# CC1004 - Modelos de Computação
# Teórica 14

Ana Paula Tomás

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto

Abril 2021

# Autómato produto de dois AFDs

O **autómato produto** $A_1 \times A_2$ de $A_1 = (S, \Sigma, \delta_1, s_0, F_1)$ e $A_2 = (Q, \Sigma, \delta_2, q_0, F_2)$ é um AFD

$$A_1 \times A_2 = (S \times Q, \Sigma, \delta, (s_0, q_0), F),$$

com $\delta((s, q), a) = (\delta_1(s, a), \delta_2(q, a))$, para $(s, q) \in S \times Q$, e $a \in \Sigma$. Em vez de $S \times Q$, podemos restringir aos **estados acessíveis do estado inicial** $(s_0, q_0)$.

Tal AFD simula a execução de $A_1$ e $A_2$ paralelamente e, dependendo do modo como definimos $F$, reconhecerá $\mathcal{L}(A_1) \cap \mathcal{L}(A_2)$, $\mathcal{L}(A_1) \cup \mathcal{L}(A_2)$, $\mathcal{L}(A_1) \setminus \mathcal{L}(A_2)$.

- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$ se $F = F_1 \times F_2 = \{(s, q) \mid s \in F_1 \text{ e } q \in F_2\}$;
- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$ se $F = (F_1 \times Q) \cup (S \times F_2) = \{(s, q) \mid s \in F_1 \text{ ou } q \in F_2\}$.
- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \setminus \mathcal{L}(A_2)$ se $F = F_1 \times (Q \setminus F_2) = \{(s, q) \mid s \in F_1 \text{ e } q \notin F_2\}$.

Os exemplos seguintes mostram que **o AFD produto pode não ser o AFD mínimo**.

# Autómato produto de dois AFDs

O **autómato produto** $A_1 \times A_2$ de $A_1 = (S, \Sigma, \delta_1, s_0, F_1)$ e $A_2 = (Q, \Sigma, \delta_2, q_0, F_2)$ é um AFD

$$A_1 \times A_2 = (S \times Q, \Sigma, \delta, (s_0, q_0), F),$$

com $\delta((s, q), a) = (\delta_1(s, a), \delta_2(q, a))$, para $(s, q) \in S \times Q$, e $a \in \Sigma$. Em vez de $S \times Q$, podemos restringir aos **estados acessíveis do estado inicial** $(s_0, q_0)$.

Tal AFD simula a execução de $A_1$ e $A_2$ paralelamente e, dependendo do modo como definimos $F$, reconhecerá $\mathcal{L}(A_1) \cap \mathcal{L}(A_2)$, $\mathcal{L}(A_1) \cup \mathcal{L}(A_2)$, $\mathcal{L}(A_1) \setminus \mathcal{L}(A_2)$.

- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$ se $F = F_1 \times F_2 = \{(s, q) \mid s \in F_1 \text{ e } q \in F_2\}$;
- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$ se $F = (F_1 \times Q) \cup (S \times F_2) = \{(s, q) \mid s \in F_1 \text{ ou } q \in F_2\}$.
- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \setminus \mathcal{L}(A_2)$ se $F = F_1 \times (Q \setminus F_2) = \{(s, q) \mid s \in F_1 \text{ e } q \notin F_2\}$.

Os exemplos seguintes mostram que **o AFD produto pode não ser o AFD mínimo**.

# Autómato produto de dois AFDs

O **autómato produto** $A_1 \times A_2$ de $A_1 = (S, \Sigma, \delta_1, s_0, F_1)$ e $A_2 = (Q, \Sigma, \delta_2, q_0, F_2)$ é um AFD

$$A_1 \times A_2 = (S \times Q, \Sigma, \delta, (s_0, q_0), F),$$

com $\delta((s, q), a) = (\delta_1(s, a), \delta_2(q, a))$, para $(s, q) \in S \times Q$, e $a \in \Sigma$. Em vez de $S \times Q$, podemos restringir aos **estados acessíveis do estado inicial** $(s_0, q_0)$.

Tal AFD simula a execução de $A_1$ e $A_2$ paralelamente e, dependendo do modo como definimos $F$, reconhecerá $\mathcal{L}(A_1) \cap \mathcal{L}(A_2)$, $\mathcal{L}(A_1) \cup \mathcal{L}(A_2)$, $\mathcal{L}(A_1) \setminus \mathcal{L}(A_2)$.

- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$ se $F = F_1 \times F_2 = \{(s, q) \mid s \in F_1 \text{ e } q \in F_2\}$;
- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$ se $F = (F_1 \times Q) \cup (S \times F_2) = \{(s, q) \mid s \in F_1 \text{ ou } q \in F_2\}$.
- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \setminus \mathcal{L}(A_2)$ se $F = F_1 \times (Q \setminus F_2) = \{(s, q) \mid s \in F_1 \text{ e } q \notin F_2\}$.

Os exemplos seguintes mostram que **o AFD produto pode não ser o AFD mínimo**.

# Autómato produto de dois AFDs

O **autómato produto** $A_1 \times A_2$ de $A_1 = (S, \Sigma, \delta_1, s_0, F_1)$ e $A_2 = (Q, \Sigma, \delta_2, q_0, F_2)$ é um AFD

$$A_1 \times A_2 = (S \times Q, \Sigma, \delta, (s_0, q_0), F),$$

com $\delta((s, q), a) = (\delta_1(s, a), \delta_2(q, a))$, para $(s, q) \in S \times Q$, e $a \in \Sigma$. Em vez de $S \times Q$, podemos restringir aos **estados acessíveis do estado inicial** $(s_0, q_0)$.

Tal AFD simula a execução de $A_1$ e $A_2$ paralelamente e, dependendo do modo como definimos $F$, reconhecerá $\mathcal{L}(A_1) \cap \mathcal{L}(A_2)$, $\mathcal{L}(A_1) \cup \mathcal{L}(A_2)$, $\mathcal{L}(A_1) \setminus \mathcal{L}(A_2)$.

- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$ se $F = F_1 \times F_2 = \{(s, q) \mid s \in F_1 \text{ e } q \in F_2\}$;
- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$ se $F = (F_1 \times Q) \cup (S \times F_2) = \{(s, q) \mid s \in F_1 \text{ ou } q \in F_2\}$.
- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \setminus \mathcal{L}(A_2)$ se $F = F_1 \times (Q \setminus F_2) = \{(s, q) \mid s \in F_1 \text{ e } q \notin F_2\}$.

Os exemplos seguintes mostram que **o AFD produto pode não ser o AFD mínimo**.

# Autómato produto de dois AFDs

O **autómato produto** $A_1 \times A_2$ de $A_1 = (S, \Sigma, \delta_1, s_0, F_1)$ e $A_2 = (Q, \Sigma, \delta_2, q_0, F_2)$ é um AFD

$$A_1 \times A_2 = (S \times Q, \Sigma, \delta, (s_0, q_0), F),$$

com $\delta((s, q), a) = (\delta_1(s, a), \delta_2(q, a))$, para $(s, q) \in S \times Q$, e $a \in \Sigma$. Em vez de $S \times Q$, podemos restringir aos **estados acessíveis do estado inicial** $(s_0, q_0)$.

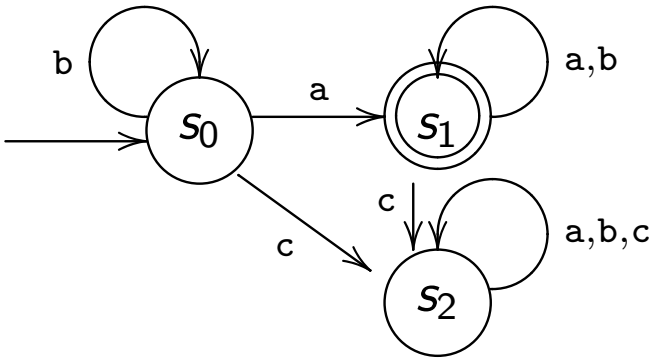
Tal AFD simula a execução de $A_1$ e $A_2$ paralelamente e, dependendo do modo como definimos $F$, reconhecerá $\mathcal{L}(A_1) \cap \mathcal{L}(A_2)$, $\mathcal{L}(A_1) \cup \mathcal{L}(A_2)$, $\mathcal{L}(A_1) \setminus \mathcal{L}(A_2)$.

