# CC1004 - Modelos de Computação
# Teóricas 23

Ana Paula Tomás

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

Maio 2021

# Existem linguagens que não são LICs

Qualquer que seja o alfabeto $\Sigma$, existem linguagens de alfabeto $\Sigma$ que não são independentes de contexto.

**Exemplos de linguagens que não são LICs:**

$$\{a^n b^n c^n \mid n \in \mathbb{N}\} \qquad \{ww \mid w \in \{a, b\}^\star\}$$

$$\{a^p \mid p \text{ primo}\} \qquad \{a^{n^2} \mid n \in \mathbb{N}\}$$

Nenhuma satisfaz a condição definida pelo Lema da Repetição para LIC.

**Lema da Repetição para LICs**

Se $L$ é uma LIC então existe uma constante $n \in \mathbb{Z}^+$, só dependente de $L$, tal que qualquer que seja $z \in L$, se $|z| \geq n$ então podemos escrever $z$ como $uvwxy$ de forma que $|vx| \geq 1$, $|vwx| \leq n$ e, para todo $i \in \mathbb{N}$, se tem $uv^i wx^i y \in L$.

Na próxima aula, veremos a prova e como o podemos aplicar...

# União, concatenação e fecho de Kleene de LICs

A classe das LICs é fechada para a união (finita), concatenação, reverso, e fecho de Kleene. Não é fechada para a interseção e para a complementação.

Sejam $G_1 = (V_1, \Sigma, P_1, S_1)$ e $G_2 = (V_2, \Sigma, P_2, S_2)$ GICs, tais que $V_1 \cap V_2 = \emptyset$.

- $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$ gera $\mathcal{L}(G_1) \cup \mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to S_1 S_2\}, S)$ gera $\mathcal{L}(G_1)\mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to \varepsilon, S \to S_1 S\}, S)$ gera $\mathcal{L}(G_1)^\star$. $S$ é um símbolo novo.
- $G = (V_1, \Sigma, \{X \to \gamma^R \mid (X \to \gamma) \in P_1\}, S_1)$ gera $\mathcal{L}(G_1)^R$, sendo $\gamma^R$ o reverso de $\gamma$.
- Existem LICs $L_1$ e $L_2$ tais que $L_1 \cap L_2$ não é LIC.
  Por exemplo, $L_1 = \{a^m b^m \mid m \in \mathbb{N}\}$ e $L_2 = \{b^k c^k \mid k \in \mathbb{N}\}$ são LICs mas $L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ não é LIC.
- Existem LICs cujo complementar não é LIC. Caso contrário, como $L \cap M = \overline{\overline{L} \cup \overline{M}}$, a interseção de LICs seria LIC.

# União, concatenação e fecho de Kleene de LICs

A classe das LICs é fechada para a união (finita), concatenação, reverso, e fecho de Kleene. Não é fechada para a interseção e para a complementação.

Sejam $G_1 = (V_1, \Sigma, P_1, S_1)$ e $G_2 = (V_2, \Sigma, P_2, S_2)$ GICs, tais que $V_1 \cap V_2 = \emptyset$.

- $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$ gera $\mathcal{L}(G_1) \cup \mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to S_1 S_2\}, S)$ gera $\mathcal{L}(G_1)\mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to \varepsilon, S \to S_1 S\}, S)$ gera $\mathcal{L}(G_1)^\star$. $S$ é um símbolo novo.
- $G = (V_1, \Sigma, \{X \to \gamma^R \mid (X \to \gamma) \in P_1\}, S_1)$ gera $\mathcal{L}(G_1)^R$, sendo $\gamma^R$ o reverso de $\gamma$.
- Existem LICs $L_1$ e $L_2$ tais que $L_1 \cap L_2$ não é LIC.
  Por exemplo, $L_1 = \{a^m b^m \mid m \in \mathbb{N}\}$ e $L_2 = \{b^k c^k \mid k \in \mathbb{N}\}$ são LICs mas $L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ não é LIC.
- Existem LICs cujo complementar não é LIC. Caso contrário, como $L \cap M = \overline{\overline{L} \cup \overline{M}}$, a interseção de LICs seria LIC.

# União, concatenação e fecho de Kleene de LICs

A classe das LICs é fechada para a união (finita), concatenação, reverso, e fecho de Kleene. Não é fechada para a interseção e para a complementação.

Sejam $G_1 = (V_1, \Sigma, P_1, S_1)$ e $G_2 = (V_2, \Sigma, P_2, S_2)$ GICs, tais que $V_1 \cap V_2 = \emptyset$.

- $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$ gera $\mathcal{L}(G_1) \cup \mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to S_1 S_2\}, S)$ gera $\mathcal{L}(G_1)\mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to \varepsilon, S \to S_1 S\}, S)$ gera $\mathcal{L}(G_1)^\star$. $S$ é um símbolo novo.
- $G = (V_1, \Sigma, \{X \to \gamma^R \mid (X \to \gamma) \in P_1\}, S_1)$ gera $\mathcal{L}(G_1)^R$, sendo $\gamma^R$ o reverso de $\gamma$.
- Existem LICs $L_1$ e $L_2$ tais que $L_1 \cap L_2$ não é LIC.
  Por exemplo, $L_1 = \{a^m b^m \mid m \in \mathbb{N}\}$ e $L_2 = \{b^k c^k \mid k \in \mathbb{N}\}$ são LICs mas $L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ não é LIC.
- Existem LICs cujo complementar não é LIC. Caso contrário, como $L \cap M = \overline{\overline{L} \cup \overline{M}}$, a interseção de LICs seria LIC.

# União, concatenação e fecho de Kleene de LICs

A classe das LICs é fechada para a união (finita), concatenação, reverso, e fecho de Kleene. Não é fechada para a interseção e para a complementação.

Sejam $G_1 = (V_1, \Sigma, P_1, S_1)$ e $G_2 = (V_2, \Sigma, P_2, S_2)$ GICs, tais que $V_1 \cap V_2 = \emptyset$.

- $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$ gera $\mathcal{L}(G_1) \cup \mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to S_1 S_2\}, S)$ gera $\mathcal{L}(G_1)\mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to \varepsilon, S \to S_1 S\}, S)$ gera $\mathcal{L}(G_1)^\star$. $S$ é um símbolo novo.
- $G = (V_1, \Sigma, \{X \to \gamma^R \mid (X \to \gamma) \in P_1\}, S_1)$ gera $\mathcal{L}(G_1)^R$, sendo $\gamma^R$ o reverso de $\gamma$.
- Existem LICs $L_1$ e $L_2$ tais que $L_1 \cap L_2$ não é LIC.
  Por exemplo, $L_1 = \{a^m b^m \mid m \in \mathbb{N}\}$ e $L_2 = \{b^k c^k \mid k \in \mathbb{N}\}$ são LICs mas $L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ não é LIC.
- Existem LICs cujo complementar não é LIC. Caso contrário, como $L \cap M = \overline{\overline{L} \cup \overline{M}}$, a interseção de LICs seria LIC.

# União, concatenação e fecho de Kleene de LICs

A classe das LICs é fechada para a união (finita), concatenação, reverso, e fecho de Kleene. Não é fechada para a interseção e para a complementação.

Sejam $G_1 = (V_1, \Sigma, P_1, S_1)$ e $G_2 = (V_2, \Sigma, P_2, S_2)$ GICs, tais que $V_1 \cap V_2 = \emptyset$.

