# CC1004 - Modelos de Computação
# Teóricas 20 a 22

Ana Paula Tomás

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

Maio 2021

# Exemplo 6: Palavras com igual número de 0's e 1's

**Exemplo 6** A linguagem $L = \{x \mid x \in \{0,1\}^\star \text{ tem igual número de 0's e 1's}\}$ é reconhecida por pilha vazia por $\mathcal{A} = (\{s_0\}, \{0,1\}, \{\mathtt{Z}, \mathtt{A}, \mathtt{B}\}, \delta, s_0, \mathtt{Z}, \{\ \})$, com

$$
\begin{array}{lcl@{\qquad\qquad}lcl}
\delta(s_0, \varepsilon, \mathtt{Z}) & = & \{(s_0, \varepsilon)\} & & & \\
\delta(s_0, 0, \mathtt{Z}) & = & \{(s_0, \mathtt{BZ})\} & \delta(s_0, 1, \mathtt{Z}) & = & \{(s_0, \mathtt{AZ})\} \\
\delta(s_0, 0, \mathtt{B}) & = & \{(s_0, \mathtt{BB})\} & \delta(s_0, 1, \mathtt{A}) & = & \{(s_0, \mathtt{AA})\} \\
\delta(s_0, 0, \mathtt{A}) & = & \{(s_0, \varepsilon)\} & \delta(s_0, 1, \mathtt{B}) & = & \{(s_0, \varepsilon)\}
\end{array}
$$

Para este autómato $\mathcal{A}$, quaisquer que sejam $x, y \in \Sigma^\star$, tem-se:

- $(s_0, xy, \mathtt{Z}) \vdash^\star (s_0, y, \mathtt{B}^k\mathtt{Z})$, com $k \geq 1$, se e só se $\#_0(x) - \#_1(x) = k$.
- $(s_0, xy, \mathtt{Z}) \vdash^\star (s_0, y, \mathtt{A}^k\mathtt{Z})$, com $k \geq 1$, se e só se $\#_1(x) - \#_0(x) = k$.
- $(s_0, xy, \mathtt{Z}) \vdash^\star (s_0, y, \mathtt{Z})$ se e só se $\#_1(x) = \#_0(x)$.
- $(s_0, xy, \mathtt{Z}) \vdash^\star (s_0, y, \varepsilon)$ se e só se $\#_1(x) = \#_0(x)$.

Portanto, $(s_0, x, \mathtt{Z}) \vdash^\star (s_0, \varepsilon, \varepsilon)$ se e só se $x \in L$.

**Notação:** Para $a \in \Sigma$, denotamos **o número de $a$'s em $x$** por $\#_a(x)$.

# Palavras com menos 0's do que 1's

**Exemplo 7** **Como adaptar o autómato anterior** para definir um autómato que reconhece $L = \{x \mid x \in \{0,1\}^\star \text{ e } \#_0(x) < \#_1(x)\text{'s}\}$ por pilha vazia?

**No Exemplo 6 tinhamos.**

- $(s_0, xy, \mathtt{Z}) \vdash^\star (s_0, y, \mathtt{Z})$ se e só se $\#_1(x) = \#_0(x)$.
- $(s_0, xy, \mathtt{Z}) \vdash^\star (s_0, y, \mathtt{A}^k\mathtt{Z})$, com $k \geq 1$, se e só se $\#_1(x) - \#_0(x) = k$

Ideia: Em $s_0$, não retira Z. Mas, com A no topo, vai poder esvaziar a pilha.

Definimos $\mathcal{A}' = (\{s_0, s_1\}, \{0,1\}, \{\mathtt{Z}, \mathtt{A}, \mathtt{B}\}, \delta, s_0, \mathtt{Z}, \{\ \})$, com

$$
\begin{array}{lcl@{\qquad\qquad}lcl}
\delta(s_0, 0, \mathtt{Z}) & = & \{(s_0, \mathtt{BZ})\} & \delta(s_0, 1, \mathtt{Z}) & = & \{(s_0, \mathtt{AZ})\} \\
\delta(s_0, 0, \mathtt{B}) & = & \{(s_0, \mathtt{BB})\} & \delta(s_0, 1, \mathtt{A}) & = & \{(s_0, \mathtt{AA})\} \\
\delta(s_0, 0, \mathtt{A}) & = & \{(s_0, \varepsilon)\} & \delta(s_0, 1, \mathtt{B}) & = & \{(s_0, \varepsilon)\} \\
\delta(s_0, \varepsilon, \mathtt{A}) & = & \{(s_1, \varepsilon)\} & \delta(s_1, \varepsilon, \mathtt{A}) & = & \{(s_1, \varepsilon)\} \\
\delta(s_1, \varepsilon, \mathtt{Z}) & = & \{(s_1, \varepsilon)\} & & &
\end{array}
$$

# Palavras com menos 0's do que 1's

**Exemplo 7** **Como adaptar o autómato anterior** para definir um autómato que reconhece $L = \{x \mid x \in \{0,1\}^\star \text{ e } \#_0(x) < \#_1(x)\text{'s}\}$ por pilha vazia?

**No Exemplo 6 tinhamos.**

- $(s_0, xy, \mathtt{Z}) \vdash^\star (s_0, y, \mathtt{Z})$ se e só se $\#_1(x) = \#_0(x)$.
- $(s_0, xy, \mathtt{Z}) \vdash^\star (s_0, y, \mathtt{A}^k\mathtt{Z})$, com $k \geq 1$, se e só se $\#_1(x) - \#_0(x) = k$

Ideia: Em $s_0$, não retira Z. Mas, com A no topo, vai poder esvaziar a pilha.

Definimos $\mathcal{A}' = (\{s_0, s_1\}, \{0,1\}, \{\mathtt{Z}, \mathtt{A}, \mathtt{B}\}, \delta, s_0, \mathtt{Z}, \{\ \})$, com

$$
\begin{array}{lcl@{\qquad\qquad}lcl}
\delta(s_0, 0, \mathtt{Z}) & = & \{(s_0, \mathtt{BZ})\} & \delta(s_0, 1, \mathtt{Z}) & = & \{(s_0, \mathtt{AZ})\} \\
\delta(s_0, 0, \mathtt{B}) & = & \{(s_0, \mathtt{BB})\} & \delta(s_0, 1, \mathtt{A}) & = & \{(s_0, \mathtt{AA})\} \\
\delta(s_0, 0, \mathtt{A}) & = & \{(s_0, \varepsilon)\} & \delta(s_0, 1, \mathtt{B}) & = & \{(s_0, \varepsilon)\} \\
\delta(s_0, \varepsilon, \mathtt{A}) & = & \{(s_1, \varepsilon)\} & \delta(s_1, \varepsilon, \mathtt{A}) & = & \{(s_1, \varepsilon)\} \\
\delta(s_1, \varepsilon, \mathtt{Z}) & = & \{(s_1, \varepsilon)\} & & &
\end{array}
$$

# Palavras com menos 0's do que 1's

**Exemplo 7** **Como adaptar o autómato anterior** para definir um autómato que reconhece $L = \{x \mid x \in \{0,1\}^\star \text{ e } \#_0(x) < \#_1(x)\text{'s}\}$ por pilha vazia?

**No Exemplo 6 tinhamos.**

- $(s_0, xy, \mathtt{Z}) \vdash^\star (s_0, y, \mathtt{Z})$ se e só se $\#_1(x) = \#_0(x)$.
- $(s_0, xy, \mathtt{Z}) \vdash^\star (s_0, y, \mathtt{A}^k\mathtt{Z})$, com $k \geq 1$, se e só se $\#_1(x) - \#_0(x) = k$

Ideia: Em $s_0$, não retira $\mathtt{Z}$. Mas, com $\mathtt{A}$ no topo, vai poder esvaziar a pilha.