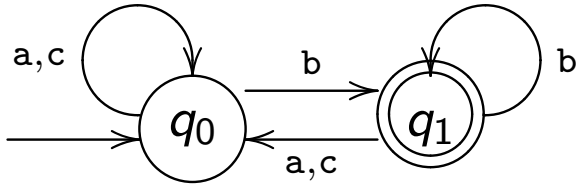
- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$ se $F = F_1 \times F_2 = \{(s, q) \mid s \in F_1 \text{ e } q \in F_2\}$;
- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$ se $F = (F_1 \times Q) \cup (S \times F_2) = \{(s, q) \mid s \in F_1 \text{ ou } q \in F_2\}$.
- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \setminus \mathcal{L}(A_2)$ se $F = F_1 \times (Q \setminus F_2) = \{(s, q) \mid s \in F_1 \text{ e } q \notin F_2\}$.

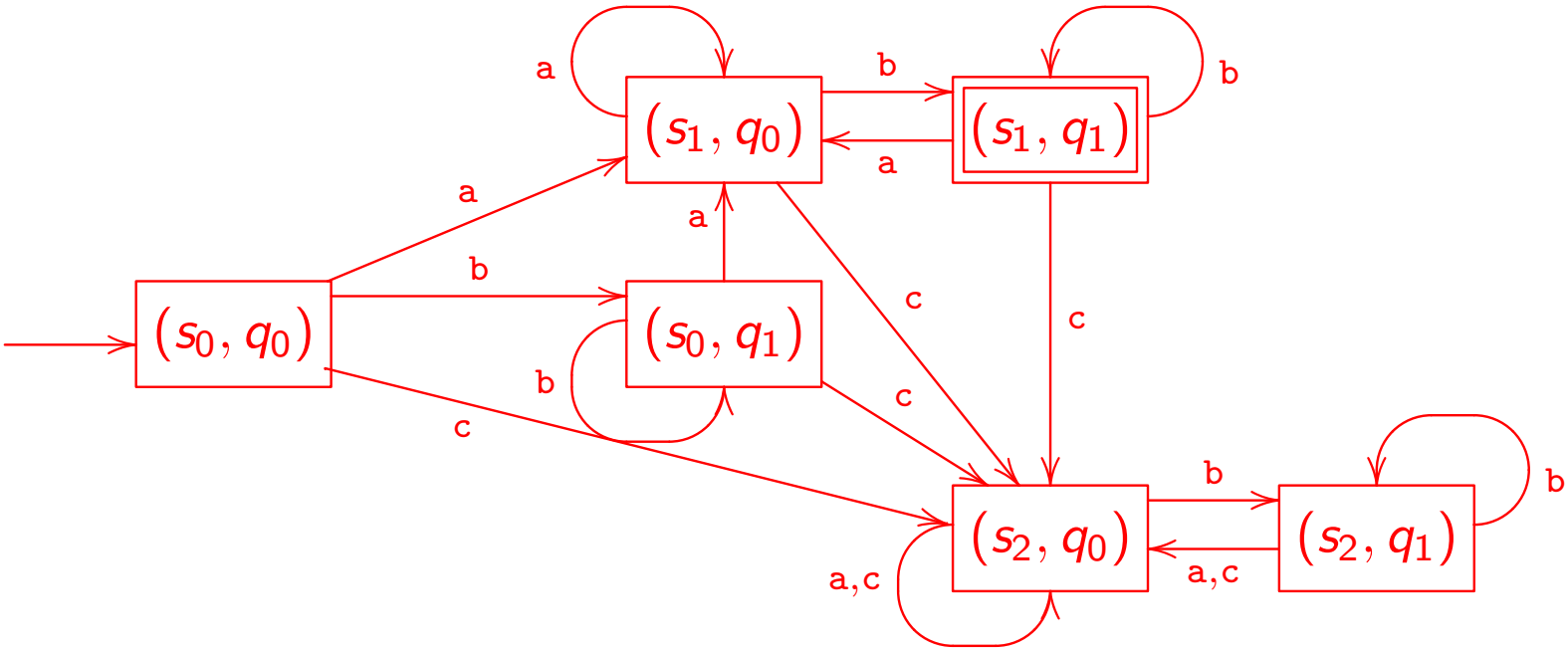
Os exemplos seguintes mostram que **o AFD produto pode não ser o AFD mínimo**.