- $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$ gera $\mathcal{L}(G_1) \cup \mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to S_1 S_2\}, S)$ gera $\mathcal{L}(G_1)\mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to \varepsilon, S \to S_1 S\}, S)$ gera $\mathcal{L}(G_1)^\star$. $S$ é um símbolo novo.
- $G = (V_1, \Sigma, \{X \to \gamma^R \mid (X \to \gamma) \in P_1\}, S_1)$ gera $\mathcal{L}(G_1)^R$, sendo $\gamma^R$ o reverso de $\gamma$.
- Existem LICs $L_1$ e $L_2$ tais que $L_1 \cap L_2$ não é LIC.
  Por exemplo, $L_1 = \{a^m b^m \mid m \in \mathbb{N}\}$ e $L_2 = \{b^k c^k \mid k \in \mathbb{N}\}$ são LICs mas $L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ não é LIC.
- Existem LICs cujo complementar não é LIC. Caso contrário, como $L \cap M = \overline{\overline{L} \cup \overline{M}}$, a interseção de LICs seria LIC.

# União, concatenação e fecho de Kleene de LICs

A classe das LICs é fechada para a união (finita), concatenação, reverso, e fecho de Kleene. Não é fechada para a interseção e para a complementação.

Sejam $G_1 = (V_1, \Sigma, P_1, S_1)$ e $G_2 = (V_2, \Sigma, P_2, S_2)$ GICs, tais que $V_1 \cap V_2 = \emptyset$.

- $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$ gera $\mathcal{L}(G_1) \cup \mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to S_1 S_2\}, S)$ gera $\mathcal{L}(G_1)\mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to \varepsilon, S \to S_1 S\}, S)$ gera $\mathcal{L}(G_1)^\star$. $S$ é um símbolo novo.
- $G = (V_1, \Sigma, \{X \to \gamma^R \mid (X \to \gamma) \in P_1\}, S_1)$ gera $\mathcal{L}(G_1)^R$, sendo $\gamma^R$ o reverso de $\gamma$.
- Existem LICs $L_1$ e $L_2$ tais que $L_1 \cap L_2$ não é LIC.
  Por exemplo, $L_1 = \{a^m b^m \mid m \in \mathbb{N}\}$ e $L_2 = \{b^k c^k \mid k \in \mathbb{N}\}$ são LICs mas $L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ não é LIC.
- Existem LICs cujo complementar não é LIC. Caso contrário, como $L \cap M = \overline{\overline{L} \cup \overline{M}}$, a interseção de LICs seria LIC.

# União, concatenação e fecho de Kleene de LICs

A classe das LICs é fechada para a união (finita), concatenação, reverso, e fecho de Kleene. Não é fechada para a interseção e para a complementação.

Sejam $G_1 = (V_1, \Sigma, P_1, S_1)$ e $G_2 = (V_2, \Sigma, P_2, S_2)$ GICs, tais que $V_1 \cap V_2 = \emptyset$.

- $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$ gera $\mathcal{L}(G_1) \cup \mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to S_1 S_2\}, S)$ gera $\mathcal{L}(G_1)\mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to \varepsilon, S \to S_1 S\}, S)$ gera $\mathcal{L}(G_1)^\star$. $S$ é um símbolo novo.
- $G = (V_1, \Sigma, \{X \to \gamma^R \mid (X \to \gamma) \in P_1\}, S_1)$ gera $\mathcal{L}(G_1)^R$, sendo $\gamma^R$ o reverso de $\gamma$.
- Existem LICs $L_1$ e $L_2$ tais que $L_1 \cap L_2$ não é LIC.
  Por exemplo, $L_1 = \{\mathtt{a}^m\mathtt{b}^m \mid m \in \mathbb{N}\}$ e $L_2 = \{\mathtt{b}^k\mathtt{c}^k \mid k \in \mathbb{N}\}$ são LICs mas $L_1 \cap L_2 = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \in \mathbb{N}\}$ não é LIC.
- Existem LICs cujo complementar não é LIC. Caso contrário, como $L \cap M = \overline{\overline{L} \cup \overline{M}}$, a interseção de LICs seria LIC.

# União, concatenação e fecho de Kleene de LICs

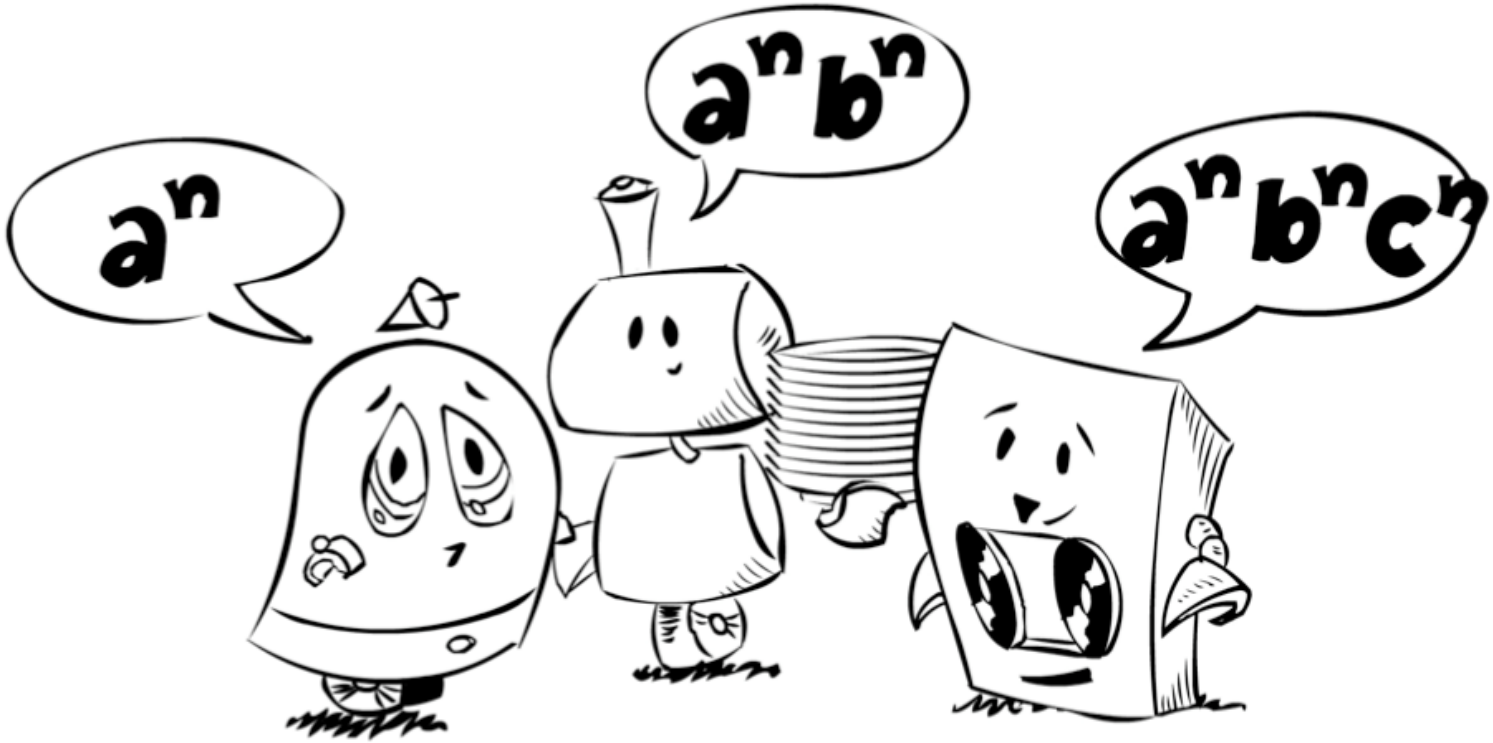
A classe das LICs é fechada para a união (finita), concatenação, reverso, e fecho de Kleene. Não é fechada para a interseção e para a complementação.