Definimos $\mathcal{A}' = (\{s_0, s_1\}, \{0,1\}, \{\mathtt{Z}, \mathtt{A}, \mathtt{B}\}, \delta, s_0, \mathtt{Z}, \{\ \})$, com

$$
\begin{array}{lcl}
\delta(s_0, 0, \mathtt{Z}) & = & \{(s_0, \mathtt{BZ})\} \\
\delta(s_0, 0, \mathtt{B}) & = & \{(s_0, \mathtt{BB})\} \\
\delta(s_0, 0, \mathtt{A}) & = & \{(s_0, \varepsilon)\} \\
\delta(s_0, \varepsilon, \mathtt{A}) & = & \{(s_1, \varepsilon)\} \\
\delta(s_1, \varepsilon, \mathtt{Z}) & = & \{(s_1, \varepsilon)\}
\end{array}
\qquad
\begin{array}{lcl}
\delta(s_0, 1, \mathtt{Z}) & = & \{(s_0, \mathtt{AZ})\} \\
\delta(s_0, 1, \mathtt{A}) & = & \{(s_0, \mathtt{AA})\} \\
\delta(s_0, 1, \mathtt{B}) & = & \{(s_0, \varepsilon)\} \\
\delta(s_1, \varepsilon, \mathtt{A}) & = & \{(s_1, \varepsilon)\}
\end{array}
$$

# Algumas Propriedades de Autómatos de Pilha (APs)

## Conversão: aceitação por estados finais para aceitação por pilha vazia

Seja $\mathcal{A} = (S, \Sigma, \Gamma, \delta, s_0, Z_0, F)$ um AP com aceitação por estados finais. Então, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ para $\mathcal{A}' = (S \cup \{s_e\}, \Sigma, \Gamma, \delta', s_0, Z_0, \{\})$ com aceitação por pilha vazia, onde $s_e$ é um novo estado, i.e., $s_e \notin S$, e $\delta'$ uma extensão de $\delta$, dada por

- $\delta'(s, a, X) = \delta(s, a, X)$, para $(s, a, X) \in S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$
- $\delta'(f, \varepsilon, X) = \{(s_e, \varepsilon)\}$, para $(f, X) \in F \times \Gamma$
- $\delta'(s_e, \varepsilon, X) = \{(s_e, \varepsilon)\}$, para $X \in \Gamma$
- $\delta'(s_e, a, X) = \{\}$, para $(a, X) \in \Sigma \times \Gamma$

**Ideia:** $\mathcal{A}'$ simula $\mathcal{A}$ mas, sempre que $\mathcal{A}$ puder estar num estado final $f$, $\mathcal{A}'$ pode efetuar uma transição por $\varepsilon$ para um novo estado $s_e$ em que irá retirar todos os símbolos da pilha. Assim, $(s_0, x, Z_0) \vdash^\star_{\mathcal{A}} (f, \varepsilon, \gamma)$ se e só se $(s_0, x, Z_0) \vdash^\star_{\mathcal{A}'} (f, \varepsilon, \gamma)$ e, como se $(s, w, \gamma) \vdash^\star_{\mathcal{A}'} (s_e, w, \varepsilon)$ então $s \in F \cup \{s_e\}$, tem-se

$$(s_0, x, Z_0) \vdash^\star_{\mathcal{A}} (f, \varepsilon, \gamma) \text{ se e só se } (s_0, x, Z_0) \vdash^\star_{\mathcal{A}'} (s_e, \varepsilon, \varepsilon)$$

# Algumas Propriedades de Autómatos de Pilha (APs)

## Conversão: aceitação por estados finais para aceitação por pilha vazia

Seja $\mathcal{A} = (S, \Sigma, \Gamma, \delta, s_0, Z_0, F)$ um AP com aceitação por estados finais. Então, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ para $\mathcal{A}' = (S \cup \{s_e\}, \Sigma, \Gamma, \delta', s_0, Z_0, \{\})$ com aceitação por pilha vazia, onde $s_e$ é um novo estado, i.e., $s_e \notin S$, e $\delta'$ uma extensão de $\delta$, dada por

- $\delta'(s, a, X) = \delta(s, a, X)$, para $(s, a, X) \in S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$
- $\delta'(f, \varepsilon, X) = \{(s_e, \varepsilon)\}$, para $(f, X) \in F \times \Gamma$
- $\delta'(s_e, \varepsilon, X) = \{(s_e, \varepsilon)\}$, para $X \in \Gamma$
- $\delta'(s_e, a, X) = \{\}$, para $(a, X) \in \Sigma \times \Gamma$

**Ideia:** $\mathcal{A}'$ simula $\mathcal{A}$ mas, sempre que $\mathcal{A}$ puder estar num estado final $f$, $\mathcal{A}'$ pode efetuar uma transição por $\varepsilon$ para um novo estado $s_e$ em que irá retirar todos os símbolos da pilha. Assim, $(s_0, x, Z_0) \vdash^\star_{\mathcal{A}} (f, \varepsilon, \gamma)$ se e só se $(s_0, x, Z_0) \vdash^\star_{\mathcal{A}'} (f, \varepsilon, \gamma)$ e, como se $(s, w, \gamma) \vdash^\star_{\mathcal{A}'} (s_e, w, \varepsilon)$ então $s \in F \cup \{s_e\}$, tem-se

$$(s_0, x, Z_0) \vdash^\star_{\mathcal{A}} (f, \varepsilon, \gamma) \text{ se e só se } (s_0, x, Z_0) \vdash^\star_{\mathcal{A}'} (s_e, \varepsilon, \varepsilon)$$

# Algumas Propriedades de Autómatos de Pilha (APs)

### Conversão: aceitação por estados finais para aceitação por pilha vazia

Seja $\mathcal{A} = (S, \Sigma, \Gamma, \delta, s_0, Z_0, F)$ um AP com aceitação por estados finais. Então, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ para $\mathcal{A}' = (S \cup \{s_e\}, \Sigma, \Gamma, \delta', s_0, Z_0, \{\})$ com aceitação por pilha vazia, onde $s_e$ é um novo estado, i.e., $s_e \notin S$, e $\delta'$ uma extensão de $\delta$, dada por

- $\delta'(s, a, X) = \delta(s, a, X)$, para $(s, a, X) \in S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$
- $\delta'(f, \varepsilon, X) = \{(s_e, \varepsilon)\}$, para $(f, X) \in F \times \Gamma$
- $\delta'(s_e, \varepsilon, X) = \{(s_e, \varepsilon)\}$, para $X \in \Gamma$
- $\delta'(s_e, a, X) = \{\}$, para $(a, X) \in \Sigma \times \Gamma$

**Ideia:** $\mathcal{A}'$ simula $\mathcal{A}$ mas, sempre que $\mathcal{A}$ puder estar num estado final $f$, $\mathcal{A}'$ pode efetuar uma transição por $\varepsilon$ para um novo estado $s_e$ em que irá retirar todos os símbolos da pilha. Assim, $(s_0, x, Z_0) \vdash_{\mathcal{A}}^{\star} (f, \varepsilon, \gamma)$ se e só se $(s_0, x, Z_0) \vdash_{\mathcal{A}'}^{\star} (f, \varepsilon, \gamma)$ e, como se $(s, w, \gamma) \vdash_{\mathcal{A}'}^{\star} (s_e, w, \varepsilon)$ então $s \in F \cup \{s_e\}$, tem-se

$$(s_0, x, Z_0) \vdash_{\mathcal{A}}^{\star} (f, \varepsilon, \gamma) \text{ se e só se } (s_0, x, Z_0) \vdash_{\mathcal{A}'}^{\star} (s_e, \varepsilon, \varepsilon)$$

# Algumas Propriedades de Autómatos de Pilha (APs)

## Conversão: aceitação por pilha vazia para aceitação por estados finais

Seja $\mathcal{A} = (S, \Sigma, \Gamma, \delta, s_0, Z_0, \{\})$ um AP com aceitação por pilha vazia. Então $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ para $\mathcal{A}' = (S \cup \{f, s_0'\}, \Sigma, \Gamma \cup \{Z_0'\}, \delta', s_0', Z_0', \{f\})$ com aceitação por estados finais, onde $f$ e $s_0'$ são novos estados, $Z_0'$ é um novo símbolo, e $\delta'$ uma extensão de $\delta$, dada por

- $\delta'(s_0', \varepsilon, Z_0') = \{(s_0, Z_0 Z_0')\}$
- $\delta'(s, a, X) = \delta(s, a, X)$, para $(s, a, X) \in S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$
- $\delta'(s, \varepsilon, Z_0') = \{(f, Z_0')\}$, para $s \in S$

e para todos os ternos não indicados $\delta'(s, a, X) = \emptyset$.

**Ideia:** $\mathcal{A}'$ começa num estado inicial novo $s_0'$ e com um símbolo inicial $Z_0'$. Sem consumir símbolos da palavra, coloca $Z_0$ na pilha (sem retirar $Z_0'$) e passa a $s_0$. A partir daí simula $\mathcal{A}$. Se voltar a ter $Z_0'$ no topo da pilha, o que significa que $\mathcal{A}$ estaria em estado de aceitação (pilha vazia), então pode passar ao estado final $f$. Tem-se

$$(s_0', x, Z_0') \vdash^\star_{\mathcal{A}'} (f, \varepsilon, \gamma) \text{ se e só se } \gamma = Z_0' \text{ e } (s_0, x, Z_0) \vdash^\star_{\mathcal{A}} (s, \varepsilon, \varepsilon), \text{ para algum } s \in S$$

# Algumas Propriedades de Autómatos de Pilha (APs)

### Conversão: aceitação por pilha vazia para aceitação por estados finais

Seja $\mathcal{A} = (S, \Sigma, \Gamma, \delta, s_0, Z_0, \{\})$ um AP com aceitação por pilha vazia. Então $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ para $\mathcal{A}' = (S \cup \{f, s_0'\}, \Sigma, \Gamma \cup \{Z_0'\}, \delta', s_0', Z_0', \{f\})$ com aceitação por estados finais, onde $f$ e $s_0'$ são novos estados, $Z_0'$ é um novo símbolo, e $\delta'$ uma extensão de $\delta$, dada por

- $\delta'(s_0', \varepsilon, Z_0') = \{(s_0, Z_0 Z_0')\}$
- $\delta'(s, a, X) = \delta(s, a, X)$, para $(s, a, X) \in S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$
- $\delta'(s, \varepsilon, Z_0') = \{(f, Z_0')\}$, para $s \in S$

e para todos os ternos não indicados $\delta'(s, a, X) = \emptyset$.