# Autómato produto de dois AFDs

**Exemplo:** Dois AFDs $A_1$ e $A_2$ e o AFD produto para a interseção $\mathcal{L}(A_1) \cap \mathcal{L}(A_2)$

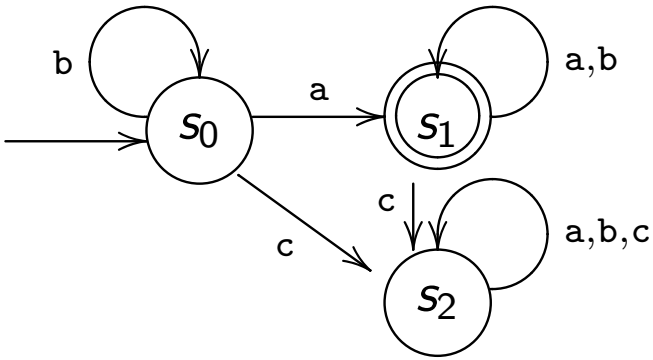


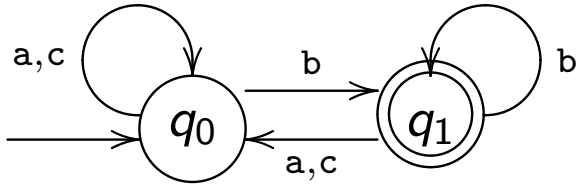



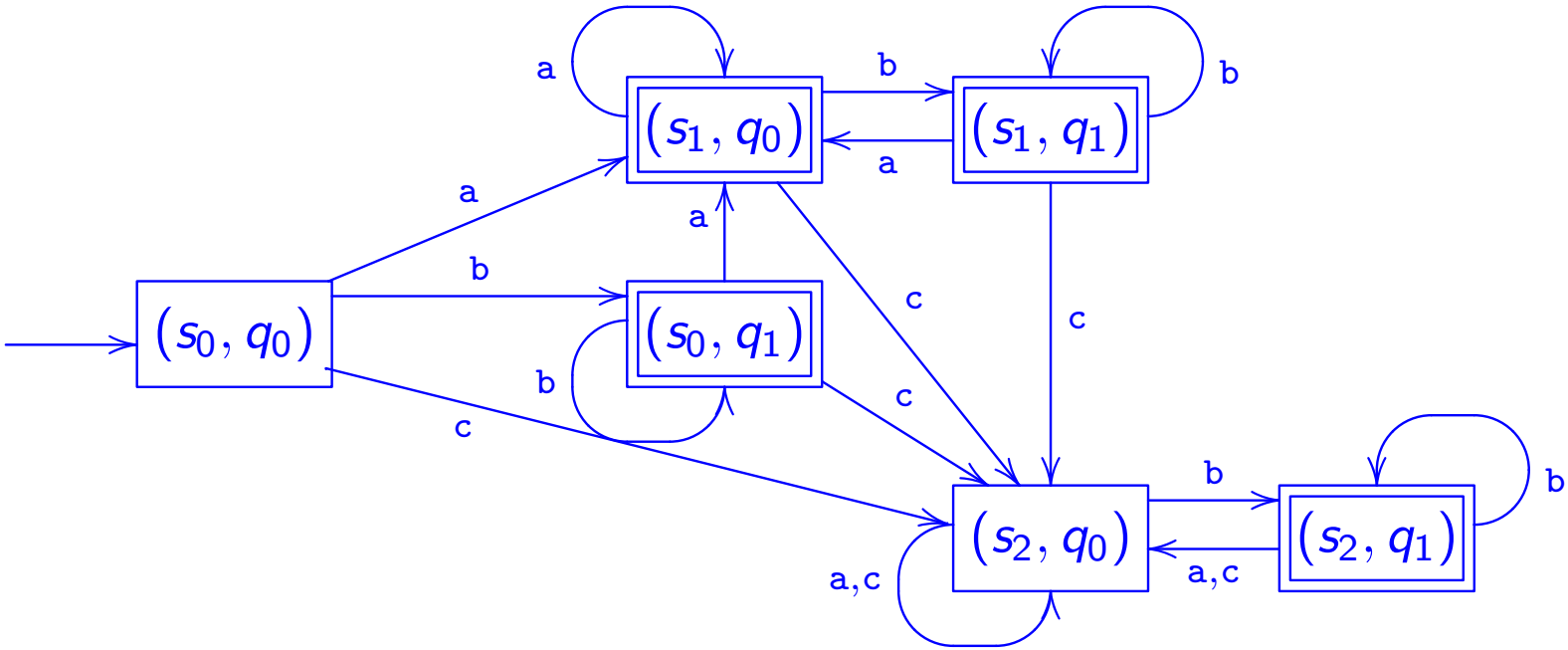



# Autómato produto de dois AFDs

**Exemplo:** Dois AFDs $A_1$ e $A_2$ e o AFD produto para a união $\mathcal{L}(A_1) \cap \mathcal{L}(A_2)$

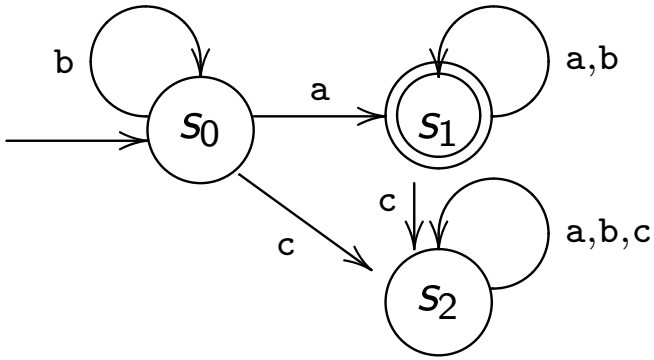


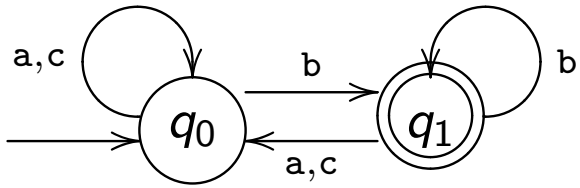



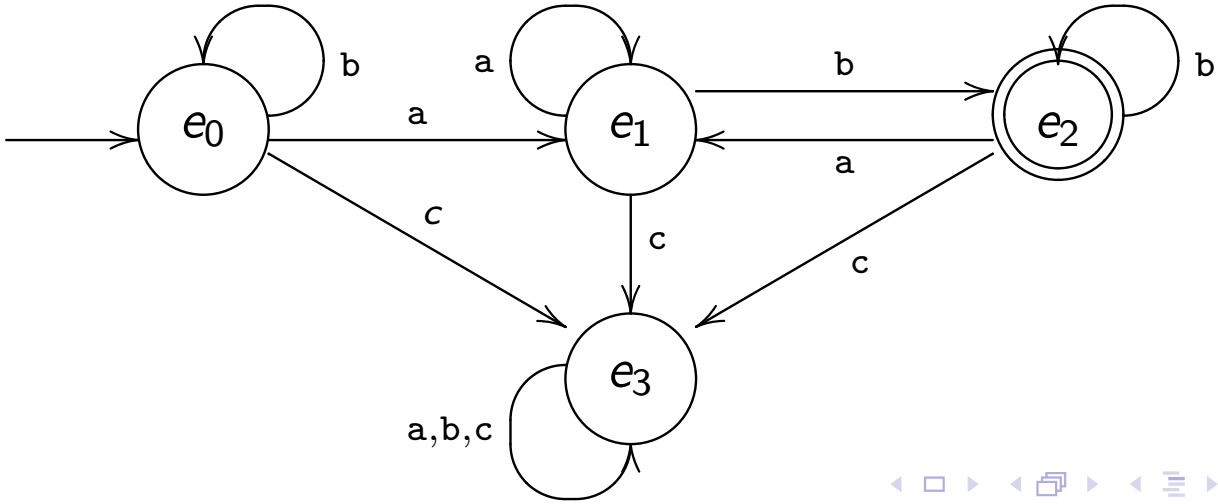
# Autómato produto de dois AFDs

O AFD obtido pela construção de AFD produto pode não ser o AFD mínimo. Por exemplo, vimos que tinha seis estados para



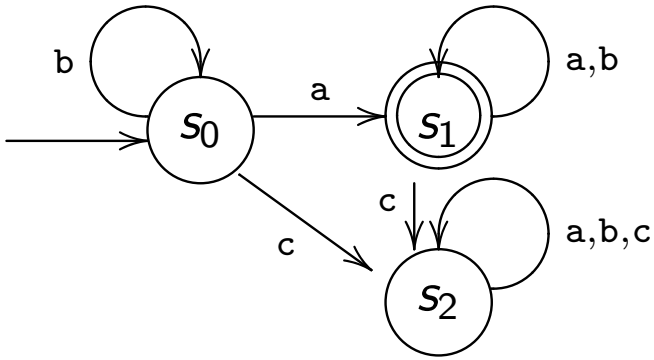



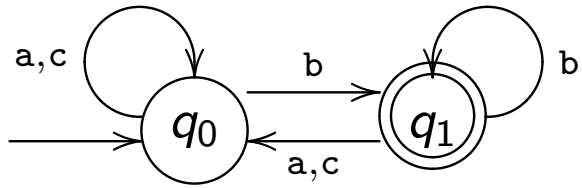

mas $\mathcal{L}(A_1) \cap \mathcal{L}(A_2)$ é o conjunto das palavras de $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^\star$ que têm algum a, terminam em b e não têm c's, e o AFD mínimo para $\mathcal{L}(A_1) \cap \mathcal{L}(A_2)$ é



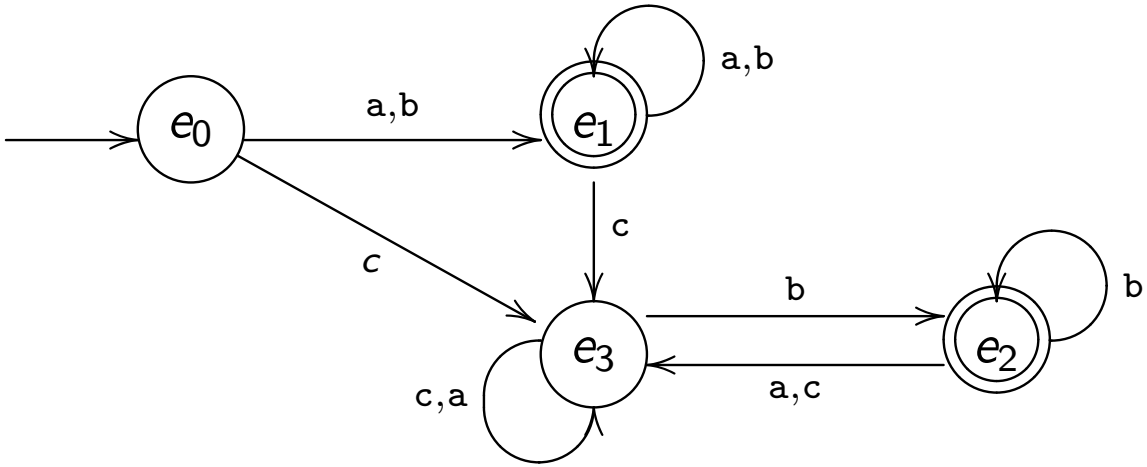
# Autómato produto de dois AFDs

Também para $\mathcal{L}(A_1) \cup \mathcal{L}(A_2)$ o AFD obtido pela construção de AFD produto para




não seria o AFD mínimo. $\mathcal{L}(A_1) \cup \mathcal{L}(A_2)$ é o conjunto das palavras de $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^\star$ que terminam em a ou b e não têm c's ou têm c's e terminam em b. O AFD minimo para essa linguagem tem quatro estados.



# Existência de algoritmos de decisão

Nem todos os problemas podem ser resolvidos computacionalmente.