Sejam $G_1 = (V_1, \Sigma, P_1, S_1)$ e $G_2 = (V_2, \Sigma, P_2, S_2)$ GICs, tais que $V_1 \cap V_2 = \emptyset$.

- $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$ gera $\mathcal{L}(G_1) \cup \mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to S_1 S_2\}, S)$ gera $\mathcal{L}(G_1)\mathcal{L}(G_2)$. $S$ é um símbolo novo.
- $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to \varepsilon, S \to S_1 S\}, S)$ gera $\mathcal{L}(G_1)^\star$. $S$ é um símbolo novo.
- $G = (V_1, \Sigma, \{X \to \gamma^R \mid (X \to \gamma) \in P_1\}, S_1)$ gera $\mathcal{L}(G_1)^R$, sendo $\gamma^R$ o reverso de $\gamma$.
- Existem LICs $L_1$ e $L_2$ tais que $L_1 \cap L_2$ não é LIC.
  Por exemplo, $L_1 = \{\mathtt{a}^m\mathtt{b}^m \mid m \in \mathbb{N}\}$ e $L_2 = \{\mathtt{b}^k\mathtt{c}^k \mid k \in \mathbb{N}\}$ são LICs mas $L_1 \cap L_2 = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \in \mathbb{N}\}$ não é LIC.
- Existem LICs cujo complementar não é LIC. Caso contrário, como $L \cap M = \overline{\overline{L} \cup \overline{M}}$, a interseção de LICs seria LIC.

# Linguagens Formais e Computabilidade

Área de TCS. Limites da computação. Computabilidade: **existe algoritmo para resolução do problema?** Complexidade: **Que recursos requer** (tempo e espaço)? Como se **descreve o problema**? Que **tipo de máquina** usará?



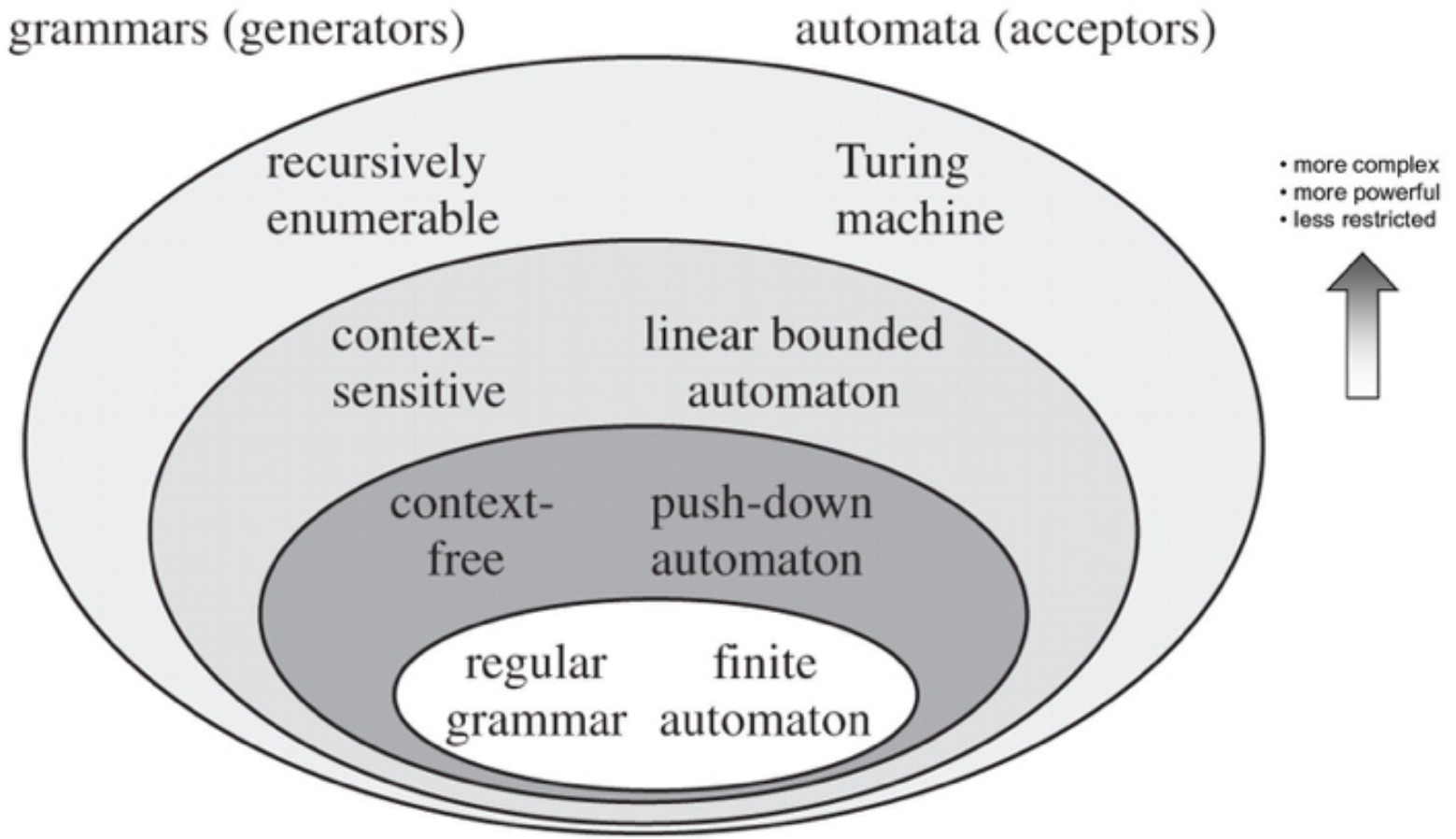

Fonte:http://www.ic.uff.br/~ueverton/files/LF/aula09.pdf

# Recordando o Programa de CC1004

- Noção de linguagem formal.
- Linguagens regulares.
  - Expressões regulares.
  - Autómatos finitos determinísticos e não determinísticos.
  - Propriedades. Lema da repetição para linguagens regulares.
- Linguagens independentes de contexto (LICs).
  - Gramáticas independentes de contexto.
  - Autómatos de pilha.
  - Propriedades. Lema da repetição para LICs.
- Linguagens recursivamente enumeráveis. Máquinas de Turing e noção de computabilidade.

**No fim desta UC deverá ser capaz de** especificar linguagens formais simples, usando formas de descrição alternativas, e determinar a sua classificação na hierarquia de poder computacional (*hierarquia de Chomsky*).