**Ideia:** $\mathcal{A}'$ começa num estado inicial novo $s_0'$ e com um símbolo inicial $Z_0'$. Sem consumir símbolos da palavra, coloca $Z_0$ na pilha (sem retirar $Z_0'$) e passa a $s_0$. A partir daí simula $\mathcal{A}$. Se voltar a ter $Z_0'$ no topo da pilha, o que significa que $\mathcal{A}$ estaria em estado de aceitação (pilha vazia), então pode passar ao estado final $f$. Tem-se

$$(s_0', x, Z_0') \vdash^\star_{\mathcal{A}'} (f, \varepsilon, \gamma) \text{ se e só se } \gamma = Z_0' \text{ e } (s_0, x, Z_0) \vdash^\star_{\mathcal{A}} (s, \varepsilon, \varepsilon), \text{ para algum } s \in S$$

# Algumas Propriedades de Autómatos de Pilha (APs)

### Conversão de AFND-$\varepsilon$ para autómato de pilha (AP)

Dado $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um AFND-$\varepsilon$, seja $\mathcal{A}' = (S, \Sigma, \{\mathrm{Z}\}, \delta', s_0, \mathrm{Z}, F)$ um AP com $\delta'(s, a, \mathrm{Z}) = \{(s', \mathrm{Z}) \mid s' \in \delta(s, a)\}$, para $a \in \Sigma \cup \{\varepsilon\}$, $s \in S$, com aceitação por estados finais. Então, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

**Justificação:** quaisquer que sejam $x \in \Sigma^\star$ e $f \in F$ tem-se

$$(s_0, x) \vdash_{\mathcal{A}} (f, \varepsilon) \text{ se e só se } (s_0, x, \mathrm{Z}) \vdash_{\mathcal{A}'} (f, \varepsilon, \mathrm{Z})$$

Aqui, estamos a considerar $\vdash_{\mathcal{A}}$ definida por $(s, ax) \vdash_{\mathcal{A}} (s', x)$ se $s' \in \delta(s, a)$, para $(s, a) \in S \times (\Sigma \cup \varepsilon)$ embora, por conveniência, não tenha sido esta a noção introduzida para AFNDs-$\varepsilon$.

### Corolário

Qualquer linguagem regular pode ser reconhecida por um autómato de pilha.

# Algumas Propriedades de Autómatos de Pilha (APs)

## Conversão de AFND-$\varepsilon$ para autómato de pilha (AP)

Dado $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um AFND-$\varepsilon$, seja $\mathcal{A}' = (S, \Sigma, \{\mathrm{Z}\}, \delta', s_0, \mathrm{Z}, F)$ um AP com $\delta'(s, a, \mathrm{Z}) = \{(s', \mathrm{Z}) \mid s' \in \delta(s, a)\}$, para $a \in \Sigma \cup \{\varepsilon\}$, $s \in S$, com aceitação por estados finais. Então, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

**Justificação:** quaisquer que sejam $x \in \Sigma^\star$ e $f \in F$ tem-se

$$(s_0, x) \vdash_{\mathcal{A}} (f, \varepsilon) \text{ se e só se } (s_0, x, \mathrm{Z}) \vdash_{\mathcal{A}'} (f, \varepsilon, \mathrm{Z})$$

Aqui, estamos a considerar $\vdash_{\mathcal{A}}$ definida por $(s, ax) \vdash_{\mathcal{A}} (s', x)$ se $s' \in \delta(s, a)$, para $(s, a) \in S \times (\Sigma \cup \varepsilon)$ embora, por conveniência, não tenha sido esta a noção introduzida para AFNDs-$\varepsilon$.

## Corolário

Qualquer linguagem regular pode ser reconhecida por um autómato de pilha.

# Algumas Propriedades de Autómatos de Pilha (APs)

### Conversão de AFND-$\varepsilon$ para autómato de pilha (AP)

Dado $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um AFND-$\varepsilon$, seja $\mathcal{A}' = (S, \Sigma, \{\mathrm{Z}\}, \delta', s_0, \mathrm{Z}, F)$ um AP com $\delta'(s, a, \mathrm{Z}) = \{(s', \mathrm{Z}) \mid s' \in \delta(s, a)\}$, para $a \in \Sigma \cup \{\varepsilon\}$, $s \in S$, com aceitação por estados finais. Então, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

**Justificação:** quaisquer que sejam $x \in \Sigma^\star$ e $f \in F$ tem-se

$$(s_0, x) \vdash_{\mathcal{A}} (f, \varepsilon) \text{ se e só se } (s_0, x, \mathrm{Z}) \vdash_{\mathcal{A}'} (f, \varepsilon, \mathrm{Z})$$

Aqui, estamos a considerar $\vdash_{\mathcal{A}}$ definida por $(s, ax) \vdash_{\mathcal{A}} (s', x)$ se $s' \in \delta(s, a)$, para $(s, a) \in S \times (\Sigma \cup \varepsilon)$ embora, por conveniência, não tenha sido esta a noção introduzida para AFNDs-$\varepsilon$.

### Corolário

Qualquer linguagem regular pode ser reconhecida por um autómato de pilha.

# Linguagens Aceites por APs são LICs

### APs computacionalmente mais potentes do que AFs

Existem linguagens de alfabeto $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ que não são regulares e, portanto, não podem ser reconhecidas por autómatos finitos, mas podem ser reconhecidas por autómatos de pilha.

**Exemplos:**

$$\{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\}$$

$$\{wtw^R \mid w \in \{\mathtt{a}, \mathtt{b}\}^\star, t \in \{\varepsilon, \mathtt{a}, \mathtt{b}\}\}$$

Estas linguagens são independentes de contexto (mas não regulares).

Vamos ver que **as linguagens reconhecidas por APs são LICs e as LICs são reconhecidas por APs.**

# APs reconhecem LICs

**Transformação AP para GIC**

Seja $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, s_0, Z_0, \{\})$ um AP com aceitação por pilha vazia. Então $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{G})$ para a GIC $\mathcal{G} = (V, \Sigma, P, S)$, que tem por **variáveis** os ternos $[q, Z, q']$, para $q, q' \in Q$ e $Z \in \Gamma$, além do **símbolo inicial** $S$. As **regras** são:

- $S \to [q_0, Z_0, q]$, para todo $q \in Q$;
- $[q, Z, q'] \to a$, se $(q', \varepsilon) \in \delta(q, a, Z)$, com $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$ e $q, q' \in Q$;
- $[q, Z, q_n] \to a[q', X_1, q_1] \cdots [q_{n-1}, X_n, q_n]$, se $(q', X_1 \cdots X_n) \in \delta(q, a, Z)$, com $q, q' \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, e $X_1, \ldots, X_n \in \Gamma$, para **todas as sequências** $(q_1, \ldots, q_{n-1}, q_n)$, com $q_i \in Q$, para todo $i$.

**Ideia:** $[q, Z, q']$ designa o conjunto das palavras que, partindo do estado $q$ com $Z$ no topo da pilha, podem ser consumidas e retirar $Z$ deixando $\mathcal{A}$ em $q'$, no sentido de

$$(q, xy, Z\gamma) \vdash^{\star}_{\mathcal{A}} (q', y, \gamma)$$

Podemos mostrar que $\mathcal{L}(A) = \mathcal{L}(G)$ por análise da relação entre $\vdash_{\mathcal{A}}$ e $\Rightarrow_{\mathcal{G}}$. Numa derivação pela esquerda, a sequência $x$ que $G$ gera é a sequência $x$ que $\mathcal{A}$ consome.