Mas, muitos **problemas de decisão** sobre linguagens regulares e autómatos finitos podem ser resolvidos computacionalmente: **existem algoritmos** que produzem uma resposta "sim/não" para cada instância do problema.

**Por exemplo, para os problemas seguintes:**

- Dado um autómato finito $A$ e dado $x \in \Sigma^\star$, determinar se $x \in \mathcal{L}(A)$.
- Dada uma expressão regular $r$ e uma palavra $x \in \Sigma^\star$, determinar se $x \in \mathcal{L}(r)$.
- Dados dois AFDs $A_1$ e $A_2$, decidir se
  - $\mathcal{L}(A_1) \cup \mathcal{L}(A_2) = \Sigma^\star$
  - $\mathcal{L}(A_1) = \mathcal{L}(A_2)$, isto é, os AFDs $A_1$ e $A_2$ são equivalentes
  - $\mathcal{L}(A_1) \cap \mathcal{L}(A_2) = \emptyset$
  - $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$
- Dadas duas expressões regulares $r_1$ e $r_2$, decidir se $r_1$ e $r_2$ são equivalentes, i.e., decidir se $\mathcal{L}(r_1) = \mathcal{L}(r_2)$.

# Existência de algoritmos de decisão

**Justificação:**

- Dado um autómato finito $A$ e dado $x \in \Sigma^\star$, determinar se $x \in \mathcal{L}(A)$.

  *Verificar se existe um percurso de $s_0$ para algum $f \in F$, sendo $x$ a concatenação dos valores nos ramos.*

- Dada uma expressão regular $r$ e uma palavra $x \in \Sigma^\star$, determinar se $x \in \mathcal{L}(r)$.

  *Converter a expressão para um AF e aplicar o anterior.*

- Dadas duas expressões regulares $r_1$ e $r_2$, decidir se $r_1$ e $r_2$ são equivalentes, i.e., decidir se $\mathcal{L}(r_1) = \mathcal{L}(r_2)$.

  Aplicar o algoritmo de Thompson às expressões para ter AFNDs-$\varepsilon$; converter para AFDs; aplicar o algoritmo de Moore para os minimizar: Verificar se os AFDs mínimos são iguais a menos da designação de estados.

# Existência de algoritmos de decisão

**Justificação:**

- Dado um autómato finito $A$ e dado $x \in \Sigma^\star$, determinar se $x \in \mathcal{L}(A)$.

  *Verificar se existe um percurso de $s_0$ para algum $f \in F$, sendo $x$ a concatenação dos valores nos ramos.*

- Dada uma expressão regular $r$ e uma palavra $x \in \Sigma^\star$, determinar se $x \in \mathcal{L}(r)$.

  *Converter a expressão para um AF e aplicar o anterior.*

- Dadas duas expressões regulares $r_1$ e $r_2$, decidir se $r_1$ e $r_2$ são equivalentes, i.e., decidir se $\mathcal{L}(r_1) = \mathcal{L}(r_2)$.

  Aplicar o algoritmo de Thompson às expressões para ter AFNDs-$\varepsilon$; converter para AFDs; aplicar o algoritmo de Moore para os minimizar: Verificar se os AFDs mínimos são iguais a menos da designação de estados.

# Existência de algoritmos de decisão

**Justificação:**

- Dado um autómato finito $A$ e dado $x \in \Sigma^\star$, determinar se $x \in \mathcal{L}(A)$.

  *Verificar se existe um percurso de $s_0$ para algum $f \in F$, sendo $x$ a concatenação dos valores nos ramos.*

- Dada uma expressão regular $r$ e uma palavra $x \in \Sigma^\star$, determinar se $x \in \mathcal{L}(r)$.

  *Converter a expressão para um AF e aplicar o anterior.*

- Dadas duas expressões regulares $r_1$ e $r_2$, decidir se $r_1$ e $r_2$ são equivalentes, i.e., decidir se $\mathcal{L}(r_1) = \mathcal{L}(r_2)$.

  Aplicar o algoritmo de Thompson às expressões para ter AFNDs-$\varepsilon$; converter para AFDs; aplicar o algoritmo de Moore para os minimizar: Verificar se os AFDs mínimos são iguais a menos da designação de estados.

# Existência de algoritmos de decisão

Nos exemplos seguintes, **definimos o AFD produto, mantendo apenas os estados acessíveis do estado inicial**.

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) \cup \mathcal{L}(A_2) = \Sigma^\star$.

  *Construir o AFD produto para a união e verificar se todos os estados são finais.*

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) = \mathcal{L}(A_2)$.

  *Construir o AFD produto para a interseção e verificar se os estados não finais (se existirem) não incluem estados finais de $A_1$ nem de $A_2$.*

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) \cap \mathcal{L}(A_2) = \emptyset$

  *Construir o AFD produto para a interseção e verificar se não tem estados finais.*

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$

  *Construir o AFD produto para a interseção e verificar se os estados não finais (se existirem) não incluem estados finais de $A_1$.*

O número de estados do AFD produto é polinomial no número de estados dos AFDs de partida pois, no pior caso, é $|S_1 \times S_2| = |S_1||S_2|$, o que permite ter algoritmos eficientes.

# Existência de algoritmos de decisão

Nos exemplos seguintes, **definimos o AFD produto, mantendo apenas os estados acessíveis do estado inicial**.

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) \cup \mathcal{L}(A_2) = \Sigma^\star$.

  *Construir o AFD produto para a união e verificar se todos os estados são finais.*

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) = \mathcal{L}(A_2)$.

  *Construir o AFD produto para a interseção e verificar se os estados não finais (se existirem) não incluem estados finais de $A_1$ nem de $A_2$.*

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) \cap \mathcal{L}(A_2) = \emptyset$

  *Construir o AFD produto para a interseção e verificar se não tem estados finais.*

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$

  *Construir o AFD produto para a interseção e verificar se os estados não finais (se existirem) não incluem estados finais de $A_1$.*

O número de estados do AFD produto é polinomial no número de estados dos AFDs de partida pois, no pior caso, é $|S_1 \times S_2| = |S_1||S_2|$, o que permite ter algoritmos eficientes.

# Existência de algoritmos de decisão

Nos exemplos seguintes, **definimos o AFD produto, mantendo apenas os estados acessíveis do estado inicial**.

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) \cup \mathcal{L}(A_2) = \Sigma^\star$.

  *Construir o AFD produto para a união e verificar se todos os estados são finais.*

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) = \mathcal{L}(A_2)$.

  *Construir o AFD produto para a interseção e verificar se os estados não finais (se existirem) não incluem estados finais de $A_1$ nem de $A_2$.*

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) \cap \mathcal{L}(A_2) = \emptyset$

  *Construir o AFD produto para a interseção e verificar se não tem estados finais.*

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$

  *Construir o AFD produto para a interseção e verificar se os estados não finais (se existirem) não incluem estados finais de $A_1$.*

O número de estados do AFD produto é polinomial no número de estados dos AFDs de partida pois, no pior caso, é $|S_1 \times S_2| = |S_1||S_2|$, o que permite ter algoritmos eficientes.

# Existência de algoritmos de decisão

Nos exemplos seguintes, **definimos o AFD produto, mantendo apenas os estados acessíveis do estado inicial**.

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) \cup \mathcal{L}(A_2) = \Sigma^\star$.

  *Construir o AFD produto para a união e verificar se todos os estados são finais.*

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) = \mathcal{L}(A_2)$.

  *Construir o AFD produto para a interseção e verificar se os estados não finais (se existirem) não incluem estados finais de $A_1$ nem de $A_2$.*

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) \cap \mathcal{L}(A_2) = \emptyset$

  *Construir o AFD produto para a interseção e verificar se não tem estados finais.*

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$

  *Construir o AFD produto para a interseção e verificar se os estados não finais (se existirem) não incluem estados finais de $A_1$.*

O número de estados do AFD produto é polinomial no número de estados dos AFDs de partida pois, no pior caso, é $|S_1 \times S_2| = |S_1||S_2|$, o que permite ter algoritmos eficientes.

# Existência de algoritmos de decisão

Nos exemplos seguintes, **definimos o AFD produto, mantendo apenas os estados acessíveis do estado inicial**.

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) \cup \mathcal{L}(A_2) = \Sigma^\star$.

