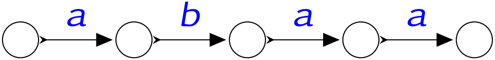
$$L(\mathscr{A}) = \{\rho(P) \mid P \text{ é um passeio vencedor}\}$$

Definição temporária: uma linguagem é **N-reconhecível** se for a linguagem de um AND.

Exemplo

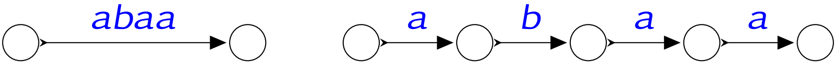
Desenho:



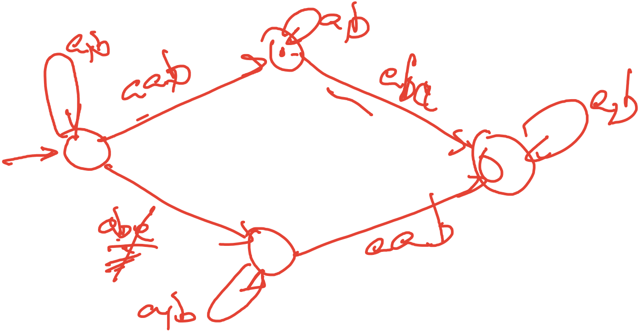


Exemplo

Desenho:



$\{x \in (a+b)^* \mid x \text{ tem fatores } aab \text{ e } aba\}$



AD $\times$ AND

AD × AND

O grafo de um AD pode ser visto como de um AND:

$S = \{s\}$

$\Delta = \{(p, \sigma, p\sigma) \mid p \in K,\ \sigma \in \Sigma\}$.

AD $\times$ AND

O grafo de um AD pode ser visto como de um AND:

$S = \{s\}$

$\Delta = \{(p, \sigma, p\sigma) \mid p \in K,\ \sigma \in \Sigma\}.$

A linguagem aceita pelo grafo é a mesma. Assim, todo AD pode "ser visto" como AND.

AD × AND

O grafo de um AD pode ser visto como de um AND:

$S = \{s\}$

$\Delta = \{(p, \sigma, p\sigma) \mid p \in K,\ \sigma \in \Sigma\}$.

A linguagem aceita pelo grafo é a mesma. Assim, todo AD pode "ser visto" como AND.

Autômato determinístico é caso particular de não determinístico.

AD $\times$ AND

O grafo de um AD pode ser visto como de um AND:

$S = \{s\}$

$\Delta = \{(p, \sigma, p\sigma) \mid p \in K,\ \sigma \in \Sigma\}$.

A linguagem aceita pelo grafo é a mesma. Assim, todo AD pode "ser visto" como AND.

Autômato determinístico é caso particular de não determinístico. Parece piada de mau gosto.

AD × AND

O grafo de um AD pode ser visto como de um AND:
$S = \{s\}$
$\Delta = \{(p, \sigma, p\sigma) \mid p \in K,\ \sigma \in \Sigma\}$.

A linguagem aceita pelo grafo é a mesma. Assim, todo AD pode "ser visto" como AND.

Autômato determinístico é caso particular de não determinístico. Parece piada de mau gosto.

Portanto, se uma linguagem é reconhecível, então ela é N-reconhecível

ED para AND

ED para AND

Principal operação: encontrar passeios em $G_{\mathscr{A}}$ dado o rótulo.

ED para AND

Principal operação: encontrar passeios em $G_{\mathscr{A}}$ dado o rótulo.

Conjuntos como K e Σ podem ser armazenados de qualquer forma; é bom poder indexar por inteiros.

ED para AND

Principal operação: encontrar passeios em $G_{\mathscr{A}}$ dado o rótulo.

Conjuntos como K e Σ podem ser armazenados de qualquer forma; é bom poder indexar por inteiros.

Δ pode ser representado por uma tabela $K \times \Sigma$, em que cada entrada é uma lista de estados.

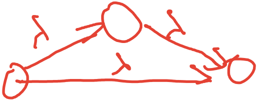
ED para AND

Principal operação: encontrar passeios em $G_{\mathscr{A}}$ dado o rótulo.

Conjuntos como K e Σ podem ser armazenados de qual~~quer forma~~; é bom poder indexar por inteiros.