# Hierarquia de Chomsky




Chomsky Hierarchy Levels. Source: Fitch. 2014.*

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

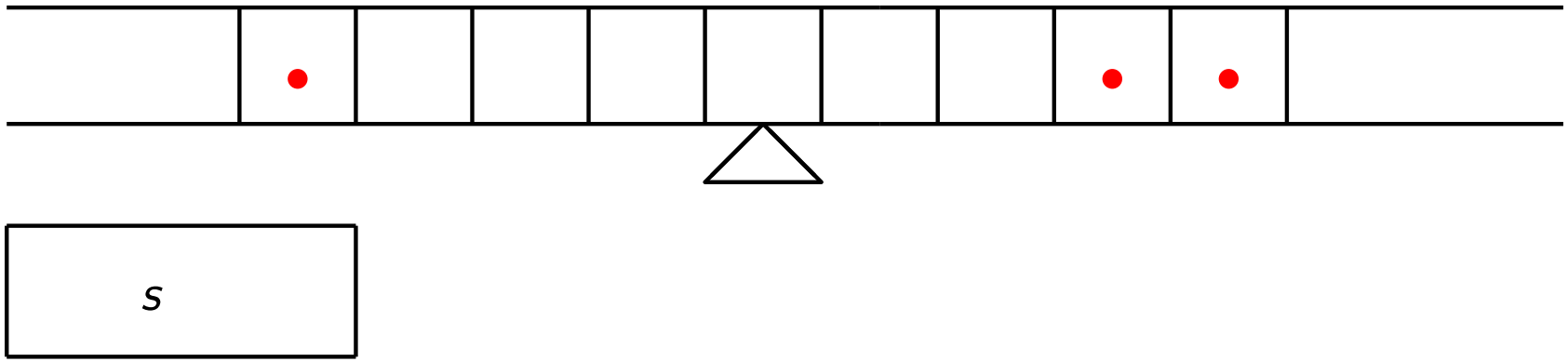
# Máquinas de Turing

A **máquina de Turing** (1936) foi proposta como um modelo matemático para "métodos algorítmicos". **Intuitivamente**, um método algorítmico é *"uma qualquer sequência finita de instruções que possa ser executada mecanicamente"*.

- Esta **noção não tem precisão matemática**, pelo que não se conseguiu demonstrar que seja equivalente ao modelo (formal) de máquina de Turing.
- As máquinas de Turing são tão gerais que parecem poder simular qualquer computação possível.
- Foram propostos outros modelos de computação com o mesmo objetivo, mas nenhum é mais geral do que a máquina de Turing (alguns são equivalentes).
- Embora não tenha sido demonstrada, aceita-se a **conjetura de Church–Turing** que diz que existe um método algorítmico para resolver um dado problema se e só se existe uma máquina de Turing para o resolver.
- Também se sabe que existem problemas que não podem ser resolvidos por máquinas de Turing.

# Máquinas de Turing

Existem **vários modelos de máquinas de Turing**. O que vamos descrever pode ser visto como o modelo de uma máquina que dispõe de uma **fita dividida em células**. A fita é **infinita** para a esquerda e para a direita.




A máquina tem um conjunto de estados **finito** e uma cabeça de leitura/escrita, que pode deslocar para a esquerda ou para a direita. Pode **ler** e **escrever** símbolos na fita. Cada célula contém um símbolo. Como a fita é infinita, usa-se um **símbolo branco**, para delimitar a parte da fita que contém a informação relevante. Na figura, é •.

# Máquinas de Turing

Uma **máquina de Turing** $\mathcal{M}$ é um sistema $\mathcal{M} = (S, \Sigma, \Gamma, \delta, s_0, b, F)$, com

- $S$ – **conjunto de estados**, $S$ é finito,
- $\Gamma$ – **alfabeto da fita**, $\Sigma \subset \Gamma$ – **alfabeto de entrada**,
- $s_0 \in S$ – **estado inicial**,
- $b \in \Gamma$ – **símbolo branco**,
- $F \subseteq S$ – conjunto de **estados finais**,
- $\delta$ – **relação de transição** – relação binária de $S \times \Gamma$ em $S \times \Gamma \times \{e, d\}$, onde $e$ indica **movimento para esquerda** (de uma posição) e $d$ **para direita**. Podemos admitir que permaneça na mesma célula e usar $\{e, d, -\}$. Ou, se se preferir, as iniciais em Inglês, de "Left" e "Right", e $\{L, R, -\}$.

A **máquina aceita a palavra** $x$ de $\Sigma^\star$ sse partindo do estado inicial $s_0$, com $x$ na fita e a cabeça de leitura no símbolo de $x$ mais à esquerda, puder realizar uma sequência finita de transições (zero ou mais) e encravar (parar) num estado de $F$.

# Exemplo de MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$

**Notação:** $(s, a, s', a', m)$ é uma transição: $s$ o estado atual, $a$ o símbolo na célula, $s'$ o estado a que passa, $a'$ o símbolo que fica na célula e $m \in \{e, d\}$ o movimento.

**Exemplo: MT que reconhece $\{0^n1^n2^n \mid n \in \mathbb{N}\}$**

Estado inicial: `inicio`; $F = \{$`aceita`$\}$; símbolo branco: •.

| | |
|---|---|
| (*inicio*, •, *aceita*, •, *e*) | *Se tem • no início, a sequência é* $\varepsilon$ |
| (*inicio*, `0`, *proc1*, `Z`, *d*) | *Marca o* `0` *com* `Z` *e vai procurar um* `1`. |
| (*proc1*, `0`, *proc1*, `0`, *d*) | *Ignora os* `0`*'s quando procura* `1`*'s.* |
| (*proc1*, `1`, *proc2*, `U`, *d*) | *Quando encontra* `1`*, marca-o e vai procurar* `2`. |
| (*proc2*, `1`, *proc2*, `1`, *d*) | *Ignora os* `1`*'s quando procura* `2`*'s.* |
| (*proc2*, `2`, *procZ*, `B`, *e*) | *Quando encontra* `2`*, marca-o e vai procurar* `0`*'s à esquerda. O último* `0` *que viu ficou marcado com* `Z`*. Vai procurar* `Z`. |
| (*procZ*, `B`, *procZ*, `B`, *e*) | *Passa os* `B`*'s quando procura o* `Z`. |
| (*procZ*, `U`, *procZ*, `U`, *e*) | *Passa os* `U`*'s quando procura o* `Z`. |
| (*procZ*, `1`, *procZ*, `1`, *e*) | *Passa os* `1`*'s quando procura o* `Z`. |
| (*procZ*, `0`, *procZ*, `0`, *e*) | *Passa os* `0`*'s quando procura o* `Z`. |
| (*procZ*, `Z`, *proc0*, `Z`, *d*) | *Encontra* `Z`*. Vai ver se ainda há* `0`*'s.* |

# Exemplo de MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$

**Notação:** $(s, a, s', a', m)$ é uma transição: $s$ o estado atual, $a$ o símbolo na célula, $s'$ o estado a que passa, $a'$ o símbolo que fica na célula e $m \in \{e, d\}$ o movimento.