  *Construir o AFD produto para a união e verificar se todos os estados são finais.*

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) = \mathcal{L}(A_2)$.

  *Construir o AFD produto para a interseção e verificar se os estados não finais (se existirem) não incluem estados finais de $A_1$ nem de $A_2$.*

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) \cap \mathcal{L}(A_2) = \emptyset$

  *Construir o AFD produto para a interseção e verificar se não tem estados finais.*

- Dados dois AFDs $A_1$ e $A_2$, decidir se $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$

  *Construir o AFD produto para a interseção e verificar se os estados não finais (se existirem) não incluem estados finais de $A_1$.*

O número de estados do AFD produto é polinomial no número de estados dos AFDs de partida pois, no pior caso, é $|S_1 \times S_2| = |S_1||S_2|$, o que permite ter algoritmos eficientes.

# Autómato para a reversa $\mathcal{L}(A)^R$, sendo $A$ um AFD

A **palavra reversa de** $w$, denotada por $w^R$, é $w$ escrita da direita para a esquerda. Tal operação pode ser definida formalmente por: $\varepsilon^R = \varepsilon$ e $(ax)^R = x^R a$, para todo $x \in \Sigma^\star$ e $a \in \Sigma$. A **linguagem reversa de** $L$ é

$$L^R = \{w^R \mid w \in L\}$$

Dado um AFD $A = (S, \Sigma, \delta, s_0, F)$, **o seguinte AFND-$\varepsilon$ $A'$ reconhece** $\mathcal{L}(A)^R$:

$$A' = (S \cup \{i\}, \Sigma, \delta', i, \{s_0\})$$

onde $i$ é um estado novo e a função $\delta'$ de $(S \cup \{i\}) \times (\Sigma \cup \{\varepsilon\})$ é dada por

$$\begin{aligned} \delta'(i, \varepsilon) &= F \\ \delta'(s, a) &= \{s' \mid \delta(s', a) = s\}, \text{para todo } s \in S \text{ e } a \in \Sigma \\ \delta'(s, \alpha) &= \{\}, \text{para os restantes } (s, \alpha) \in (S \cup \{i\}) \times (\Sigma \cup \{\varepsilon\}) \end{aligned}$$

**Ideia:** Se $w$ é aceite pelo AFD $A$, existe um percurso no diagrama de $A$ desde o estado $s_0$ até um estado final $f$. Se revertermos os arcos, podemos efetuar um percurso de $f$ até $s_0$ para consumir $w^R$ e aceitar $w^R$ se $s_0$ for final em $A'$. Como $A'$ só deve ter um estado inicial, criamos o estado $i$ e ligamo-lo aos que eram finais em $A$, para poder iniciar a pesquisa em qualquer $f \in F$.

# Autómato para a reversa $\mathcal{L}(A)^R$, sendo $A$ um AFD

A **palavra reversa de** $w$, denotada por $w^R$, é $w$ escrita da direita para a esquerda. Tal operação pode ser definida formalmente por: $\varepsilon^R = \varepsilon$ e $(ax)^R = x^R a$, para todo $x \in \Sigma^\star$ e $a \in \Sigma$. A **linguagem reversa de** $L$ é

$$L^R = \{w^R \mid w \in L\}$$

Dado um AFD $A = (S, \Sigma, \delta, s_0, F)$, **o seguinte AFND-$\varepsilon$ $A'$ reconhece** $\mathcal{L}(A)^R$:

$$A' = (S \cup \{i\}, \Sigma, \delta', i, \{s_0\})$$

onde $i$ é um estado novo e a função $\delta'$ de $(S \cup \{i\}) \times (\Sigma \cup \{\varepsilon\})$ é dada por

$$\begin{aligned} \delta'(i, \varepsilon) &= F \\ \delta'(s, a) &= \{s' \mid \delta(s', a) = s\}, \text{para todo } s \in S \text{ e } a \in \Sigma \\ \delta'(s, \alpha) &= \{\}, \text{para os restantes } (s, \alpha) \in (S \cup \{i\}) \times (\Sigma \cup \{\varepsilon\}) \end{aligned}$$

**Ideia:** Se $w$ é aceite pelo AFD $A$, existe um percurso no diagrama de $A$ desde o estado $s_0$ até um estado final $f$. Se revertermos os arcos, podemos efetuar um percurso de $f$ até $s_0$ para consumir $w^R$ e aceitar $w^R$ se $s_0$ for final em $A'$. Como $A'$ só deve ter um estado inicial, criamos o estado $i$ e ligamo-lo aos que eram finais em $A$, para poder iniciar a pesquisa em qualquer $f \in F$.

# Autómato para a reversa $\mathcal{L}(A)^R$, sendo $A$ um AFD

A **palavra reversa de** $w$, denotada por $w^R$, é $w$ escrita da direita para a esquerda. Tal operação pode ser definida formalmente por: $\varepsilon^R = \varepsilon$ e $(ax)^R = x^R a$, para todo $x \in \Sigma^\star$ e $a \in \Sigma$. A **linguagem reversa de** $L$ é

$$L^R = \{w^R \mid w \in L\}$$

Dado um AFD $A = (S, \Sigma, \delta, s_0, F)$, **o seguinte AFND-$\varepsilon$ $A'$ reconhece** $\mathcal{L}(A)^R$:

$$A' = (S \cup \{i\}, \Sigma, \delta', i, \{s_0\})$$

onde $i$ é um estado novo e a função $\delta'$ de $(S \cup \{i\}) \times (\Sigma \cup \{\varepsilon\})$ é dada por

$$\begin{aligned} \delta'(i, \varepsilon) &= F \\ \delta'(s, a) &= \{s' \mid \delta(s', a) = s\}, \text{para todo } s \in S \text{ e } a \in \Sigma \\ \delta'(s, \alpha) &= \{\}, \text{para os restantes } (s, \alpha) \in (S \cup \{i\}) \times (\Sigma \cup \{\varepsilon\}) \end{aligned}$$

**Ideia:** Se $w$ é aceite pelo AFD $A$, existe um percurso no diagrama de $A$ desde o estado $s_0$ até um estado final $f$. Se revertermos os arcos, podemos efetuar um percurso de $f$ até $s_0$ para consumir $w^R$ e aceitar $w^R$ se $s_0$ for final em $A'$. Como $A'$ só deve ter um estado inicial, criamos o estado $i$ e ligamo-lo aos que eram finais em $A$, para poder iniciar a pesquisa em qualquer $f \in F$.

# Autómato para a reversa $\mathcal{L}(A)^R$, sendo $A$ um AFD

A **palavra reversa de** $w$, denotada por $w^R$, é $w$ escrita da direita para a esquerda. Tal operação pode ser definida formalmente por: $\varepsilon^R = \varepsilon$ e $(ax)^R = x^R a$, para todo $x \in \Sigma^\star$ e $a \in \Sigma$. A **linguagem reversa de** $L$ é

$$L^R = \{w^R \mid w \in L\}$$

Dado um AFD $A = (S, \Sigma, \delta, s_0, F)$, **o seguinte AFND-$\varepsilon$ $A'$ reconhece** $\mathcal{L}(A)^R$:

$$A' = (S \cup \{i\}, \Sigma, \delta', i, \{s_0\})$$

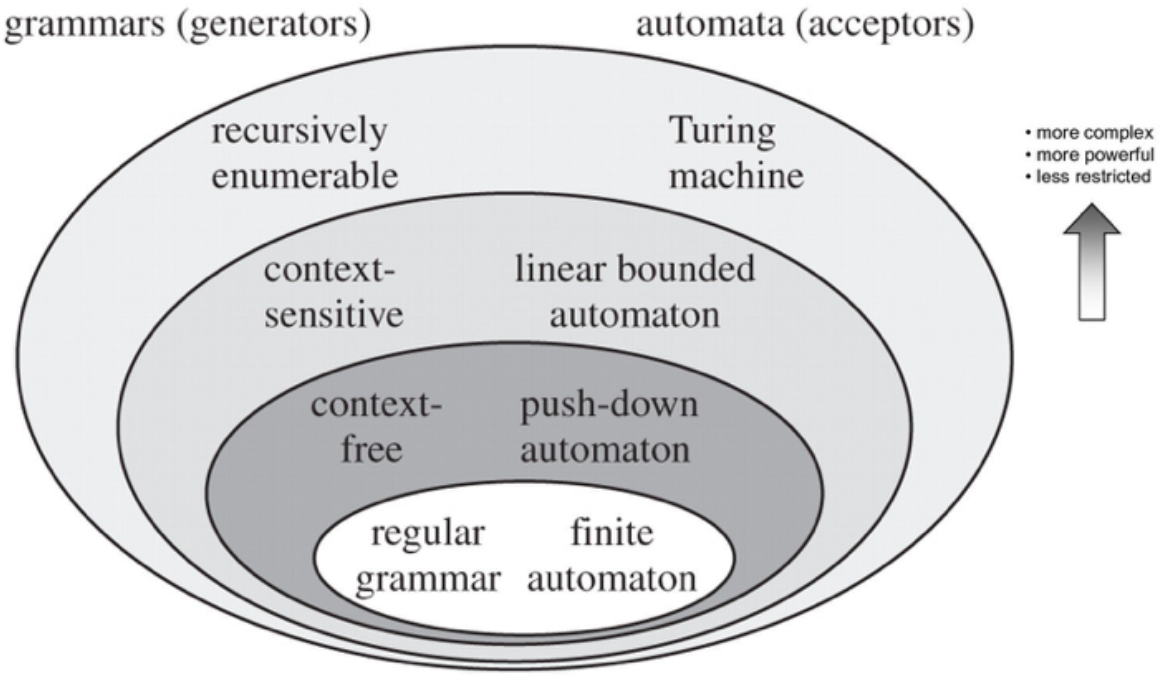
onde $i$ é um estado novo e a função $\delta'$ de $(S \cup \{i\}) \times (\Sigma \cup \{\varepsilon\})$ é dada por

$$\begin{aligned} \delta'(i, \varepsilon) &= F \\ \delta'(s, a) &= \{s' \mid \delta(s', a) = s\}, \text{para todo } s \in S \text{ e } a \in \Sigma \\ \delta'(s, \alpha) &= \{\}, \text{para os restantes } (s, \alpha) \in (S \cup \{i\}) \times (\Sigma \cup \{\varepsilon\}) \end{aligned}$$

**Ideia:** Se $w$ é aceite pelo AFD $A$, existe um percurso no diagrama de $A$ desde o estado $s_0$ até um estado final $f$. Se revertermos os arcos, podemos efetuar um percurso de $f$ até $s_0$ para consumir $w^R$ e aceitar $w^R$ se $s_0$ for final em $A'$. Como $A'$ só deve ter um estado inicial, criamos o estado $i$ e ligamo-lo aos que eram finais em $A$, para poder iniciar a pesquisa em qualquer $f \in F$.

# Fecho da classe das linguagens regulares para operações

O conjunto das linguagens regulares sobre $\Sigma$, isto é das que podem ser descritas por *expressões regulares* é uma classe na hierarquia de Chomsky. **Uma linguagem é regular se e só se pode ser reconhecida por um autómato finito**. Mais à frente, vamos ver que podem ser caraterizadas por *gramáticas regulares*.



Chomsky Hierarchy Levels. Source: Fitch. 2014.*

Fonte:`https://devopedia.org/chomsky-hierarchy`

# Fecho da classe das linguagens regulares para operações

**Proposição:** A classe das linguagens regulares sobre $\Sigma$ é fechada para as operações de união, intersecção, diferença, complementação, fecho de Kleene e reverso.

1. Se $L$ e $M$ são linguagens regulares então $L \cup M$ é regular.
2. Se $L$ e $M$ são linguagens regulares então $L \cap M$ é regular.
3. Se $L$ e $M$ são linguagens regulares então $L \setminus M$ é regular.
4. Se $L$ é regular então $\overline{L}$ é regular.
5. Se $L$ é regular então $L^\star$ é regular.
6. Se $L$ é regular então $L^R$ é regular.

# Fecho da classe das linguagens regulares para operações

**Prova:**

1. Se $L$ e $M$ são linguagens regulares então $L \cup M$ é regular.

   Se $r$ e $s$ são expressões regulares tais que $L = \mathcal{L}(r)$ e $M = \mathcal{L}(s)$, a expressão $(r + s)$ descreve $L \cup M$. □

2. Se $L$ e $M$ são linguagens regulares então $L \cap M$ é regular.

3. Se $L$ e $M$ são linguagens regulares então $L \setminus M$ é regular.

   Sejam $A_1$ e $A_2$ AFDs tais que $L = \mathcal{L}(A_1)$ e $M = \mathcal{L}(A_2)$. Podemos definir **AFDs produto** para reconhecer $L \cap M$ e para reconhecer $L \setminus M$. □

4. Se $L$ é regular então $\overline{L}$ é regular.

   Seja $A$ um AFD tal que $L = \mathcal{L}(A)$. Se trocarmos os estados finais por não finais (e vice-versa), teremos um AFD que reconhece $\overline{L}$. □

5. Se $L$ é regular então $L^\star$ é regular.

   Se $r$ é uma expressão regular com $L = \mathcal{L}(r)$, a expressão $(r^\star)$ descreve $L^\star$. □

6. Se $L$ é regular então $L^R$ é regular.

   Seja $A$ um AFD tal que $L = \mathcal{L}(A)$. Vimos que podiamos construir um AFND $\varepsilon$ que aceita $L(A)^R$. □

# Fecho da classe das linguagens regulares para operações

**Prova:**

1. Se $L$ e $M$ são linguagens regulares então $L \cup M$ é regular.

   Se $r$ e $s$ são expressões regulares tais que $L = \mathcal{L}(r)$ e $M = \mathcal{L}(s)$, a expressão $(r + s)$ descreve $L \cup M$. □

2. Se $L$ e $M$ são linguagens regulares então $L \cap M$ é regular.

3. Se $L$ e $M$ são linguagens regulares então $L \setminus M$ é regular.

   Sejam $A_1$ e $A_2$ AFDs tais que $L = \mathcal{L}(A_1)$ e $M = \mathcal{L}(A_2)$. Podemos definir **AFDs produto** para reconhecer $L \cap M$ e para reconhecer $L \setminus M$. □

4. Se $L$ é regular então $\overline{L}$ é regular.

   Seja $A$ um AFD tal que $L = \mathcal{L}(A)$. Se trocarmos os estados finais por não finais (e vice-versa), teremos um AFD que reconhece $\overline{L}$. □

5. Se $L$ é regular então $L^\star$ é regular.

   Se $r$ é uma expressão regular com $L = \mathcal{L}(r)$, a expressão $(r^\star)$ descreve $L^\star$. □

6. Se $L$ é regular então $L^R$ é regular.

   Seja $A$ um AFD tal que $L = \mathcal{L}(A)$. Vimos que podiamos construir um AFND $\varepsilon$ que aceita $L(A)^R$. □

# Fecho da classe das linguagens regulares para operações

**Prova:**

1. Se $L$ e $M$ são linguagens regulares então $L \cup M$ é regular.

   Se $r$ e $s$ são expressões regulares tais que $L = \mathcal{L}(r)$ e $M = \mathcal{L}(s)$, a expressão $(r + s)$ descreve $L \cup M$. □

2. Se $L$ e $M$ são linguagens regulares então $L \cap M$ é regular.

3. Se $L$ e $M$ são linguagens regulares então $L \setminus M$ é regular.

   Sejam $A_1$ e $A_2$ AFDs tais que $L = \mathcal{L}(A_1)$ e $M = \mathcal{L}(A_2)$. Podemos definir **AFDs produto** para reconhecer $L \cap M$ e para reconhecer $L \setminus M$. □

4. Se $L$ é regular então $\overline{L}$ é regular.

   Seja $A$ um AFD tal que $L = \mathcal{L}(A)$. Se trocarmos os estados finais por não finais (e vice-versa), teremos um AFD que reconhece $\overline{L}$. □

5. Se $L$ é regular então $L^\star$ é regular.

   Se $r$ é uma expressão regular com $L = \mathcal{L}(r)$, a expressão $(r^\star)$ descreve $L^\star$. □

6. Se $L$ é regular então $L^R$ é regular.

   Seja $A$ um AFD tal que $L = \mathcal{L}(A)$. Vimos que podiamos construir um AFND $\varepsilon$ que aceita $L(A)^R$. □

# Outras provas

- Como a classe das linguagens regulares é **fechada para a união e para a complementação**, podemos concluir que é **fechada para a intersecção** pois, pelas leis de De Morgan,

$$L \cap M = \overline{\overline{L} \cup \overline{M}}$$

Portanto, se $L$ e $M$ são regulares, então $\overline{L}$ e $\overline{M}$ são regulares.