Δ pode ser representado por uma tabela $K \times \Sigma$, em que cada entrada é uma lista de estados.

A coluna λ pode ser incrementada por meio de uma busca, colocando na linha p todos os estados acessíveis de p por passeios de rótulo λ. A estrutura fica maior.



ED para AND

Principal operação: encontrar passeios em $G_{\mathscr{A}}$ dado o rótulo.

Conjuntos como K e Σ podem ser armazenados de qualquer forma; é bom poder indexar por inteiros.

Δ pode ser representado por uma tabela $K \times \Sigma$, em que cada entrada é uma lista de estados.

A coluna λ pode ser incrementada por meio de uma busca, colocando na linha p todos os estados acessíveis de p por passeios de rótulo λ. A estrutura fica maior.

Dá para eliminar a coluna λ ao custo de multiplicar o tamanho da estrutura toda por $O(|K|)$.

AND e operações booleanas

AND e operações booleanas

sem arestas λ

Tentando imitar as construções que
funcionaram para AD

AND e operações booleanas

Tentando imitar as construções que funcionaram para AD

Interseção:

AND e operações booleanas

Tentando imitar as construções que
funcionaram para AD

Interseção: *funciona fácil*

AND e operações booleanas

Tentando imitar as construções que funcionaram para AD

Interseção: *funciona fácil*

União:

AND e operações booleanas

Tentando imitar as construções que funcionaram para AD

Interseção: *funciona fácil*

$S = S_1 \times S_2$

União: *funciona*, mas precisa pensar. $F = F_1 \times K_2 \cup K_1 \times F_2$

AND e operações booleanas

Tentando imitar as construções que funcionaram para AD

Interseção: *funciona fácil*

União: *funciona*, mas precisa pensar.

Complemento:

AND e operações booleanas

Tentando imitar as construções que funcionaram para AD

Interseção: *funciona fácil*

União: *funciona*, mas precisa pensar.

Complemento: falha feio!

O Teorema de Kleene

Definição

As famílias de linguagens seguintes coincidem:

a) ***Reg***: *o conjunto de linguagens regulares.*
b) ***Rec***: *o conjunto de linguagens reconhecíveis.*
c) ***NRec***: *o conjunto de linguagens N-reconhecíveis.*

O Teorema de Kleene

Definição

As famílias de linguagens seguintes coincidem:

a) ***Reg**: o conjunto de linguagens regulares.*
b) ***Rec**: o conjunto de linguagens reconhecíveis.*
c) ***NRec**: o conjunto de linguagens N-reconhecíveis.*

O Teorema de Kleene

Definição

As famílias de linguagens seguintes coincidem:

a) ***Reg**: o conjunto de linguagens regulares.*
b) ***Rec**: o conjunto de linguagens reconhecíveis.*
c) ***NRec**: o conjunto de linguagens N-reconhecíveis.*

Já vimos que **Rec** $\subseteq$ **NRec**.

Facilidades diferentes

Facilidades diferentes

O reverso de uma palavra x é $x^R = x_n x_{n-1} \cdots x_1$.

Facilidades diferentes

O reverso de uma palavra x é $x^R = x_n x_{n-1} \cdots x_1$.

Definição formal:

Facilidades diferentes

O reverso de uma palavra x é $x^R = x_n x_{n-1} \cdots x_1$.

Definição formal: $\lambda^R = \lambda, (x\sigma)^R = \sigma x^R$.

Facilidades diferentes

O reverso de uma palavra x é $x^R = x_n x_{n-1} \cdots x_1$.

Definição formal: $\lambda^R = \lambda$, $(x\sigma)^R = \sigma x^R$.

Para $L \subseteq \Sigma^*$, $L^R = \{x^R \mid x \in L\}$.

Facilidades diferentes

O reverso de uma palavra x é $x^R = x_n x_{n-1} \cdots x_1$.

Definição formal: $\lambda^R = \lambda, (x\sigma)^R = \sigma x^R$.

Para $L \subseteq \Sigma^*$, $L^R = \{x^R \mid x \in L\}$.

Sem o Teorema de Kleene:

Facilidades diferentes

O reverso de uma palavra x é $x^R = x_n x_{n-1} \cdots x_1$.

Definição formal: $\lambda^R = \lambda$, $(x\sigma)^R = \sigma x^R$.