# APs reconhecem LICs

### Transformação AP para GIC

Seja $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, s_0, Z_0, \{\})$ um AP com aceitação por pilha vazia. Então $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{G})$ para a GIC $\mathcal{G} = (V, \Sigma, P, S)$, que tem por **variáveis** os ternos $[q, Z, q']$, para $q, q' \in Q$ e $Z \in \Gamma$, além do **símbolo inicial** $S$. As **regras** são:

- $S \to [q_0, Z_0, q]$, para todo $q \in Q$;
- $[q, Z, q'] \to a$, se $(q', \varepsilon) \in \delta(q, a, Z)$, com $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$ e $q, q' \in Q$;
- $[q, Z, q_n] \to a[q', X_1, q_1] \cdots [q_{n-1}, X_n, q_n]$, se $(q', X_1 \cdots X_n) \in \delta(q, a, Z)$, com $q, q' \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, e $X_1, \ldots, X_n \in \Gamma$, para **todas as sequências** $(q_1, \ldots, q_{n-1}, q_n)$, com $q_i \in Q$, para todo $i$.

**Ideia:** $[q, Z, q']$ designa o conjunto das palavras que, partindo do estado $q$ com $Z$ no topo da pilha, podem ser consumidas e retirar $Z$ deixando $\mathcal{A}$ em $q'$, no sentido de

$$(q, xy, Z\gamma) \vdash_{\mathcal{A}}^{\star} (q', y, \gamma)$$

Podemos mostrar que $\mathcal{L}(A) = \mathcal{L}(G)$ por análise da relação entre $\vdash_{\mathcal{A}}$ e $\Rightarrow_{\mathcal{G}}$. Numa derivação pela esquerda, a sequência $x$ que $G$ gera é a sequência $x$ que $\mathcal{A}$ consome.

# APs reconhecem LICs

### Transformação AP para GIC

Seja $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, s_0, Z_0, \{\})$ um AP com aceitação por pilha vazia. Então $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{G})$ para a GIC $\mathcal{G} = (V, \Sigma, P, S)$, que tem por **variáveis** os ternos $[q, Z, q']$, para $q, q' \in Q$ e $Z \in \Gamma$, além do **símbolo inicial** $S$. As **regras** são:

- $S \to [q_0, Z_0, q]$, para todo $q \in Q$;
- $[q, Z, q'] \to a$, se $(q', \varepsilon) \in \delta(q, a, Z)$, com $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$ e $q, q' \in Q$;
- $[q, Z, q_n] \to a[q', X_1, q_1] \cdots [q_{n-1}, X_n, q_n]$, se $(q', X_1 \cdots X_n) \in \delta(q, a, Z)$, com $q, q' \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, e $X_1, \ldots, X_n \in \Gamma$, para **todas as sequências** $(q_1, \ldots, q_{n-1}, q_n)$, com $q_i \in Q$, para todo $i$.

**Ideia:** $[q, Z, q']$ designa o conjunto das palavras que, partindo do estado $q$ com $Z$ no topo da pilha, podem ser consumidas e retirar $Z$ deixando $\mathcal{A}$ em $q'$, no sentido de

$$(q, xy, Z\gamma) \vdash^{\star}_{\mathcal{A}} (q', y, \gamma)$$

Podemos mostrar que $\mathcal{L}(A) = \mathcal{L}(G)$ por análise da relação entre $\vdash_{\mathcal{A}}$ e $\Rightarrow_{\mathcal{G}}$. Numa derivação pela esquerda, a sequência $x$ que $G$ gera é a sequência $x$ que $\mathcal{A}$ consome.

# Exemplo: AP para GIC

**Exemplo** Seja $\mathcal{A} = (\{s_0, s_1\}, \{0, 1\}, \{\mathtt{Z}, \mathtt{A}\}, \delta, s_0, \mathtt{Z}, \{\ \})$ um autómato de pilha, com aceitação por pilha vazia, onde $\delta$ é dada por:

$$\begin{array}{lclclcl}
\delta(s_0, \varepsilon, \mathtt{Z}) & = & \{(s_1, \mathtt{Z})\} & \quad & \delta(s_0, 0, \mathtt{Z}) & = & \{(s_0, \mathtt{AZ})\} \\
\delta(s_0, 0, \mathtt{A}) & = & \{(s_0, \mathtt{AA})\} & & \delta(s_0, 1, \mathtt{A}) & = & \{(s_1, \varepsilon)\} \\
\delta(s_1, 1, \mathtt{A}) & = & \{(s_1, \varepsilon)\} & & \delta(s_1, 1, \mathtt{Z}) & = & \{(s_1, \varepsilon), (s_1, \mathtt{Z})\}
\end{array}$$

As variáveis da GIC $\mathcal{G} = (V, \{0, 1\}, P, S)$ definida pelo método de conversão seriam: $S$, $[s_0, \mathtt{Z}, s_0]$, $[s_0, \mathtt{Z}, s_1]$, $[s_1, \mathtt{Z}, s_0]$, $[s_1, \mathtt{Z}, s_1]$, $[s_0, \mathtt{A}, s_0]$, $[s_0, \mathtt{A}, s_1]$, $[s_1, \mathtt{A}, s_0]$, $[s_1, \mathtt{A}, s_1]$.

Algumas destas variáveis poderão não gerar sequências de terminais por, de facto, não constituirem uma possibilidade real no autómato. Podem ser descartadas.

Na tradução das transições, podemos restringir-nos às variáveis que vão surgindo no lado direito de regras, começando pelas regras para $S$.

# Exemplo: AP para GIC

**Exemplo** Seja $\mathcal{A} = (\{s_0, s_1\}, \{0, 1\}, \{\mathtt{Z}, \mathtt{A}\}, \delta, s_0, \mathtt{Z}, \{\ \})$ um autómato de pilha, com aceitação por pilha vazia, onde $\delta$ é dada por:

$$\begin{array}{rcl@{\qquad}rcl}
\delta(s_0, \varepsilon, \mathtt{Z}) & = & \{(s_1, \mathtt{Z})\} & \delta(s_0, 0, \mathtt{Z}) & = & \{(s_0, \mathtt{AZ})\} \\
\delta(s_0, 0, \mathtt{A}) & = & \{(s_0, \mathtt{AA})\} & \delta(s_0, 1, \mathtt{A}) & = & \{(s_1, \varepsilon)\} \\
\delta(s_1, 1, \mathtt{A}) & = & \{(s_1, \varepsilon)\} & \delta(s_1, 1, \mathtt{Z}) & = & \{(s_1, \varepsilon), (s_1, \mathtt{Z})\}
\end{array}$$

Regras para $S$:

$$S \rightarrow [s_0, \mathrm{Z}, s_0] \qquad\qquad S \rightarrow [s_0, \mathrm{Z}, s_1]$$

Estes regras traduzem o facto de as palavras geradas pela gramática (consumidas pelo autómato) devem ser as que, partindo do estado $s_0$ com Z no topo da pilha permitem chegar a pilha vazia, podendo ter de acrescentar e retirar outros símbolos até o conseguir. No fim, o estado pode ser qualquer, isto é, $s_0$ ou $s_1$.

# Exemplo (cont.): AP para GIC

**Exemplo (cont)** AP $\mathcal{A} = (\{s_0, s_1\}, \{0, 1\}, \{\mathtt{Z}, \mathtt{A}\}, \delta, s_0, \mathtt{Z}, \{\ \})$ com aceitação por pilha vazia e $\delta$ dada por:

$$
\begin{array}{rclcrcl}
\delta(s_0, \varepsilon, \mathtt{Z}) & = & \{(s_1, \mathtt{Z})\} & & \delta(s_0, 0, \mathtt{Z}) & = & \{(s_0, \mathtt{AZ})\} \\
\delta(s_0, 0, \mathtt{A}) & = & \{(s_0, \mathtt{AA})\} & & \delta(s_0, 1, \mathtt{A}) & = & \{(s_1, \varepsilon)\} \\
\delta(s_1, 1, \mathtt{A}) & = & \{(s_1, \varepsilon)\} & & \delta(s_1, 1, \mathtt{Z}) & = & \{(s_1, \varepsilon), (s_1, \mathtt{Z})\}
\end{array}
$$

Tradução das transições em regras da gramática:

- $\delta(s_0, \varepsilon, \mathtt{Z}) = \{(s_1, \mathtt{Z})\}$ é traduzida por duas regras de produção:

$$
\begin{array}{rcl}
[s_0, \mathtt{Z}, s_0] & \rightarrow & [s_1, \mathtt{Z}, s_0] \\
[s_0, \mathtt{Z}, s_1] & \rightarrow & [s_1, \mathtt{Z}, s_1]
\end{array}
$$