Consequentemente, $\overline{L} \cup \overline{M}$ é regular.

E, portanto, $\overline{\overline{L} \cup \overline{M}}$ é regular.

- Definimos a **reversa $\mathcal{R}(.)$ de uma expressão regular** assim:
  - $\mathcal{R}(\varepsilon) = \varepsilon$, $\mathcal{R}(\emptyset) = \emptyset$, $\mathcal{R}(a) = a$, com $a \in \Sigma$
  - $\mathcal{R}((rs)) = (\mathcal{R}(s)\mathcal{R}(r))$
  - $\mathcal{R}((r + s)) = (\mathcal{R}(r) + \mathcal{R}(s))$
  - $\mathcal{R}((r^\star)) = (\mathcal{R}(s)^\star)$

Se $r$ descreve $L$ então $\mathcal{R}(r)$ descreve $L^R$.

# União Finita vs União Infinita

**Proposição:** A classe das linguagens regulares é fechada para a união finita mas não para a união infinita:

1. Se $L_1, L_2, \ldots L_n$ são regulares, com $n \geq 2$, constante, então $\bigcup_{i=1}^{n} L_i$ é regular.
2. Existe uma família $\{L_i\}_{i \in \mathbb{N}}$ de linguagens regulares tal que $\bigcup_{i \in \mathbb{N}} L_i$ **não** é regular.

**Prova:**

1. Por indução matemática.
   (i) **Caso de base (n=2):** Vimos que se $L_1$ e $L_2$ são regulares, $L_1 \cup L_2$ é regular.
   (ii) **Hereditariedade:** Para todo $k \geq 2$, se $\bigcup_{i=1}^{k} L_i$ é regular, quaisquer que sejam $L_1, L_2, \ldots L_k$ regulares, então $\bigcup_{i=1}^{k+1} M_i$ é regular, quaisquer que sejam $M_1, M_2, \ldots M_k, M_{k+1}$ regulares. De facto, como a união de duas linguagens regulares é regular e, por hipótese de indução, $\bigcup_{i=1}^{k} M_i$ é regular, concluimos que $\bigcup_{i=1}^{k+1} M_i = (\bigcup_{i=1}^{k} M_i) \cup M_{k+1}$ é regular.
   Portanto, pelo princípio de indução matemática, de (i) e (ii) segue que a união de $n$ linguagens regulares é regular, qualquer que seja $n \geq 2$. □
2. A linguagem $\{0^n 1^n \mid n \in \mathbb{N}\} = \bigcup_{n \in \mathbb{N}} \{0^n 1^n\}$ não é regular, mas cada uma das linguagens $\{0^n 1^n\}$ é regular, pois só tem uma palavra. Se $|\Sigma| = 1$, podiamos tomar $\{0^p \mid p \text{ primo}\}$. □

# União Finita vs União Infinita

**Proposição:** A classe das linguagens regulares é fechada para a união finita mas não para a união infinita:

1. Se $L_1, L_2, \ldots L_n$ são regulares, com $n \geq 2$, constante, então $\bigcup_{i=1}^{n} L_i$ é regular.
2. Existe uma família $\{L_i\}_{i \in \mathbb{N}}$ de linguagens regulares tal que $\bigcup_{i \in \mathbb{N}} L_i$ **não** é regular.

**Prova:**

1. Por indução matemática.

   (i) **Caso de base (n=2):** Vimos que se $L_1$ e $L_2$ são regulares, $L_1 \cup L_2$ é regular.

   (ii) **Hereditariedade:** Para todo $k \geq 2$, se $\bigcup_{i=1}^{k} L_i$ é regular, quaisquer que sejam $L_1, L_2, \ldots L_k$ regulares, então $\bigcup_{i=1}^{k+1} M_i$ é regular, quaisquer que sejam $M_1, M_2, \ldots M_k, M_{k+1}$ regulares. De facto, como a união de duas linguagens regulares é regular e, por hipótese de indução, $\bigcup_{i=1}^{k} M_i$ é regular, concluimos que $\bigcup_{i=1}^{k+1} M_i = (\bigcup_{i=1}^{k} M_i) \cup M_{k+1}$ é regular.

   Portanto, pelo princípio de indução matemática, de (i) e (ii) segue que a união de $n$ linguagens regulares é regular, qualquer que seja $n \geq 2$. □

2. A linguagem $\{0^n 1^n \mid n \in \mathbb{N}\} = \bigcup_{n \in \mathbb{N}} \{0^n 1^n\}$ não é regular, mas cada uma das linguagens $\{0^n 1^n\}$ é regular, pois só tem uma palavra. Se $|\Sigma| = 1$, podiamos tomar $\{0^p \mid p \text{ primo}\}$. □

# União Finita vs União Infinita

**Proposição:** A classe das linguagens regulares é fechada para a união finita mas não para a união infinita:

1. Se $L_1, L_2, \ldots L_n$ são regulares, com $n \geq 2$, constante, então $\bigcup_{i=1}^{n} L_i$ é regular.
2. Existe uma família $\{L_i\}_{i \in \mathbb{N}}$ de linguagens regulares tal que $\bigcup_{i \in \mathbb{N}} L_i$ **não** é regular.

**Prova:**

1. Por indução matemática.

   (i) **Caso de base (n=2):** Vimos que se $L_1$ e $L_2$ são regulares, $L_1 \cup L_2$ é regular.

   (ii) **Hereditariedade:** Para todo $k \geq 2$, se $\bigcup_{i=1}^{k} L_i$ é regular, quaisquer que sejam $L_1, L_2, \ldots L_k$ regulares, então $\bigcup_{i=1}^{k+1} M_i$ é regular, quaisquer que sejam $M_1, M_2, \ldots M_k, M_{k+1}$ regulares. De facto, como a união de duas linguagens regulares é regular e, por hipótese de indução, $\bigcup_{i=1}^{k} M_i$ é regular, concluimos que $\bigcup_{i=1}^{k+1} M_i = (\bigcup_{i=1}^{k} M_i) \cup M_{k+1}$ é regular.

   Portanto, pelo princípio de indução matemática, de (i) e (ii) segue que a união de $n$ linguagens regulares é regular, qualquer que seja $n \geq 2$. □

2. A linguagem $\{0^n 1^n \mid n \in \mathbb{N}\} = \bigcup_{n \in \mathbb{N}} \{0^n 1^n\}$ não é regular, mas cada uma das linguagens $\{0^n 1^n\}$ é regular, pois só tem uma palavra. Se $|\Sigma| = 1$, podiamos tomar $\{0^p \mid p \text{ primo}\}$. □

# União Finita vs União Infinita

**Proposição:** A classe das linguagens regulares é fechada para a união finita mas não para a união infinita:

1. Se $L_1, L_2, \ldots L_n$ são regulares, com $n \geq 2$, constante, então $\bigcup_{i=1}^{n} L_i$ é regular.
2. Existe uma família $\{L_i\}_{i \in \mathbb{N}}$ de linguagens regulares tal que $\bigcup_{i \in \mathbb{N}} L_i$ **não** é regular.

**Prova:**

1. Por indução matemática.

   (i) **Caso de base (n=2):** Vimos que se $L_1$ e $L_2$ são regulares, $L_1 \cup L_2$ é regular.

   (ii) **Hereditariedade:** Para todo $k \geq 2$, se $\bigcup_{i=1}^{k} L_i$ é regular, quaisquer que sejam $L_1, L_2, \ldots L_k$ regulares, então $\bigcup_{i=1}^{k+1} M_i$ é regular, quaisquer que sejam $M_1, M_2, \ldots M_k, M_{k+1}$ regulares. De facto, como a união de duas linguagens regulares é regular e, por hipótese de indução, $\bigcup_{i=1}^{k} M_i$ é regular, concluimos que $\bigcup_{i=1}^{k+1} M_i = (\bigcup_{i=1}^{k} M_i) \cup M_{k+1}$ é regular.