Para $L \subseteq \Sigma^*$, $L^R = \{x^R \mid x \in L\}$.

Sem o Teorema de Kleene:

Ex: Se L é regular, então L^R também é.

Facilidades diferentes

O reverso de uma palavra x é $x^R = x_n x_{n-1} \cdots x_1$.

Definição formal: $\lambda^R = \lambda$, $(x\sigma)^R = \sigma x^R$.

$(xy)^R = y^R x^R$

Para $L \subseteq \Sigma^*$, $L^R = \{x^R \mid x \in L\}$.

Sem o Teorema de Kleene:

Ex: Se L é regular, então L^R também é.

$(A \cup B)^R = A^R \cup B^R$, $(AB)^R = B^R A^R$, $A^{*R} = A^{R*}$.

Facilidades diferentes

O reverso de uma palavra x é $x^R = x_n x_{n-1} \cdots x_1$.

Definição formal: $\lambda^R = \lambda$, $(x\sigma)^R = \sigma x^R$.

Para $L \subseteq \Sigma^*$, $L^R = \{x^R \mid x \in L\}$.

Sem o Teorema de Kleene:

Ex: Se L é regular, então L^R também é.

$(A \cup B)^R = A^R \cup B^R$, $(AB)^R = B^R A^R$, $A^{*R} = A^{R*}$.

Ex: Se L é N-reconhecível, então L^R também é.

Facilidades diferentes

O reverso de uma palavra x é $x^R = x_n x_{n-1} \cdots x_1$.

Definição formal: $\lambda^R = \lambda$, $(x\sigma)^R = \sigma x^R$.

Para $L \subseteq \Sigma^*$, $L^R = \{x^R \mid x \in L\}$.
Sem o Teorema de Kleene:

Ex: Se L é regular, então L^R também é.
$(A \cup B)^R = A^R \cup B^R$, $(AB)^R = B^R A^R$, $A^{*R} = A^{R*}$.

Ex: Se L é N-reconhecível, então L^R também é.
Bico:

Facilidades diferentes

O reverso de uma palavra x é $x^R = x_n x_{n-1} \cdots x_1$.

Definição formal: $\lambda^R = \lambda$, $(x\sigma)^R = \sigma x^R$.

Para $L \subseteq \Sigma^*$, $L^R = \{x^R \mid x \in L\}$.
Sem o Teorema de Kleene:

Ex: Se L é regular, então L^R também é.
$(A \cup B)^R = A^R \cup B^R$, $(AB)^R = B^R A^R$, $A^{*R} = A^{R*}$.

Ex: Se L é N-reconhecível, então L^R também é.
Bico: Reverta o autômato.

Facilidades diferentes

O reverso de uma palavra x é $x^R = x_n x_{n-1} \cdots x_1$.

Definição formal: $\lambda^R = \lambda$, $(x\sigma)^R = \sigma x^R$.

Para $L \subseteq \Sigma^*$, $L^R = \{x^R \mid x \in L\}$.
Sem o Teorema de Kleene:

Ex: Se L é regular, então L^R também é.
$(A \cup B)^R = A^R \cup B^R$, $(AB)^R = B^R A^R$, $A^{*R} = A^{R*}$.

Ex: Se L é N-reconhecível, então L^R também é.
Bico: Reverta o autômato.

Ex: Se L é reconhecível, então L^R também é.

Facilidades diferentes

O reverso de uma palavra x é $x^R = x_n x_{n-1} \cdots x_1$.

Definição formal: $\lambda^R = \lambda$, $(x\sigma)^R = \sigma x^R$.

Para $L \subseteq \Sigma^*$, $L^R = \{x^R \mid x \in L\}$.

Sem o Teorema de Kleene:

Ex: Se L é regular, então L^R também é.

$(A \cup B)^R = A^R \cup B^R$, $(AB)^R = B^R A^R$, $A^{*R} = A^{R*}$.

Ex: Se L é N-reconhecível, então L^R também é.

Bico: Reverta o autômato.

Ex: Se L é reconhecível, então L^R também é.

Difícil, só com mais teoria!

Autômato com dois terminais

Autômato com dois terminais

É um AND com um único estado inicial, s, um único final, f, e tal que nenhuma aresta entra em s e nenhuma sai de f.