- $\delta(s_0, 0, \mathtt{Z}) = \{(s_0, \mathtt{AZ})\}$ dá origem às quatro regras seguintes, que resultam de o estado final poder ser $s_1$ ou $s_0$ e o intermédio também:

$$
\begin{array}{rclcrcl}
[s_0, \mathtt{Z}, s_1] & \rightarrow & 0[s_0, \mathtt{A}, s_0][s_0, \mathtt{Z}, s_1] & & [s_0, \mathtt{Z}, s_1] & \rightarrow & 0[s_0, \mathtt{A}, s_1][s_1, \mathtt{Z}, s_1] \\
[s_0, \mathtt{Z}, s_0] & \rightarrow & 0[s_0, \mathtt{A}, s_0][s_0, \mathtt{Z}, s_0] & & [s_0, \mathtt{Z}, s_0] & \rightarrow & 0[s_0, \mathtt{A}, s_1][s_1, \mathtt{Z}, s_0]
\end{array}
$$

# Exemplo (cont.): AP para GIC

**Exemplo (cont)** AP $\mathcal{A} = (\{s_0, s_1\}, \{0, 1\}, \{\mathtt{Z}, \mathtt{A}\}, \delta, s_0, \mathtt{Z}, \{\ \})$ com aceitação por pilha vazia e $\delta$ dada por:

$$
\begin{array}{lclclcl}
\delta(s_0, \varepsilon, \mathtt{Z}) & = & \{(s_1, \mathtt{Z})\} & \quad & \delta(s_0, 0, \mathtt{Z}) & = & \{(s_0, \mathtt{AZ})\} \\
\delta(s_0, 0, \mathtt{A}) & = & \{(s_0, \mathtt{AA})\} & & \delta(s_0, 1, \mathtt{A}) & = & \{(s_1, \varepsilon)\} \\
\delta(s_1, 1, \mathtt{A}) & = & \{(s_1, \varepsilon)\} & & \delta(s_1, 1, \mathtt{Z}) & = & \{(s_1, \varepsilon), (s_1, \mathtt{Z})\}
\end{array}
$$

Tradução das transições em regras da gramática (cont.):

- $\delta(s_0, 0, \mathtt{A}) = \{(s_0, \mathtt{AA})\}$ dá origem às quatro regras também:

$$
\begin{array}{lclclcl}
[s_0, \mathtt{A}, s_1] & \to & 0[s_0, \mathtt{A}, s_0][s_0, \mathtt{A}, s_1] & \quad & [s_0, \mathtt{A}, s_1] & \to & 0[s_0, \mathtt{A}, s_1][s_1, \mathtt{A}, s_1] \\
[s_0, \mathtt{A}, s_0] & \to & 0[s_0, \mathtt{A}, s_0][s_0, \mathtt{A}, s_0] & & [s_0, \mathtt{A}, s_0] & \to & 0[s_0, \mathtt{A}, s_1][s_1, \mathtt{A}, s_0]
\end{array}
$$

- $\delta(s_0, 1, \mathtt{A}) = \{(s_1, \varepsilon)\}$ dá apenas a regra:

$$[s_0, \mathtt{A}, s_1] \quad \to \quad 1$$

- $\delta(s_1, 1, \mathtt{A}) = \{(s_1, \varepsilon)\}$ dá apenas a regra:

$$[s_1, \mathtt{A}, s_1] \quad \to \quad 1$$

# Exemplo (cont.): AP para GIC

**Exemplo (cont)** AP $\mathcal{A} = (\{s_0, s_1\}, \{0, 1\}, \{\mathtt{Z}, \mathtt{A}\}, \delta, s_0, \mathtt{Z}, \{\ \})$ com aceitação por pilha vazia e $\delta$ dada por:

$$\begin{array}{rcl@{\qquad}rcl}
\delta(s_0, \varepsilon, \mathtt{Z}) & = & \{(s_1, \mathtt{Z})\} & \delta(s_0, 0, \mathtt{Z}) & = & \{(s_0, \mathtt{AZ})\} \\
\delta(s_0, 0, \mathtt{A}) & = & \{(s_0, \mathtt{AA})\} & \delta(s_0, 1, \mathtt{A}) & = & \{(s_1, \varepsilon)\} \\
\delta(s_1, 1, \mathtt{A}) & = & \{(s_1, \varepsilon)\} & \delta(s_1, 1, \mathtt{Z}) & = & \{(s_1, \varepsilon), (s_1, \mathtt{Z})\}
\end{array}$$

$\boxed{\text{Tradução das transições em regras da gramática (cont.):}}$

- $\delta(s_1, 1, \mathtt{Z}) = \{(s_1, \varepsilon), (s_1, \mathtt{Z})\}$ dá:

$$[s_1, \mathtt{Z}, s_1] \quad \to \quad 1$$

porque $(s_1, \varepsilon) \in \delta(s_1, 1, \mathtt{Z})$. E, porque $(s_1, \mathtt{Z}) \in \delta(s_1, 1, \mathtt{Z})$ temos ainda

$$[s_1, \mathtt{Z}, s_1] \quad \to \quad 1[s_1, \mathtt{Z}, s_1] \qquad [s_1, \mathtt{Z}, s_0] \quad \to \quad 1[s_1, \mathtt{Z}, s_0]$$

# Exemplo (cont.): AP para GIC

Obtivemos a gramática:

$$
\begin{array}{lcl}
S & \rightarrow & [s_0, \mathtt{Z}, s_0] \\
S & \rightarrow & [s_0, \mathtt{Z}, s_1] \\
[s_0, \mathtt{Z}, s_0] & \rightarrow & [s_1, \mathtt{Z}, s_0] \\
[s_0, \mathtt{Z}, s_1] & \rightarrow & [s_1, \mathtt{Z}, s_1] \\
[s_0, \mathtt{Z}, s_1] & \rightarrow & 0[s_0, \mathtt{A}, s_0][s_0, \mathtt{Z}, s_1] \\
[s_0, \mathtt{Z}, s_1] & \rightarrow & 0[s_0, \mathtt{A}, s_1][s_1, \mathtt{Z}, s_1] \\
[s_0, \mathtt{Z}, s_0] & \rightarrow & 0[s_0, \mathtt{A}, s_0][s_0, \mathtt{Z}, s_0] \\
[s_0, \mathtt{Z}, s_0] & \rightarrow & 0[s_0, \mathtt{A}, s_1][s_1, \mathtt{Z}, s_0] \\
[s_0, \mathtt{A}, s_0] & \rightarrow & 0[s_0, \mathtt{A}, s_0][s_0, \mathtt{A}, s_0] \\
[s_0, \mathtt{A}, s_0] & \rightarrow & 0[s_0, \mathtt{A}, s_1][s_1, \mathtt{A}, s_0] \\
[s_0, \mathtt{A}, s_1] & \rightarrow & 0[s_0, \mathtt{A}, s_0][s_0, \mathtt{A}, s_1] \\
[s_0, \mathtt{A}, s_1] & \rightarrow & 0[s_0, \mathtt{A}, s_1][s_1, \mathtt{A}, s_1] \\
[s_0, \mathtt{A}, s_1] & \rightarrow & 1 \\
[s_1, \mathtt{A}, s_1] & \rightarrow & 1 \\
[s_1, \mathtt{Z}, s_1] & \rightarrow & 1 \\
[s_1, \mathtt{Z}, s_1] & \rightarrow & 1[s_1, \mathtt{Z}, s_1] \\
[s_1, \mathtt{Z}, s_0] & \rightarrow & 1[s_1, \mathtt{Z}, s_0]
\end{array}
$$

# Exemplo (cont.): AP para GIC

Podemos substituir as designações das variáveis, ficando com:

$$
\begin{array}{lcl}
S & \rightarrow & C \mid D \\
C & \rightarrow & E \mid 0MC \mid 0RE \\
D & \rightarrow & F \mid 0MD \mid 0RF \\
M & \rightarrow & 0MM \mid 0RT \\
R & \rightarrow & 0MR \mid 0RW \mid 1 \\
W & \rightarrow & 1 \\
F & \rightarrow & 1 \mid 1F \\
E & \rightarrow & 1E
\end{array}
$$

Torna-se evidente que $T$ é desnecessária pois não há qualquer regra que defina $T$.
Também $\mathcal{L}(E) = \emptyset$, pois não é possível eliminar $E$ numa derivação.
Se retiramos $E$ e $T$ e as regras em que ocorrem, ficamos apenas com uma regra para $C$, que é $C \rightarrow 0MC$, e para $M$, que é $M \rightarrow 0MM$. Logo, $\mathcal{L}(C) = \mathcal{L}(M) = \emptyset$ e podemos retirar $C$ e $M$.