**Exemplo: MT que reconhece $\{0^n1^n2^n \mid n \in \mathbb{N}\}$**

Estado inicial: `inicio`; $F = \{$`aceita`$\}$; símbolo branco: •.

| | |
|---|---|
| (*inicio*, •, *aceita*, •, *e*) | *Se tem • no início, a sequência é* $\varepsilon$ |
| (*inicio*, `0`, *proc*1, `Z`, *d*) | *Marca o* `0` *com* `Z` *e vai procurar um* `1`. |
| (*proc*1, `0`, *proc*1, `0`, *d*) | *Ignora os* `0`*'s quando procura* `1`*'s.* |
| (*proc*1, `1`, *proc*2, `U`, *d*) | *Quando encontra* `1`*, marca-o e vai procurar* `2`. |
| (*proc*2, `1`, *proc*2, `1`, *d*) | *Ignora os* `1`*'s quando procura* `2`*'s.* |
| (*proc*2, `2`, *procZ*, `B`, *e*) | *Quando encontra* `2`*, marca-o e vai procurar* `0`*'s à esquerda. O último* `0` *que viu ficou marcado com* `Z`*. Vai procurar* `Z`. |
| (*procZ*, `B`, *procZ*, `B`, *e*) | *Passa os* `B`*'s quando procura o* `Z`. |
| (*procZ*, `U`, *procZ*, `U`, *e*) | *Passa os* `U`*'s quando procura o* `Z`. |
| (*procZ*, `1`, *procZ*, `1`, *e*) | *Passa os* `1`*'s quando procura o* `Z`. |
| (*procZ*, `0`, *procZ*, `0`, *e*) | *Passa os* `0`*'s quando procura o* `Z`. |
| (*procZ*, `Z`, *proc*0, `Z`, *d*) | *Encontra* `Z`*. Vai ver se ainda há* `0`*'s.* |

# Exemplo de MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$

**Notação:** $(s, a, s', a', m)$ é uma transição: $s$ o estado atual, $a$ o símbolo na célula, $s'$ o estado a que passa, $a'$ o símbolo que fica na célula e $m \in \{e, d\}$ o movimento.

**Exemplo: MT que reconhece $\{0^n1^n2^n \mid n \in \mathbb{N}\}$**

Estado inicial: `inicio`; $F = \{$`aceita`$\}$; símbolo branco: •.

| | |
|---|---|
| (*inicio*, •, *aceita*, •, *e*) | *Se tem • no início, a sequência é* $\varepsilon$ |
| (*inicio*, 0, *proc*1, Z, *d*) | *Marca o* 0 *com* Z *e vai procurar um* 1. |
| (*proc*1, 0, *proc*1, 0, *d*) | *Ignora os* 0*'s quando procura* 1*'s.* |
| (*proc*1, 1, *proc*2, U, *d*) | *Quando encontra* 1*, marca-o e vai procurar* 2. |
| (*proc*2, 1, *proc*2, 1, *d*) | *Ignora os* 1*'s quando procura* 2*'s.* |
| (*proc*2, 2, *procZ*, B, *e*) | *Quando encontra* 2*, marca-o e vai procurar* 0*'s à esquerda. O último* 0 *que viu ficou marcado com* Z*. Vai procurar* Z. |
| (*procZ*, B, *procZ*, B, *e*) | *Passa os* B*'s quando procura o* Z. |
| (*procZ*, U, *procZ*, U, *e*) | *Passa os* U*'s quando procura o* Z. |
| (*procZ*, 1, *procZ*, 1, *e*) | *Passa os* 1*'s quando procura o* Z. |
| (*procZ*, 0, *procZ*, 0, *e*) | *Passa os* 0*'s quando procura o* Z. |
| (*procZ*, Z, *proc*0, Z, *d*) | *Encontra* Z*. Vai ver se ainda há* 0*'s.* |

# Exemplo de MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$

**Notação:** $(s, a, s', a', m)$ é uma transição: $s$ o estado atual, $a$ o símbolo na célula, $s'$ o estado a que passa, $a'$ o símbolo que fica na célula e $m \in \{e, d\}$ o movimento.

**Exemplo: MT que reconhece $\{0^n1^n2^n \mid n \in \mathbb{N}\}$**

Estado inicial: `inicio`; $F = \{$`aceita`$\}$; símbolo branco: •.

| | |
|---|---|
| (*inicio*, •, *aceita*, •, *e*) | *Se tem • no início, a sequência é* $\varepsilon$ |
| (*inicio*, 0, *proc*1, Z, *d*) | *Marca o* 0 *com* Z *e vai procurar um* 1. |
| (*proc*1, 0, *proc*1, 0, *d*) | *Ignora os* 0*'s quando procura* 1*'s.* |
| (*proc*1, 1, *proc*2, U, *d*) | *Quando encontra* 1*, marca-o e vai procurar* 2. |
| (*proc*2, 1, *proc*2, 1, *d*) | *Ignora os* 1*'s quando procura* 2*'s.* |
| (*proc*2, 2, *procZ*, B, *e*) | *Quando encontra* 2*, marca-o e vai procurar* 0*'s à esquerda. O último* 0 *que viu ficou marcado com* Z*. Vai procurar* Z. |
| (*procZ*, B, *procZ*, B, *e*) | *Passa os* B*'s quando procura o* Z. |
| (*procZ*, U, *procZ*, U, *e*) | *Passa os* U*'s quando procura o* Z. |
| (*procZ*, 1, *procZ*, 1, *e*) | *Passa os* 1*'s quando procura o* Z. |
| (*procZ*, 0, *procZ*, 0, *e*) | *Passa os* 0*'s quando procura o* Z. |
| (*procZ*, Z, *proc*0, Z, *d*) | *Encontra* Z*. Vai ver se ainda há* 0*'s.* |

# Exemplo de MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$

**Notação:** $(s, a, s', a', m)$ é uma transição: $s$ o estado atual, $a$ o símbolo na célula, $s'$ o estado a que passa, $a'$ o símbolo que fica na célula e $m \in \{e, d\}$ o movimento.

**Exemplo: MT que reconhece $\{0^n1^n2^n \mid n \in \mathbb{N}\}$**

Estado inicial: `inicio`; $F = \{$`aceita`$\}$; símbolo branco: •.

| | |
|---|---|
| (*inicio*, •, *aceita*, •, *e*) | *Se tem* • *no início, a sequência é* $\varepsilon$ |
| (*inicio*, 0, *proc*1, Z, *d*) | *Marca o* 0 *com* Z *e vai procurar um* 1. |
| (*proc*1, 0, *proc*1, 0, *d*) | *Ignora os* 0*'s quando procura* 1*'s.* |
| (*proc*1, 1, *proc*2, U, *d*) | *Quando encontra* 1*, marca-o e vai procurar* 2. |
| (*proc*2, 1, *proc*2, 1, *d*) | *Ignora os* 1*'s quando procura* 2*'s.* |
| (*proc*2, 2, *procZ*, B, *e*) | *Quando encontra* 2*, marca-o e vai procurar* 0*'s à esquerda. O último* 0 *que viu ficou marcado com* Z*. Vai procurar* Z. |
| (*procZ*, B, *procZ*, B, *e*) | *Passa os* B*'s quando procura o* Z. |
| (*procZ*, U, *procZ*, U, *e*) | *Passa os* U*'s quando procura o* Z. |
| (*procZ*, 1, *procZ*, 1, *e*) | *Passa os* 1*'s quando procura o* Z. |
| (*procZ*, 0, *procZ*, 0, *e*) | *Passa os* 0*'s quando procura o* Z. |
| (*procZ*, Z, *proc*0, Z, *d*) | *Encontra* Z*. Vai ver se ainda há* 0*'s.* |

# Exemplo de MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$

**Notação:** $(s, a, s', a', m)$ é uma transição: $s$ o estado atual, $a$ o símbolo na célula, $s'$ o estado a que passa, $a'$ o símbolo que fica na célula e $m \in \{e, d\}$ o movimento.