Autômato com dois terminais

É um AND com um único estado inicial, s, um único final, f, e tal que nenhuma aresta entra em s e nenhuma sai de f.

Prop: Toda linguagem N-reconhecível o é por um autômato com 2 terminais.

Autômato com dois terminais

É um AND com um único estado inicial, s, um único final, f, e tal que nenhuma aresta entra em s e nenhuma sai de f.

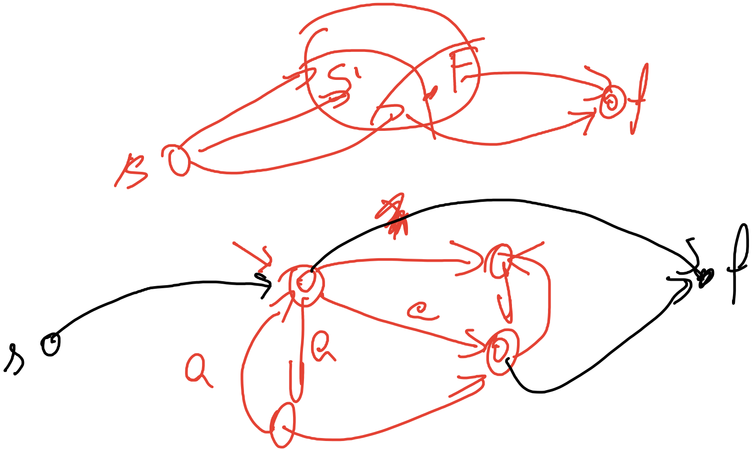
Prop: Toda linguagem N-reconhecível o é por um autômato com 2 terminais.

Dem: Dado $\mathscr{A} = (K, \Sigma, \Delta, S, F)$ seja $\mathscr{A}'$ o resultado de adicionar a $\mathscr{A}$ dois novos estados s,f, e as transições

$$(s, \lambda, p),\ p \in S \qquad (p, \lambda, f),\ p \in F.$$

Então existe uma correspondência bijetora entre os passeios vencedores dos dois autômatos que preserva rótulos. $\square$

Autômato com dois terminais



União de linguagens de A-2t

União de linguagens de A-2t

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

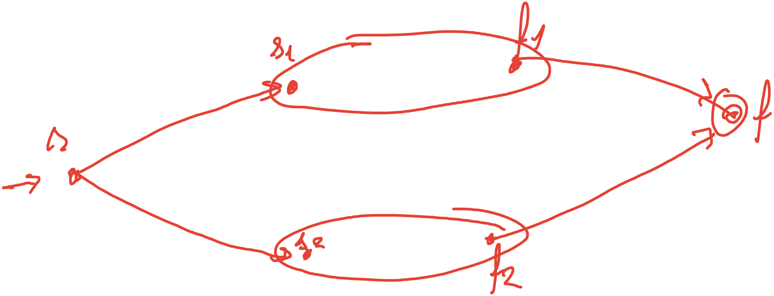
União de linguagens de A-2t

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

$\mathscr{A} = (K, \Sigma, \Delta, s, f)$, s,f novos

$K = K_1 \cup K_2$ $\cup \{s, f\}$

$\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, \lambda, s_1), (s, \lambda, s_2), (f_1, \lambda, f), (f_2, \lambda, f)\}$



União de linguagens de A-2t

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

$\mathscr{A} = (K, \Sigma, \Delta, s, f)$, s,f novos

$K = K_1 \cup K_2$

$\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, \lambda, s_1), (s, \lambda, s_2), (f_1, \lambda, f), (f_2, \lambda, f)\}$

Prop: $L(\mathscr{A}) = L(\mathscr{A}_1) \cup L(\mathscr{A}_2)$

União de linguagens de A-2t

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

$\mathscr{A} = (K, \Sigma, \Delta, s, f)$, s,f novos

$K = K_1 \cup K_2$

$\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, \lambda, s_1), (s, \lambda, s_2), (f_1, \lambda, f), (f_2, \lambda, f)\}$

Prop: $L(\mathscr{A}) = L(\mathscr{A}_1) \cup L(\mathscr{A}_2)$

Dem: Correspondência entre passeios vencedores. $\square$

União de linguagens de A-2t

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

$\mathscr{A} = (K, \Sigma, \Delta, s, f)$, s,f novos

$K = K_1 \cup K_2$

$\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, \lambda, s_1), (s, \lambda, s_2), (f_1, \lambda, f), (f_2, \lambda, f)\}$