# Exemplo (cont.): AP para GIC

Concluimos que a gramática

$$
\begin{array}{lcl}
S & \rightarrow & C \mid D \\
C & \rightarrow & E \mid 0MC \mid 0RE \\
D & \rightarrow & F \mid 0MD \mid 0RF \\
M & \rightarrow & 0MM \mid 0RT \\
R & \rightarrow & 0MR \mid 0RW \mid 1 \\
W & \rightarrow & 1 \\
F & \rightarrow & 1 \mid 1F \\
E & \rightarrow & 1E
\end{array}
$$

pode ser simplificada em

$$
\begin{array}{lcl}
S & \rightarrow & D \\
D & \rightarrow & F \mid 0RF \\
R & \rightarrow & 0RW \mid 1 \\
W & \rightarrow & 1 \\
F & \rightarrow & 1 \mid 1F
\end{array}
$$

# Exemplo (cont.): AP para GIC

Podemos ainda retirar $D$ se, observarmos que $S$ se reescreve em $D$ e que podemos substituir $S \to D$, por duas regras. **Finalmente, ficamos com a gramática**:

$$\begin{array}{lcl} S & \to & F \mid 0RF \\ R & \to & 0R1 \mid 1 \\ F & \to & 1 \mid 1F \end{array}$$

Note que, como $W$ só gera 1, eliminámos $W$, substituindo por 1.

É interessante observar que, para as designações iniciais, as regras que obtivemos são:

$$\begin{array}{lclcl} S & \to & [s_1, \mathtt{Z}, s_1] & \mid & 0[s_0, \mathtt{A}, s_1][s_1, \mathtt{Z}, s_1] \\ [s_0, \mathtt{A}, s_1] & \to & 0[s_0, \mathtt{A}, s_1]1 & \mid & 1 \\ [s_1, \mathtt{Z}, s_1] & \to & 1 & \mid & 1[s_1, \mathtt{Z}, s_1] \end{array}$$

Estas regras traduzem melhor o comportamento real do autómato: após analisar palavras de $\mathcal{L}(\mathcal{A})$, está sempre em $s_1$; depois de retirar um A, só pode estar em $s_1$.

# LICs são reconhecidas por APs

**Definição:** Uma GIC $G = (V, \Sigma, P, S)$ está na **forma normal de Greibach** sse as regras são da forma $A \rightarrow a\gamma$, com $\gamma \in V^\star$ e $a \in \Sigma$.

**Conversão de GIC para F.N.Greibach**

Qualquer GIC que não gere $\varepsilon$ pode ser reduzida à forma normal de Greibach.

Prova (não trivial): apresenta o algoritmo de conversão (consultar bibliografia).

# LICs são reconhecidas por APs

### Conversão de GIC na F.N.Greibach para AP

Dada uma GIC $G = (V, \Sigma, P, S)$ na forma normal de Greibach, a linguagem $\mathcal{L}(G)$ é aceite por pilha vazia pelo autómato de pilha $\mathcal{A} = (\{q\}, \Sigma, V, \delta, q, S)$, com $\delta(q, a, X) = \{(q, \gamma) \mid (X \to a\gamma) \in P\}$.

**Ideia:** As derivações pela esquerda em $G$ correspondem às sequências de mudança de configuração em $\mathcal{A}$.

**Observação:**

A restrição de que $\varepsilon \notin L(G)$, se $G$ está na f. n. Greibach, pode ser contornada para concluir que LIC $L$ pode ser reconhecida por AP. Existem extensões que admitem a regra $S \to \varepsilon$, para o símbolo inicial, se $S$ não ocorrer no lado direito de nenhuma regra.

# LICs são reconhecidas por APs

**Exemplo:**
A linguagem das expressões regulares sobre $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ pode ser definida pela gramática $G = (\{S\}, \{\boxed{\varepsilon}, \emptyset, \mathtt{)}, \mathtt{(}, \mathtt{+}, ^\star\} \cup \Sigma, P, S)$ com produções:

$$S \quad \to \quad \boxed{\varepsilon} \mid \mathtt{a} \mid \mathtt{b} \mid \emptyset \mid (SS) \mid (S{+}S) \mid (S^\star)$$

Usámos $\boxed{\varepsilon}$ para distinguir o símbolo $\varepsilon$ da palavra vazia.

**A conversão desta GIC à F. N. Greibach é trivial**

$$\begin{array}{lcl} S & \to & \boxed{\varepsilon} \mid \mathtt{a} \mid \mathtt{b} \mid \emptyset \mid (SSF \mid (SMSF \mid (SKF \\ F & \to & ) \\ M & \to & + \\ K & \to & ^\star \end{array}$$

# LICs são reconhecidas por APs

## Exemplo (cont.):

O **autómato de pilha obtido pela método de conversão** da GIC na F. N. Greibach para AP com aceitação por pilha vazia é

$$\mathcal{A} = (\{q\}, \{\boxed{\varepsilon}, \emptyset, \texttt{)}, \texttt{(}, +, ^\star\} \cup \Sigma, \{S, F, M, K\}, \delta, q, S, \{\})$$

com

$$\begin{array}{rcl}
\delta(q, S, \texttt{a}) & = & \{(q, \varepsilon)\} \\
\delta(q, S, \texttt{b}) & = & \{(q, \varepsilon)\} \\
\delta(q, S, \emptyset) & = & \{(q, \varepsilon)\} \\
\delta(q, S, \boxed{\varepsilon}) & = & \{(q, \varepsilon)\} \\
\delta(q, S, \texttt{(}) & = & \{(q, SSF)\} \\
\delta(q, S, \texttt{(}) & = & \{(q, SMSF)\} \\
\delta(q, S, \texttt{(}) & = & \{(q, SKF)\} \\
\delta(q, M, +) & = & \{(q, \varepsilon)\} \\
\delta(q, K, ^\star) & = & \{(q, \varepsilon)\} \\
\delta(q, K, \texttt{)}) & = & \{(q, \varepsilon)\}
\end{array}$$

# Forma Normal de Chomsky

**Definição:** Uma GIC $G = (V, \Sigma, P, S)$ está na **forma normal de Chomsky** sse as regras são da forma $A \to BC$ e $A \to a$, com $B, C, A \in V$ e $a \in \Sigma$.

**Exemplos:**

$\mathcal{G}_1 = (\{S, T, A, B, X\}, \{\mathtt{a}, \mathtt{b}\}, P, S)$
com $P$ dado por:

$$
\begin{array}{lcl}
T & \to & \mathtt{a} \mid AT \\
S & \to & \mathtt{a} \mid AT \mid XB \\
X & \to & AS \\
B & \to & \mathtt{b} \\
A & \to & \mathtt{a}
\end{array}
$$

$\mathcal{G}_1$ está na FN Chomsky

$\mathcal{G}_2 = (\{S, A, B, C, D\}, \{0, 1, 2\}, P, S)$
com $P$ dado por:

$$
\begin{array}{lcl}
S & \to & 0S \mid B \\
B & \to & 1B \mid C \\
C & \to & D \mid C2 \mid 2 \mid S \\
D & \to & 0D1 \mid 01
\end{array}
$$

$\mathcal{G}_2$ não está na FN Chomsky

# Forma Normal de Chomsky

**Definição:** Uma GIC $G = (V, \Sigma, P, S)$ está na **forma normal de Chomsky** sse as regras são da forma $A \to BC$ e $A \to a$, com $B, C, A \in V$ e $a \in \Sigma$.

**Exemplos:**

$\mathcal{G}_1 = (\{S, T, A, B, X\}, \{\mathtt{a}, \mathtt{b}\}, P, S)$
com $P$ dado por:

$$
\begin{array}{lcl}
T & \to & \mathtt{a} \mid AT \\
S & \to & \mathtt{a} \mid AT \mid XB \\
X & \to & AS \\
B & \to & \mathtt{b} \\
A & \to & \mathtt{a}
\end{array}
$$

$\mathcal{G}_1$ está na FN Chomsky

$\mathcal{G}_2 = (\{S, A, B, C, D\}, \{0, 1, 2\}, P, S)$
com $P$ dado por:

$$
\begin{array}{lcl}
S & \to & 0S \mid B \\
B & \to & 1B \mid C \\
C & \to & D \mid C2 \mid 2 \mid S \\
D & \to & 0D1 \mid 01
\end{array}
$$

$\mathcal{G}_2$ não está na FN Chomsky

# Forma Normal de Chomsky

**Definição:** Uma GIC $G = (V, \Sigma, P, S)$ está na **forma normal de Chomsky** sse as regras são da forma $A \to BC$ e $A \to a$, com $B, C, A \in V$ e $a \in \Sigma$.