   Portanto, pelo princípio de indução matemática, de (i) e (ii) segue que a união de $n$ linguagens regulares é regular, qualquer que seja $n \geq 2$. □

2. A linguagem $\{0^n 1^n \mid n \in \mathbb{N}\} = \bigcup_{n \in \mathbb{N}} \{0^n 1^n\}$ não é regular, mas cada uma das linguagens $\{0^n 1^n\}$ é regular, pois só tem uma palavra. Se $|\Sigma| = 1$, podiamos tomar $\{0^p \mid p \text{ primo}\}$. □

# União Finita vs União Infinita

**Proposição:** A classe das linguagens regulares é fechada para a união finita mas não para a união infinita:

1. Se $L_1, L_2, \ldots L_n$ são regulares, com $n \geq 2$, constante, então $\bigcup_{i=1}^{n} L_i$ é regular.
2. Existe uma família $\{L_i\}_{i \in \mathbb{N}}$ de linguagens regulares tal que $\bigcup_{i \in \mathbb{N}} L_i$ **não** é regular.

**Prova:**

1. Por indução matemática.

   (i) **Caso de base (n=2):** Vimos que se $L_1$ e $L_2$ são regulares, $L_1 \cup L_2$ é regular.

   (ii) **Hereditariedade:** Para todo $k \geq 2$, se $\bigcup_{i=1}^{k} L_i$ é regular, quaisquer que sejam $L_1, L_2, \ldots L_k$ regulares, então $\bigcup_{i=1}^{k+1} M_i$ é regular, quaisquer que sejam $M_1, M_2, \ldots M_k, M_{k+1}$ regulares. De facto, como a união de duas linguagens regulares é regular e, por hipótese de indução, $\bigcup_{i=1}^{k} M_i$ é regular, concluimos que $\bigcup_{i=1}^{k+1} M_i = (\bigcup_{i=1}^{k} M_i) \cup M_{k+1}$ é regular.

   Portanto, pelo princípio de indução matemática, de (i) e (ii) segue que a união de $n$ linguagens regulares é regular, qualquer que seja $n \geq 2$. □

2. A linguagem $\{0^n 1^n \mid n \in \mathbb{N}\} = \bigcup_{n \in \mathbb{N}} \{0^n 1^n\}$ não é regular, mas cada uma das linguagens $\{0^n 1^n\}$ é regular, pois só tem uma palavra. Se $|\Sigma| = 1$, podiamos tomar $\{0^p \mid p \text{ primo}\}$. □

# União Finita vs União Infinita

**Proposição:** A classe das linguagens regulares é fechada para a união finita mas não para a união infinita:

1. Se $L_1, L_2, \ldots L_n$ são regulares, com $n \geq 2$, constante, então $\bigcup_{i=1}^{n} L_i$ é regular.
2. Existe uma família $\{L_i\}_{i \in \mathbb{N}}$ de linguagens regulares tal que $\bigcup_{i \in \mathbb{N}} L_i$ **não** é regular.

**Prova:**

1. Por indução matemática.

   (i) **Caso de base (n=2):** Vimos que se $L_1$ e $L_2$ são regulares, $L_1 \cup L_2$ é regular.

   (ii) **Hereditariedade:** Para todo $k \geq 2$, se $\bigcup_{i=1}^{k} L_i$ é regular, quaisquer que sejam $L_1, L_2, \ldots L_k$ regulares, então $\bigcup_{i=1}^{k+1} M_i$ é regular, quaisquer que sejam $M_1, M_2, \ldots M_k, M_{k+1}$ regulares. De facto, como a união de duas linguagens regulares é regular e, por hipótese de indução, $\bigcup_{i=1}^{k} M_i$ é regular, concluimos que $\bigcup_{i=1}^{k+1} M_i = (\bigcup_{i=1}^{k} M_i) \cup M_{k+1}$ é regular.

   Portanto, pelo princípio de indução matemática, de (i) e (ii) segue que a união de $n$ linguagens regulares é regular, qualquer que seja $n \geq 2$. □

2. A linguagem $\{0^n 1^n \mid n \in \mathbb{N}\} = \bigcup_{n \in \mathbb{N}} \{0^n 1^n\}$ não é regular, mas cada uma das linguagens $\{0^n 1^n\}$ é regular, pois só tem uma palavra. Se $|\Sigma| = 1$, podiamos tomar $\{0^p \mid p \text{ primo}\}$. □

# União Finita vs União Infinita

**Proposição:** A classe das linguagens regulares é fechada para a união finita mas não para a união infinita:

1. Se $L_1, L_2, \ldots L_n$ são regulares, com $n \geq 2$, constante, então $\bigcup_{i=1}^{n} L_i$ é regular.
2. Existe uma família $\{L_i\}_{i \in \mathbb{N}}$ de linguagens regulares tal que $\bigcup_{i \in \mathbb{N}} L_i$ **não** é regular.

**Prova:**

1. Por indução matemática.

   (i) **Caso de base (n=2):** Vimos que se $L_1$ e $L_2$ são regulares, $L_1 \cup L_2$ é regular.

   (ii) **Hereditariedade:** Para todo $k \geq 2$, se $\bigcup_{i=1}^{k} L_i$ é regular, quaisquer que sejam $L_1, L_2, \ldots L_k$ regulares, então $\bigcup_{i=1}^{k+1} M_i$ é regular, quaisquer que sejam $M_1, M_2, \ldots M_k, M_{k+1}$ regulares. De facto, como a união de duas linguagens regulares é regular e, por hipótese de indução, $\bigcup_{i=1}^{k} M_i$ é regular, concluimos que $\bigcup_{i=1}^{k+1} M_i = (\bigcup_{i=1}^{k} M_i) \cup M_{k+1}$ é regular.

   Portanto, pelo princípio de indução matemática, de (i) e (ii) segue que a união de $n$ linguagens regulares é regular, qualquer que seja $n \geq 2$. □
2. A linguagem $\{0^n 1^n \mid n \in \mathbb{N}\} = \bigcup_{n \in \mathbb{N}} \{0^n 1^n\}$ não é regular, mas cada uma das linguagens $\{0^n 1^n\}$ é regular, pois só tem uma palavra. Se $|\Sigma| = 1$, podiamos tomar $\{0^p \mid p \text{ primo}\}$. □

# União Finita vs União Infinita

**Proposição:** A classe das linguagens regulares é fechada para a união finita mas não para a união infinita:

1. Se $L_1, L_2, \ldots L_n$ são regulares, com $n \geq 2$, constante, então $\bigcup_{i=1}^{n} L_i$ é regular.
2. Existe uma família $\{L_i\}_{i \in \mathbb{N}}$ de linguagens regulares tal que $\bigcup_{i \in \mathbb{N}} L_i$ **não** é regular.

**Prova:**

1. Por indução matemática.

   (i) **Caso de base (n=2):** Vimos que se $L_1$ e $L_2$ são regulares, $L_1 \cup L_2$ é regular.

   (ii) **Hereditariedade:** Para todo $k \geq 2$, se $\bigcup_{i=1}^{k} L_i$ é regular, quaisquer que sejam $L_1, L_2, \ldots L_k$ regulares, então $\bigcup_{i=1}^{k+1} M_i$ é regular, quaisquer que sejam $M_1, M_2, \ldots M_k, M_{k+1}$ regulares. De facto, como a união de duas linguagens regulares é regular e, por hipótese de indução, $\bigcup_{i=1}^{k} M_i$ é regular, concluimos que $\bigcup_{i=1}^{k+1} M_i = (\bigcup_{i=1}^{k} M_i) \cup M_{k+1}$ é regular.

   Portanto, pelo princípio de indução matemática, de (i) e (ii) segue que a união de $n$ linguagens regulares é regular, qualquer que seja $n \geq 2$. □
2. A linguagem $\{0^n 1^n \mid n \in \mathbb{N}\} = \bigcup_{n \in \mathbb{N}} \{0^n 1^n\}$ não é regular, mas cada uma das linguagens $\{0^n 1^n\}$ é regular, pois só tem uma palavra. Se $|\Sigma| = 1$, podiamos tomar $\{0^p \mid p \text{ primo}\}$. □