Prop: $L(\mathscr{A}) = L(\mathscr{A}_1) \cup L(\mathscr{A}_2)$

Dem: Correspondência entre passeios vencedores. $\square$

Obs: $n = n_1 + n_2 + 2$

União de linguagens de A-2t

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

$\mathscr{A} = (K, \Sigma, \Delta, s, f)$, s,f novos

$K = K_1 \cup K_2$

$\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, \lambda, s_1), (s, \lambda, s_2), (f_1, \lambda, f), (f_2, \lambda, f)\}$

Prop: $L(\mathscr{A}) = L(\mathscr{A}_1) \cup L(\mathscr{A}_2)$

Dem: Correspondência entre passeios vencedores. □

Obs: $n = n_1 + n_2 + 2$

Obs: podia ter identificado s_1 com s_2, f_1 com f_2.

União de linguagens de A-2t

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

$\mathscr{A} = (K, \Sigma, \Delta, s, f)$, s,f novos

$K = K_1 \cup K_2$

$\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, \lambda, s_1), (s, \lambda, s_2), (f_1, \lambda, f), (f_2, \lambda, f)\}$

Prop: $L(\mathscr{A}) = L(\mathscr{A}_1) \cup L(\mathscr{A}_2)$

Dem: Correspondência entre passeios vencedores. □

Obs: $n = n_1 + n_2 + 2$

Obs: podia ter identificado s_1 com s_2, f_1 com f_2.

$n = n_1 + n_2 - 2$

Produto de linguagens de A-2t

;

Produto de linguagens de A-2t

;

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

Produto de linguagens de A-2t

;

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

$\mathscr{A} = (K_1 \cup K_2, \Sigma, \Delta_1 \cup \Delta_2 \cup \{(f_1, \lambda, s_2)\}, s_1, f_2)$

Produto de linguagens de A-2t

;

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

$\mathscr{A} = (K_1 \cup K_2, \Sigma, \Delta_1 \cup \Delta_2 \cup \{(f_1, \lambda, s_2)\}, s_1, f_2)$

Obs: $n = n_1 + n_2$

Produto de linguagens de A-2t

;

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

$\mathscr{A} = (K_1 \cup K_2, \Sigma, \Delta_1 \cup \Delta_2 \cup \{(f_1, \lambda, s_2)\}, s_1, f_2)$

Obs: $n = n_1 + n_2$

Prop: $L(\mathscr{A}) = L(\mathscr{A}_1)L(\mathscr{A}_2)$

Produto de linguagens de A-2t

;

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

$\mathscr{A} = (K_1 \cup K_2, \Sigma, \Delta_1 \cup \Delta_2 \cup \{(f_1, \lambda, s_2)\}, s_1, f_2)$

Obs: $n = n_1 + n_2$

Prop: $L(\mathscr{A}) = L(\mathscr{A}_1)L(\mathscr{A}_2)$

Dem:

- $\supseteq$ Caminhos vencedores em $\mathscr{A}_1, \mathscr{A}_2$ se juntam para formar um em $\mathscr{A}$.

;

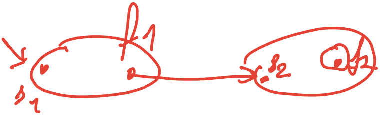
Produto de linguagens de A-2t

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

$\mathscr{A} = (K_1 \cup K_2, \Sigma, \Delta_1 \cup \Delta_2 \cup \{(f_1, \lambda, s_2)\}, s_1, f_2)$

Obs: $n = n_1 + n_2$

Prop: $L(\mathscr{A}) = L(\mathscr{A}_1)L(\mathscr{A}_2)$



Dem:

- $\supseteq$ Caminhos vencedores em $\mathscr{A}_1, \mathscr{A}_2$ se juntam para formar um em $\mathscr{A}$.
- $\subseteq$ Caminho vencedor em $\mathscr{A}$ deve usar a aresta (f_1, λ, s_2); assim decompõe em caminhos vencedores nos autômatos dados. $\square$

Produto de linguagens de A-2t

;

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

$\mathscr{A} = (K_1 \cup K_2, \Sigma, \Delta_1 \cup \Delta_2 \cup \{(f_1, \lambda, s_2)\}, s_1, f_2)$