**Exemplos:**

$\mathcal{G}_1 = (\{S, T, A, B, X\}, \{\mathtt{a}, \mathtt{b}\}, P, S)$
com $P$ dado por:

$$
\begin{array}{lcl}
T & \to & \mathtt{a} \mid AT \\
S & \to & \mathtt{a} \mid AT \mid XB \\
X & \to & AS \\
B & \to & \mathtt{b} \\
A & \to & \mathtt{a}
\end{array}
$$

$\mathcal{G}_1$ está na FN Chomsky

$\mathcal{G}_2 = (\{S, A, B, C, D\}, \{0, 1, 2\}, P, S)$
com $P$ dado por:

$$
\begin{array}{lcl}
S & \to & 0S \mid B \\
B & \to & 1B \mid C \\
C & \to & D \mid C2 \mid 2 \mid S \\
D & \to & 0D1 \mid 01
\end{array}
$$

$\mathcal{G}_2$ não está na FN Chomsky

# Redução à forma normal de Chomsky (FNC)

**Exemplo:** Transformação de $\mathcal{G}_2$ numa **GIC equivalente** mas que está **na FNC**, sendo $\mathcal{G}_2 = (\{S, A, B, C, D\}, \{0, 1, 2\}, P, S)$, com $P$ dado por

$$\begin{array}{lll} S & \to & 0S \mid B \\ B & \to & 1B \mid C \end{array} \qquad \begin{array}{lll} C & \to & D \mid C2 \mid 2 \mid S \\ D & \to & 0D1 \mid 01 \end{array}$$

- Apenas as regras da forma $X \to a$, com $a \in \Sigma$, podem ter terminais. **Corrige-se, introduzindo novas variáveis**.

$$\begin{array}{lll} S & \to & ZS \mid B \\ B & \to & UB \mid C \\ C & \to & D \mid CK \mid 2 \mid S \\ D & \to & ZDU \mid ZU \end{array} \qquad \begin{array}{lll} Z & \to & 0 \\ U & \to & 1 \\ K & \to & 2 \end{array}$$

- Não podemos ter três ou mais variáveis no lado direito das regras. **Corrige-se, introduzindo novas variáveis**.

  Podemos substituir $D \to ZDU$ por $\left| \begin{array}{l} D \to MU \\ M \to ZD \end{array} \right.$

- Mas, o que fazer com as **produções unitárias**, i.e., do tipo $X \to Y$, com $Y \in V$?

# Redução à forma normal de Chomsky (FNC)

**Exemplo:** Transformação de $\mathcal{G}_2$ numa **GIC equivalente** mas que está **na FNC**, sendo $\mathcal{G}_2 = (\{S, A, B, C, D\}, \{0, 1, 2\}, P, S)$, com $P$ dado por

$$\begin{array}{lll} S & \to & 0S \mid B \\ B & \to & 1B \mid C \end{array} \qquad \begin{array}{lll} C & \to & D \mid C2 \mid 2 \mid S \\ D & \to & 0D1 \mid 01 \end{array}$$

- Apenas as regras da forma $X \to a$, com $a \in \Sigma$, podem ter terminais.
  **Corrige-se, introduzindo novas variáveis**.

$$\begin{array}{lll} S & \to & ZS \mid B \\ B & \to & UB \mid C \\ C & \to & D \mid CK \mid 2 \mid S \\ D & \to & ZDU \mid ZU \end{array} \qquad \begin{array}{lll} Z & \to & 0 \\ U & \to & 1 \\ K & \to & 2 \end{array}$$

- Não podemos ter três ou mais variáveis no lado direito das regras.
  **Corrige-se, introduzindo novas variáveis**.

  Podemos substituir $D \to ZDU$ por $\begin{array}{|l} D \to MU \\ M \to ZD \end{array}$

- Mas, o que fazer com as **produções unitárias**, i.e., do tipo $X \to Y$, com $Y \in V$?

# Redução à forma normal de Chomsky (FNC)

**Exemplo:** Transformação de $\mathcal{G}_2$ numa **GIC equivalente** mas que está **na FNC**, sendo $\mathcal{G}_2 = (\{S, A, B, C, D\}, \{0, 1, 2\}, P, S)$, com $P$ dado por

$$\begin{array}{lll} S & \rightarrow & 0S \mid B \\ B & \rightarrow & 1B \mid C \end{array} \qquad \begin{array}{lll} C & \rightarrow & D \mid C2 \mid 2 \mid S \\ D & \rightarrow & 0D1 \mid 01 \end{array}$$

- Apenas as regras da forma $X \rightarrow a$, com $a \in \Sigma$, podem ter terminais. **Corrige-se, introduzindo novas variáveis**.

$$\begin{array}{lll} S & \rightarrow & ZS \mid B \\ B & \rightarrow & UB \mid C \\ C & \rightarrow & D \mid CK \mid 2 \mid S \\ D & \rightarrow & ZDU \mid ZU \end{array} \qquad \begin{array}{lll} Z & \rightarrow & 0 \\ U & \rightarrow & 1 \\ K & \rightarrow & 2 \end{array}$$

- Não podemos ter três ou mais variáveis no lado direito das regras. **Corrige-se, introduzindo novas variáveis**.

Podemos substituir $D \rightarrow ZDU$ por $\begin{array}{|l} D \rightarrow MU \\ M \rightarrow ZD \end{array}$

- Mas, o que fazer com as **produções unitárias**, i.e., do tipo $X \rightarrow Y$, com $Y \in V$?

# Redução à forma normal de Chomsky (FNC)

**Exemplo:** Transformação de $\mathcal{G}_2$ numa **GIC equivalente** mas que está **na FNC**, sendo $\mathcal{G}_2 = (\{S, A, B, C, D\}, \{0, 1, 2\}, P, S)$, com $P$ dado por

$$\begin{array}{lcl} S & \to & 0S \mid B \\ B & \to & 1B \mid C \end{array} \qquad \begin{array}{lcl} C & \to & D \mid C2 \mid 2 \mid S \\ D & \to & 0D1 \mid 01 \end{array}$$

- Apenas as regras da forma $X \to a$, com $a \in \Sigma$, podem ter terminais.
  **Corrige-se, introduzindo novas variáveis**.

$$\begin{array}{lcl} S & \to & ZS \mid B \\ B & \to & UB \mid C \\ C & \to & D \mid CK \mid 2 \mid S \\ D & \to & ZDU \mid ZU \end{array} \qquad \begin{array}{lcl} Z & \to & 0 \\ U & \to & 1 \\ K & \to & 2 \end{array}$$

- Não podemos ter três ou mais variáveis no lado direito das regras.
  **Corrige-se, introduzindo novas variáveis**.

  Podemos substituir $D \to ZDU$ por $\left| \begin{array}{l} D \to MU \\ M \to ZD \end{array} \right.$

- Mas, o que fazer com as **produções unitárias**, i.e., do tipo $X \to Y$, com $Y \in V$?

# Redução à forma normal de Chomsky (FNC)

**Exemplo:** Transformação de $\mathcal{G}_2$ numa **GIC equivalente** mas que está **na FNC**, sendo $\mathcal{G}_2 = (\{S, A, B, C, D\}, \{0, 1, 2\}, P, S)$, com $P$ dado por

$$\begin{array}{lcl} S & \to & 0S \mid B \\ B & \to & 1B \mid C \end{array} \qquad \begin{array}{lcl} C & \to & D \mid C2 \mid 2 \mid S \\ D & \to & 0D1 \mid 01 \end{array}$$

- Apenas as regras da forma $X \to a$, com $a \in \Sigma$, podem ter terminais.
  **Corrige-se, introduzindo novas variáveis**.

$$\begin{array}{lcl} S & \to & ZS \mid B \\ B & \to & UB \mid C \\ C & \to & D \mid CK \mid 2 \mid S \\ D & \to & ZDU \mid ZU \end{array} \qquad \begin{array}{lcl} Z & \to & 0 \\ U & \to & 1 \\ K & \to & 2 \end{array}$$

- Não podemos ter três ou mais variáveis no lado direito das regras.
  **Corrige-se, introduzindo novas variáveis**.

  Podemos substituir $D \to ZDU$ por $\left| \begin{array}{l} D \to MU \\ M \to ZD \end{array} \right.$

- Mas, o que fazer com as **produções unitárias**, i.e., do tipo $X \to Y$, com $Y \in V$?

# Redução à forma normal de Chomsky (FNC)

**Exemplo (cont):**
O que fazer com as **produções unitárias**, i.e., do tipo $X \to Y$, com $Y \in V$?

$$
\begin{array}{lcl}
S & \to & ZS \mid B \\
B & \to & UB \mid C \\
C & \to & D \mid CK \mid 2 \mid S \\
D & \to & MU \mid ZU
\end{array}
\qquad
\begin{array}{lcl}
Z & \to & 0 \\
U & \to & 1 \\
K & \to & 2 \\
M & \to & ZD
\end{array}
$$

**Gramática equivalente sem regras unitárias**: quando não há variáveis que produzem $\varepsilon$, podemos eliminar qualquer regra unitária $X \to Y$ por substituição de $Y$ usando as regras $Y \to \beta$, com $\beta \neq X$. O processo terminará sem regras unitárias mesmo que em passos intermédios se criem novas regras unitárias.

$$
\begin{array}{lcl}
S & \to & ZS \mid UB \mid MU \mid ZU \mid CK \mid 2 \\
B & \to & UB \mid MU \mid ZU \mid CK \mid 2 \mid S \\
C & \to & MU \mid ZU \mid CK \mid 2 \mid S \\
D & \to & MU \mid ZU
\end{array}
\qquad
\begin{array}{lcl}
Z & \to & 0 \\
U & \to & 1 \\
K & \to & 2 \\
M & \to & ZD
\end{array}
$$

Ainda tem regras unitárias...