**Exemplo: MT que reconhece $\{0^n1^n2^n \mid n \in \mathbb{N}\}$**

Estado inicial: `inicio`; $F = \{$`aceita`$\}$; símbolo branco: •.

| | |
|---|---|
| (*inicio*, •, *aceita*, •, *e*) | *Se tem • no início, a sequência é* $\varepsilon$ |
| (*inicio*, 0, *proc*1, Z, *d*) | *Marca o* 0 *com* Z *e vai procurar um* 1. |
| (*proc*1, 0, *proc*1, 0, *d*) | *Ignora os* 0*'s quando procura* 1*'s.* |
| (*proc*1, 1, *proc*2, U, *d*) | *Quando encontra* 1*, marca-o e vai procurar* 2. |
| (*proc*2, 1, *proc*2, 1, *d*) | *Ignora os* 1*'s quando procura* 2*'s.* |
| (*proc*2, 2, *procZ*, B, *e*) | *Quando encontra* 2*, marca-o e vai procurar* 0*'s à esquerda.* |
| | *O último* 0 *que viu ficou marcado com* Z*. Vai procurar* Z. |
| (*procZ*, B, *procZ*, B, *e*) | *Passa os* B*'s quando procura o* Z. |
| (*procZ*, U, *procZ*, U, *e*) | *Passa os* U*'s quando procura o* Z. |
| (*procZ*, 1, *procZ*, 1, *e*) | *Passa os* 1*'s quando procura o* Z. |
| (*procZ*, 0, *procZ*, 0, *e*) | *Passa os* 0*'s quando procura o* Z. |
| (*procZ*, Z, *proc*0, Z, *d*) | *Encontra* Z*. Vai ver se ainda há* 0*'s.* |

# Exemplo de MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$

**Notação:** $(s, a, s', a', m)$ é uma transição: $s$ o estado atual, $a$ o símbolo na célula, $s'$ o estado a que passa, $a'$ o símbolo que fica na célula e $m \in \{e, d\}$ o movimento.

**Exemplo: MT que reconhece $\{0^n1^n2^n \mid n \in \mathbb{N}\}$**

Estado inicial: `inicio`; $F = \{$`aceita`$\}$; símbolo branco: •.

| | |
|---|---|
| (*inicio*, •, *aceita*, •, *e*) | *Se tem* • *no início, a sequência é* $\varepsilon$ |
| (*inicio*, 0, *proc*1, Z, *d*) | *Marca o* 0 *com* Z *e vai procurar um* 1. |
| (*proc*1, 0, *proc*1, 0, *d*) | *Ignora os* 0*'s quando procura* 1*'s.* |
| (*proc*1, 1, *proc*2, U, *d*) | *Quando encontra* 1*, marca-o e vai procurar* 2. |
| (*proc*2, 1, *proc*2, 1, *d*) | *Ignora os* 1*'s quando procura* 2*'s.* |
| (*proc*2, 2, *procZ*, B, *e*) | *Quando encontra* 2*, marca-o e vai procurar* 0*'s à esquerda. O último* 0 *que viu ficou marcado com* Z*. Vai procurar* Z. |
| (*procZ*, B, *procZ*, B, *e*) | *Passa os* B*'s quando procura o* Z. |
| (*procZ*, U, *procZ*, U, *e*) | *Passa os* U*'s quando procura o* Z. |
| (*procZ*, 1, *procZ*, 1, *e*) | *Passa os* 1*'s quando procura o* Z. |
| (*procZ*, 0, *procZ*, 0, *e*) | *Passa os* 0*'s quando procura o* Z. |
| (*procZ*, Z, *proc*0, Z, *d*) | *Encontra* Z*. Vai ver se ainda há* 0*'s.* |

# Exemplo de MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$

**Notação:** $(s, a, s', a', m)$ é uma transição: $s$ o estado atual, $a$ o símbolo na célula, $s'$ o estado a que passa, $a'$ o símbolo que fica na célula e $m \in \{e, d\}$ o movimento.

**Exemplo: MT que reconhece $\{0^n1^n2^n \mid n \in \mathbb{N}\}$**

Estado inicial: `inicio`; $F = \{$`aceita`$\}$; símbolo branco: •.

| | |
|---|---|
| (*inicio*, •, *aceita*, •, *e*) | *Se tem* • *no início, a sequência é* $\varepsilon$ |
| (*inicio*, 0, *proc*1, Z, *d*) | *Marca o* 0 *com* Z *e vai procurar um* 1. |
| (*proc*1, 0, *proc*1, 0, *d*) | *Ignora os* 0*'s quando procura* 1*'s.* |
| (*proc*1, 1, *proc*2, U, *d*) | *Quando encontra* 1*, marca-o e vai procurar* 2. |
| (*proc*2, 1, *proc*2, 1, *d*) | *Ignora os* 1*'s quando procura* 2*'s.* |
| (*proc*2, 2, *procZ*, B, *e*) | *Quando encontra* 2*, marca-o e vai procurar* 0*'s à esquerda. O último* 0 *que viu ficou marcado com* Z*. Vai procurar* Z. |
| (*procZ*, B, *procZ*, B, *e*) | *Passa os* B*'s quando procura o* Z. |
| (*procZ*, U, *procZ*, U, *e*) | *Passa os* U*'s quando procura o* Z. |
| (*procZ*, 1, *procZ*, 1, *e*) | *Passa os* 1*'s quando procura o* Z. |
| (*procZ*, 0, *procZ*, 0, *e*) | *Passa os* 0*'s quando procura o* Z. |
| (*procZ*, Z, *proc*0, Z, *d*) | *Encontra* Z*. Vai ver se ainda há* 0*'s.* |

# Exemplo de MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$

**Notação:** $(s, a, s', a', m)$ é uma transição: $s$ o estado atual, $a$ o símbolo na célula, $s'$ o estado a que passa, $a'$ o símbolo que fica na célula e $m \in \{e, d\}$ o movimento.

**Exemplo: MT que reconhece $\{0^n1^n2^n \mid n \in \mathbb{N}\}$**

Estado inicial: `inicio`; $F = \{$`aceita`$\}$; símbolo branco: •.

| | |
|---|---|
| (*inicio*, •, *aceita*, •, *e*) | *Se tem* • *no início, a sequência é* $\varepsilon$ |
| (*inicio*, 0, *proc*1, Z, *d*) | *Marca o* 0 *com* Z *e vai procurar um* 1. |
| (*proc*1, 0, *proc*1, 0, *d*) | *Ignora os* 0*'s quando procura* 1*'s.* |
| (*proc*1, 1, *proc*2, U, *d*) | *Quando encontra* 1*, marca-o e vai procurar* 2. |
| (*proc*2, 1, *proc*2, 1, *d*) | *Ignora os* 1*'s quando procura* 2*'s.* |
| (*proc*2, 2, *procZ*, B, *e*) | *Quando encontra* 2*, marca-o e vai procurar* 0*'s à esquerda. O último* 0 *que viu ficou marcado com* Z*. Vai procurar* Z. |
| (*procZ*, B, *procZ*, B, *e*) | *Passa os* B*'s quando procura o* Z. |
| (*procZ*, U, *procZ*, U, *e*) | *Passa os* U*'s quando procura o* Z. |
| (*procZ*, 1, *procZ*, 1, *e*) | *Passa os* 1*'s quando procura o* Z. |
| (*procZ*, 0, *procZ*, 0, *e*) | *Passa os* 0*'s quando procura o* Z. |
| (*procZ*, Z, *proc*0, Z, *d*) | *Encontra* Z*. Vai ver se ainda há* 0*'s.* |

# Exemplo de MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$

**Notação:** $(s, a, s', a', m)$ é uma transição: $s$ o estado atual, $a$ o símbolo na célula, $s'$ o estado a que passa, $a'$ o símbolo que fica na célula e $m \in \{e, d\}$ o movimento.