Obs: $n = n_1 + n_2$

Prop: $L(\mathscr{A}) = L(\mathscr{A}_1)L(\mathscr{A}_2)$

Dem:

- $\supseteq$ Caminhos vencedores em $\mathscr{A}_1, \mathscr{A}_2$ se juntam para formar um em $\mathscr{A}$.
- $\subseteq$ Caminho vencedor em $\mathscr{A}$ deve usar a aresta (f_1, λ, s_2); assim decompõe em caminhos vencedores nos autômatos dados. $\square$

Produto de linguagens de A-2t

;

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

$\mathscr{A} = (K_1 \cup K_2, \Sigma, \Delta_1 \cup \Delta_2 \cup \{(f_1, \lambda, s_2)\}, s_1, f_2)$

Obs: $n = n_1 + n_2$

Prop: $L(\mathscr{A}) = L(\mathscr{A}_1)L(\mathscr{A}_2)$

Dem:

- $\supseteq$ Caminhos vencedores em $\mathscr{A}_1, \mathscr{A}_2$ se juntam para formar um em $\mathscr{A}$.
- $\subseteq$ Caminho vencedor em $\mathscr{A}$ deve usar a aresta (f_1, λ, s_2); assim decompõe em caminhos vencedores nos autômatos dados. $\square$

Obs: podia ter identificado f_1 com s_2.

Produto de linguagens de A-2t

;

$\mathscr{A}_i = (K_i, \Sigma, \Delta_i, s_i, f_i)$, $i = 1, 2$, $K_1 \cap K_2 = \emptyset$.

$\mathscr{A} = (K_1 \cup K_2, \Sigma, \Delta_1 \cup \Delta_2 \cup \{(f_1, \lambda, s_2)\}, s_1, f_2)$

Obs: $n = n_1 + n_2$

Prop: $L(\mathscr{A}) = L(\mathscr{A}_1)L(\mathscr{A}_2)$

Dem:

- $\supseteq$ Caminhos vencedores em $\mathscr{A}_1, \mathscr{A}_2$ se juntam para formar um em $\mathscr{A}$.
- $\subseteq$ Caminho vencedor em $\mathscr{A}$ deve usar a aresta (f_1, λ, s_2); assim decompõe em caminhos vencedores nos autômatos dados. $\square$

Obs: podia ter identificado f_1 com s_2. $n = n_1 + n_2 - 1$

Estrela de linguagem de A-2t

Estrela de linguagem de A-2t

$\mathscr{A}_1 = (K_1, \Sigma, \Delta_1, s_1, f_1)$

Estrela de linguagem de A-2t

$\mathscr{A}_1 = (K_1, \Sigma, \Delta_1, s_1, f_1)$

$\mathscr{A} = (K_1 \cup \{s, f\}, \Sigma, \Delta, s, f)$

$\Delta = \Delta_1 \cup \{(s, \lambda, s_1), (f_1, \lambda, f), (s, \lambda, f), (f_1, \lambda, s_1)\}$

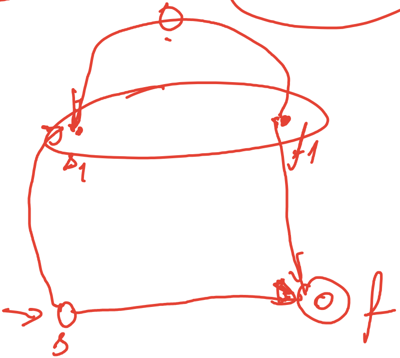
Estrela de linguagem de A-2t

$\mathscr{A}_1 = (K_1, \Sigma, \Delta_1, s_1, f_1)$

$\mathscr{A} = (K_1 \cup \{s, f\}, \Sigma, \Delta, s, f)$

$\quad \Delta = \Delta_1 \cup \{(s, \lambda, s_1), (f_1, \lambda, f), (s, \lambda, f), (f_1, \lambda, s_1)\}$

Prop: $L(\mathscr{A}) = L(\mathscr{A}_1)^*$



Estrela de linguagem de A-2t

$\mathscr{A}_1 = (K_1, \Sigma, \Delta_1, s_1, f_1)$

$\mathscr{A} = (K_1 \cup \{s, f\}, \Sigma, \Delta, s, f)$
$\quad \Delta = \Delta_1 \cup \{(s, \lambda, s_1), (f_1, \lambda, f), (s, \lambda, f), (f_1, \lambda, s_1)\}$

Prop: $L(\mathscr{A}) = L(\mathscr{A}_1)^*$

Obs: $n = n_1 + 2$

Outro pedaço do T. de Kleene

Outro pedaço do T. de Kleene

Prop: *Toda linguagem regular é N-reconhecível.*

Outro pedaço do T. de Kleene

Prop: *Toda linguagem regular é N-reconhecível.*

Dem: Vamos mostrar um algoritmo que, para toda ER E constrói um autômato 2t $\mathscr{A}$ tal que $\mathscr{L}(E) = L(\mathscr{A})$. O algoritmo é recursivo;

Outro pedaço do T. de Kleene

Prop: *Toda linguagem regular é N-reconhecível.*

Dem: Vamos mostrar um algoritmo que, para toda ER E constrói um autômato 2t $\mathscr{A}$ tal que $\mathscr{L}(E) = L(\mathscr{A})$. O algoritmo é recursivo; mais tradicionalmente, a prova é por indução.

Outro pedaço do T. de Kleene

Prop: *Toda linguagem regular é N-reconhecível.*

Dem: Vamos mostrar um algoritmo que, para toda ER E constrói um autômato 2t $\mathscr{A}$ tal que $\mathscr{L}(E) = L(\mathscr{A})$. O algoritmo é recursivo; mais tradicionalmente, a prova é por indução.

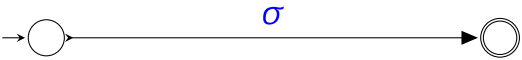
Base: $E = \sigma \in \hat{\Sigma}$

Outro pedaço do T. de Kleene

Prop: *Toda linguagem regular é N-reconhecível.*

Dem: Vamos mostrar um algoritmo que, para toda ER E constrói um autômato 2t $\mathscr{A}$ tal que $\mathscr{L}(E) = L(\mathscr{A})$. O algoritmo é recursivo; mais tradicionalmente, a prova é por indução.

Base: $E = \sigma \in \hat{\Sigma}$

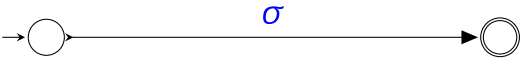


Outro pedaço do T. de Kleene

Prop: *Toda linguagem regular é N-reconhecível.*

Dem: Vamos mostrar um algoritmo que, para toda ER E constrói um autômato 2t $\mathscr{A}$ tal que $\mathscr{L}(E) = L(\mathscr{A})$. O algoritmo é recursivo; mais tradicionalmente, a prova é por indução.

Base: $E = \sigma \in \hat{\Sigma}$



Passo: E se expressa em termos de ERs menores e uma das operações. Use as construções anteriores. Ken Thompson grep Unix A. Turing $\square$

O tamanho do autômato

Começando com uma ER E.

O tamanho do autômato

Começando com uma ER E.

Cada letra de E dá dois estados.

O tamanho do autômato

Começando com uma ER E.

Cada letra de E dá dois estados.

Cada símbolo $+,^*$ dá mais dois estados.

O tamanho do autômato

Começando com uma ER E.

Cada letra de E dá dois estados.

Cada símbolo $+, ^*$ dá mais dois estados.

Total de estados menor que $2|E|$.

O tamanho do autômato

Começando com uma ER E.

Cada letra de E dá dois estados.

Cada símbolo $+,^*$ dá mais dois estados.

Total de estados menor que $2|E|$.

De cada estado~~s~~ saem no máximo 2 arestas!

O tamanho do autômato

Começando com uma ER E.

Cada letra de E dá dois estados.

Cada símbolo $+,^*$ dá mais dois estados.

Total de estados menor que $2|E|$.

De cada estados saem no máximo 2 arestas! Tabela com informação de tamanho fixo por estado.

O tamanho do autômato

Começando com uma ER E.

Cada letra de E dá dois estados.

Cada símbolo $+, ^*$ dá mais dois estados.

Total de estados menor que $2|E|$.

De cada estados saem no máximo 2 arestas! Tabela com informação de tamanho fixo por estado.

$|\Delta| \leq 4|E|$.

$$(aa + baa)^* a (ba)^*$$