**Exemplo: MT que reconhece $\{0^n1^n2^n \mid n \in \mathbb{N}\}$**

Estado inicial: `inicio`; $F = \{$`aceita`$\}$; símbolo branco: •.

| | |
|---|---|
| (*inicio*, •, *aceita*, •, *e*) | *Se tem* • *no início, a sequência é* $\varepsilon$ |
| (*inicio*, 0, *proc*1, Z, *d*) | *Marca o* 0 *com* Z *e vai procurar um* 1. |
| (*proc*1, 0, *proc*1, 0, *d*) | *Ignora os* 0*'s quando procura* 1*'s.* |
| (*proc*1, 1, *proc*2, U, *d*) | *Quando encontra* 1*, marca-o e vai procurar* 2. |
| (*proc*2, 1, *proc*2, 1, *d*) | *Ignora os* 1*'s quando procura* 2*'s.* |
| (*proc*2, 2, *procZ*, B, *e*) | *Quando encontra* 2*, marca-o e vai procurar* 0*'s à esquerda. O último* 0 *que viu ficou marcado com* Z*. Vai procurar* Z. |
| (*procZ*, B, *procZ*, B, *e*) | *Passa os* B*'s quando procura o* Z. |
| (*procZ*, U, *procZ*, U, *e*) | *Passa os* U*'s quando procura o* Z. |
| (*procZ*, 1, *procZ*, 1, *e*) | *Passa os* 1*'s quando procura o* Z. |
| (*procZ*, 0, *procZ*, 0, *e*) | *Passa os* 0*'s quando procura o* Z. |
| (*procZ*, Z, *proc*0, Z, *d*) | *Encontra* Z*. Vai ver se ainda há* 0*'s.* |

# Exemplo de MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$

**Notação:** $(s, a, s', a', m)$ é uma transição: $s$ o estado atual, $a$ o símbolo na célula, $s'$ o estado a que passa, $a'$ o símbolo que fica na célula e $m \in \{e, d\}$ o movimento.

**Exemplo: MT que reconhece $\{0^n1^n2^n \mid n \in \mathbb{N}\}$**

Estado inicial: `inicio`; $F = \{$`aceita`$\}$; símbolo branco: •.

| | |
|---|---|
| (*inicio*, •, *aceita*, •, *e*) | *Se tem* • *no início, a sequência é* $\varepsilon$ |
| (*inicio*, 0, *proc*1, Z, *d*) | *Marca o* 0 *com* Z *e vai procurar um* 1. |
| (*proc*1, 0, *proc*1, 0, *d*) | *Ignora os* 0*'s quando procura* 1*'s.* |
| (*proc*1, 1, *proc*2, U, *d*) | *Quando encontra* 1*, marca-o e vai procurar* 2. |
| (*proc*2, 1, *proc*2, 1, *d*) | *Ignora os* 1*'s quando procura* 2*'s.* |
| (*proc*2, 2, *procZ*, B, *e*) | *Quando encontra* 2*, marca-o e vai procurar* 0*'s à esquerda. O último* 0 *que viu ficou marcado com* Z*. Vai procurar* Z. |
| (*procZ*, B, *procZ*, B, *e*) | *Passa os* B*'s quando procura o* Z. |
| (*procZ*, U, *procZ*, U, *e*) | *Passa os* U*'s quando procura o* Z. |
| (*procZ*, 1, *procZ*, 1, *e*) | *Passa os* 1*'s quando procura o* Z. |
| (*procZ*, 0, *procZ*, 0, *e*) | *Passa os* 0*'s quando procura o* Z. |
| (*procZ*, Z, *proc*0, Z, *d*) | *Encontra* Z*. Vai ver se ainda há* 0*'s.* |

# Exemplo de MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$

**Notação:** $(s, a, s', a', m)$ é uma transição: $s$ o estado atual, $a$ o símbolo na célula, $s'$ o estado a que passa, $a'$ o símbolo que fica na célula e $m \in \{e, d\}$ o movimento.

**Exemplo: MT que reconhece $\{0^n1^n2^n \mid n \in \mathbb{N}\}$**

Estado inicial: `inicio`; $F = \{$`aceita`$\}$; símbolo branco: •.

| | |
|---|---|
| (*inicio*, •, *aceita*, •, *e*) | *Se tem* • *no início, a sequência é* $\varepsilon$ |
| (*inicio*, 0, *proc*1, Z, *d*) | *Marca o* 0 *com* Z *e vai procurar um* 1. |
| (*proc*1, 0, *proc*1, 0, *d*) | *Ignora os* 0*'s quando procura* 1*'s.* |
| (*proc*1, 1, *proc*2, U, *d*) | *Quando encontra* 1*, marca-o e vai procurar* 2. |
| (*proc*2, 1, *proc*2, 1, *d*) | *Ignora os* 1*'s quando procura* 2*'s.* |
| (*proc*2, 2, *procZ*, B, *e*) | *Quando encontra* 2*, marca-o e vai procurar* 0*'s à esquerda. O último* 0 *que viu ficou marcado com* Z*. Vai procurar* Z. |
| (*procZ*, B, *procZ*, B, *e*) | *Passa os* B*'s quando procura o* Z. |
| (*procZ*, U, *procZ*, U, *e*) | *Passa os* U*'s quando procura o* Z. |
| (*procZ*, 1, *procZ*, 1, *e*) | *Passa os* 1*'s quando procura o* Z. |
| (*procZ*, 0, *procZ*, 0, *e*) | *Passa os* 0*'s quando procura o* Z. |
| (*procZ*, Z, *proc*0, Z, *d*) | *Encontra* Z*. Vai ver se ainda há* 0*'s.* |

# MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$ (cont.)

| | |
|---|---|
| (*proc*0, 0, *proc*1, Z, *d*) | *Se encontra* 0, *repete o processo a partir de* proc1 |
| (*proc*0, U, *verif*, U, *d*) | *Não há mais* 0*'s! Vai ver se todos os símbolos estão cortados.* |
| (*verif*, U, *verif*, U, *d*) | *Só pode haver* U*'s ou* B*'s até* • |
| (*verif*, B, *verif*, B, *d*) | |
| (*verif*, •, *repor*, •, *e*) | A palavra é de linguagem; vai para a esquerda repô-la. |
| (*repor*, B, *repor*, 2, *e*) | |
| (*repor*, U, *repor*, 1, *e*) | |
| (*repor*, Z, *repor*, 0, *e*) | |
| (*repor*, •, *aceita*, •, *d*) | Pára em aceita na posição inicial. |
| (*proc*1, U, *proc*1, U, *d*) | À procura de 1's, tem que poder ignorar os já cortados. |
| (*proc*2, B, *proc*2, B, *d*) | À procura de 2's, tem que poder ignorar os já cortados. |

Se $x \in \{0^n1^n2^n \mid n \geq 1\}$, pára em aceita, com a cabeça de leitura/escrita na posição inicial e $x$ "intacta". Mas, se $x \notin \{0^n1^n2^n \mid n \geq 1\} \cup \{\varepsilon\}$, pára num estado não final, $x$ pode estar destruída e a cabeça de leitura/escrita pode não estar na célula inicial.

# MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$ (cont.)

| | |
|---|---|
| (*proc*0, 0, *proc*1, Z, *d*) | *Se encontra* 0*, repete o processo a partir de* proc1 |
| (*proc*0, U, *verif*, U, *d*) | *Não há mais* 0*'s! Vai ver se todos os símbolos estão cortados.* |
| (*verif*, U, *verif*, U, *d*) | *Só pode haver* U*'s ou* B*'s até* • |
| (*verif*, B, *verif*, B, *d*) | |
| (*verif*, •, *repor*, •, *e*) | A palavra é de linguagem; vai para a esquerda repô-la. |
| (*repor*, B, *repor*, 2, *e*) | |
| (*repor*, U, *repor*, 1, *e*) | |
| (*repor*, Z, *repor*, 0, *e*) | |
| (*repor*, •, *aceita*, •, *d*) | Pára em aceita na posição inicial. |
| (*proc*1, U, *proc*1, U, *d*) | À procura de 1's, tem que poder ignorar os já cortados. |
| (*proc*2, B, *proc*2, B, *d*) | À procura de 2's, tem que poder ignorar os já cortados. |

Se $x \in \{0^n1^n2^n \mid n \geq 1\}$, pára em aceita, com a cabeça de leitura/escrita na posição inicial e $x$ "intacta". Mas, se $x \notin \{0^n1^n2^n \mid n \geq 1\} \cup \{\varepsilon\}$, pára num estado não final, $x$ pode estar destruída e a cabeça de leitura/escrita pode não estar na célula inicial.

# MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$ (cont.)

| | |
|---|---|
| (*proc*0, 0, *proc*1, Z, *d*) | *Se encontra* 0, *repete o processo a partir de* proc1 |
| (*proc*0, U, *verif*, U, *d*) | *Não há mais* 0*'s! Vai ver se todos os símbolos estão cortados.* |
| (*verif*, U, *verif*, U, *d*) | *Só pode haver* U*'s ou* B*'s até* • |
| (*verif*, B, *verif*, B, *d*) | |
| (*verif*, •, *repor*, •, *e*) | A palavra é de linguagem; vai para a esquerda repô-la. |
| (*repor*, B, *repor*, 2, *e*) | |
| (*repor*, U, *repor*, 1, *e*) | |
| (*repor*, Z, *repor*, 0, *e*) | |
| (*repor*, •, *aceita*, •, *d*) | Pára em aceita na posição inicial. |
| (*proc*1, U, *proc*1, U, *d*) | À procura de 1's, tem que poder ignorar os já cortados. |
| (*proc*2, B, *proc*2, B, *d*) | À procura de 2's, tem que poder ignorar os já cortados. |

Se $x \in \{0^n1^n2^n \mid n \geq 1\}$, pára em aceita, com a cabeça de leitura/escrita na posição inicial e $x$ "intacta". Mas, se $x \notin \{0^n1^n2^n \mid n \geq 1\} \cup \{\varepsilon\}$, pára num estado não final, $x$ pode estar destruída e a cabeça de leitura/escrita pode não estar na célula inicial.

# MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$ (cont.)

| | |
|---|---|
| $(proc0,\ \texttt{0},\ proc1,\ \texttt{Z},\ d)$ | *Se encontra* `0`*, repete o processo a partir de* `proc1` |
| $(proc0,\ \texttt{U},\ verif,\ \texttt{U},\ d)$ | *Não há mais* `0`*'s! Vai ver se todos os símbolos estão cortados.* |
| $(verif,\ \texttt{U},\ verif,\ \texttt{U},\ d)$ | *Só pode haver* `U`*'s ou* `B`*'s até* $\bullet$ |
| $(verif,\ \texttt{B},\ verif,\ \texttt{B},\ d)$ | |
| $(verif,\ \bullet,\ repor,\ \bullet,\ e)$ | A palavra é de linguagem; vai para a esquerda repô-la. |
| $(repor,\ \texttt{B},\ repor,\ 2,\ e)$ | |
| $(repor,\ \texttt{U},\ repor,\ 1,\ e)$ | |
| $(repor,\ \texttt{Z},\ repor,\ \texttt{0},\ e)$ | |
| $(repor,\ \bullet,\ aceita,\ \bullet,\ d)$ | Pára em `aceita` na posição inicial. |
| $(proc1,\ \texttt{U},\ proc1,\ \texttt{U},\ d)$ | À procura de 1's, tem que poder ignorar os já cortados. |
| $(proc2,\ \texttt{B},\ proc2,\ \texttt{B},\ d)$ | À procura de 2's, tem que poder ignorar os já cortados. |

Se $x \in \{0^n1^n2^n \mid n \geq 1\}$, pára em `aceita`, com a cabeça de leitura/escrita na posição inicial e $x$ "intacta". Mas, se $x \notin \{0^n1^n2^n \mid n \geq 1\} \cup \{\varepsilon\}$, pára num estado não final, $x$ pode estar destruída e a cabeça de leitura/escrita pode não estar na célula inicial.

# MT para reconhecer $\{0^n1^n2^n \mid n \in \mathbb{N}\}$ (cont.)

| | |
|---|---|
| $(proc0,\ \texttt{0},\ proc1,\ \texttt{Z},\ d)$ | *Se encontra* `0`*, repete o processo a partir de* `proc1` |
| $(proc0,\ \texttt{U},\ verif,\ \texttt{U},\ d)$ | *Não há mais* `0`*'s! Vai ver se todos os símbolos estão cortados.* |
| $(verif,\ \texttt{U},\ verif,\ \texttt{U},\ d)$ | *Só pode haver* `U`*'s ou* `B`*'s até* $\bullet$ |
| $(verif,\ \texttt{B},\ verif,\ \texttt{B},\ d)$ | |
| $(verif,\ \bullet,\ repor,\ \bullet,\ e)$ | A palavra é de linguagem; vai para a esquerda repô-la. |
| $(repor,\ \texttt{B},\ repor,\ 2,\ e)$ | |
| $(repor,\ \texttt{U},\ repor,\ 1,\ e)$ | |
| $(repor,\ \texttt{Z},\ repor,\ \texttt{0},\ e)$ | |
| $(repor,\ \bullet,\ aceita,\ \bullet,\ d)$ | Pára em `aceita` na posição inicial. |
| $(proc1,\ \texttt{U},\ proc1,\ \texttt{U},\ d)$ | À procura de 1's, tem que poder ignorar os já cortados. |
| $(proc2,\ \texttt{B},\ proc2,\ \texttt{B},\ d)$ | À procura de 2's, tem que poder ignorar os já cortados. |

Se $x \in \{0^n1^n2^n \mid n \geq 1\}$, pára em `aceita`, com a cabeça de leitura/escrita na posição inicial e $x$ "intacta". Mas, se $x \notin \{0^n1^n2^n \mid n \geq 1\} \cup \{\varepsilon\}$, pára num estado não final, $x$ pode estar destruída e a cabeça de leitura/escrita pode não estar na célula inicial.