# Redução à forma normal de Chomsky (FNC)

**Exemplo (cont):**
O que fazer com as **produções unitárias**, i.e., do tipo $X \to Y$, com $Y \in V$?

$$\begin{array}{lll} S & \to & ZS \mid B \\ B & \to & UB \mid C \\ C & \to & D \mid CK \mid 2 \mid S \\ D & \to & MU \mid ZU \end{array} \qquad \begin{array}{lll} Z & \to & 0 \\ U & \to & 1 \\ K & \to & 2 \\ M & \to & ZD \end{array}$$

**Gramática equivalente sem regras unitárias**: quando não há variáveis que produzem $\varepsilon$, podemos eliminar qualquer regra unitária $X \to Y$ por substituição de $Y$ usando as regras $Y \to \beta$, com $\beta \neq X$. O processo terminará sem regras unitárias mesmo que em passos intermédios se criem novas regras unitárias.

$$\begin{array}{lll} S & \to & ZS \mid UB \mid MU \mid ZU \mid CK \mid 2 \\ B & \to & UB \mid MU \mid ZU \mid CK \mid 2 \mid S \\ C & \to & MU \mid ZU \mid CK \mid 2 \mid S \\ D & \to & MU \mid ZU \end{array} \qquad \begin{array}{lll} Z & \to & 0 \\ U & \to & 1 \\ K & \to & 2 \\ M & \to & ZD \end{array}$$

Ainda tem regras unitárias...

# Redução à forma normal de Chomsky (FNC)

**Exemplo (cont):**
O que fazer com as **produções unitárias**, i.e., do tipo $X \to Y$, com $Y \in V$?

$$
\begin{array}{lcl}
S & \to & ZS \mid B \\
B & \to & UB \mid C \\
C & \to & D \mid CK \mid 2 \mid S \\
D & \to & MU \mid ZU
\end{array}
\qquad
\begin{array}{lcl}
Z & \to & 0 \\
U & \to & 1 \\
K & \to & 2 \\
M & \to & ZD
\end{array}
$$

**Gramática equivalente sem regras unitárias**: quando não há variáveis que produzem $\varepsilon$, podemos eliminar qualquer regra unitária $X \to Y$ por substituição de $Y$ usando as regras $Y \to \beta$, com $\beta \neq X$. O processo terminará sem regras unitárias mesmo que em passos intermédios se criem novas regras unitárias.

$$
\begin{array}{lcl}
S & \to & ZS \mid UB \mid MU \mid ZU \mid CK \mid 2 \\
B & \to & UB \mid MU \mid ZU \mid CK \mid 2 \mid S \\
C & \to & MU \mid ZU \mid CK \mid 2 \mid S \\
D & \to & MU \mid ZU
\end{array}
\qquad
\begin{array}{lcl}
Z & \to & 0 \\
U & \to & 1 \\
K & \to & 2 \\
M & \to & ZD
\end{array}
$$

Ainda tem regras unitárias...

# Redução à forma normal de Chomsky (FNC)

**Exemplo (cont):**

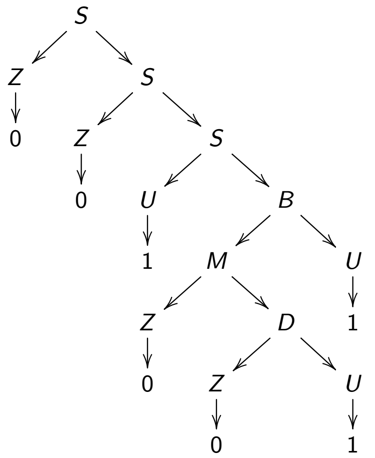
$$
\begin{array}{lcl}
S & \rightarrow & ZS \mid UB \mid MU \mid ZU \mid CK \mid 2 \\
B & \rightarrow & UB \mid MU \mid ZU \mid CK \mid 2 \mid S \\
C & \rightarrow & MU \mid ZU \mid CK \mid 2 \mid S \\
D & \rightarrow & MU \mid ZU
\end{array}
\qquad
\begin{array}{lcl}
Z & \rightarrow & 0 \\
U & \rightarrow & 1 \\
K & \rightarrow & 2 \\
M & \rightarrow & ZD
\end{array}
$$

Substituindo as regras unitárias $B \rightarrow S$ e $C \rightarrow S$, e eliminando as repetições, obtemos uma **GIC equivalente mas na forma normal de Chomsky:**

$$
\begin{array}{lcl}
S & \rightarrow & ZS \mid UB \mid MU \mid ZU \mid CK \mid 2 \\
B & \rightarrow & UB \mid MU \mid ZU \mid CK \mid 2 \mid ZS \\
C & \rightarrow & MU \mid ZU \mid CK \mid 2 \mid ZS \mid UB \\
D & \rightarrow & MU \mid ZU
\end{array}
\qquad
\begin{array}{lcl}
Z & \rightarrow & 0 \\
U & \rightarrow & 1 \\
K & \rightarrow & 2 \\
M & \rightarrow & ZD
\end{array}
$$

# Forma das árvores sintáticas para GICs na FN Chomsky

$$
\begin{array}{lcl}
S & \to & ZS \mid UB \mid MU \\
S & \to & ZU \mid CK \mid 2 \\
B & \to & UB \mid MU \mid ZU \mid CK \\
B & \to & 2 \mid ZS \\
C & \to & MU \mid ZU \mid CK \\
C & \to & 2 \mid ZS \mid UB \\
D & \to & MU \mid ZU \\
Z & \to & 0 \\
U & \to & 1 \\
K & \to & 2 \\
M & \to & ZD
\end{array}
$$




Para gramáticas na FN Chomsky, as árvores sintáticas são árvores binárias, tais que: os filhos dos **nós que têm só um filho** são terminais (i.e., símbolo de $\Sigma$); os filhos dos **nós que têm dois filhos** são não terminais.

# Algoritmos de decisão para $x \in \mathcal{L}(G)$

Como decidir se $x \in \mathcal{L}(\mathcal{G})$, quando $\mathcal{G}$ está na forma normal de Chomsky?

Se $\mathcal{G}$ não estiver na FN Chomsky e quisermos decidir se $x \in \mathcal{L}(\mathcal{G})$, podemos começar por converter $\mathcal{G}$ à FN Chomsky.

- **Algoritmo de pesquisa exaustiva "força-bruta"**: efetuar *derivações pela esquerda a partir de* $S$ até obter $x$ ou concluir que $x$ já não pode ser obtida, interrompendo derivações $S \Rightarrow^\star \gamma \Rightarrow \ldots$ que não podem gerar $x$, por $\gamma$ conter terminais não compatíveis com $x$ ou por $|\gamma| > |x|$.
  - Para cada $x \in \Sigma^\star$, a procura **termina** sempre, dando uma resposta ("sim" ou "não") à questão "$x \in \mathcal{L}(\mathcal{G})$?". O algoritmo é **ineficiente**.
  - Para ver que termina, note-se que, na FN Chomsky, nenhuma variável produz $\varepsilon$ nem uma só variável. E, sempre que usamos uma regra $A \to BC$, aumentamos o comprimento da forma final em pelo menos 1 unidade. Logo, no pior caso, terminamos uma derivação se $|\gamma| > |x|$.
- Existem algoritmos melhores, como o **Algoritmo CYK** (algoritmo de Cocke-Younger-Kasami), que introduzimos a seguir.

# Algoritmos de decisão para $x \in \mathcal{L}(G)$

Como decidir se $x \in \mathcal{L}(\mathcal{G})$, quando $\mathcal{G}$ está na forma normal de Chomsky?

Se $\mathcal{G}$ não estiver na FN Chomsky e quisermos decidir se $x \in \mathcal{L}(\mathcal{G})$, podemos começar por converter $\mathcal{G}$ à FN Chomsky.

- **Algoritmo de pesquisa exaustiva "força-bruta"**: efetuar *derivações pela esquerda a partir de* $S$ até obter $x$ ou concluir que $x$ já não pode ser obtida, interrompendo derivações $S \Rightarrow^\star \gamma \Rightarrow \ldots$ que não podem gerar $x$, por $\gamma$ conter terminais não compatíveis com $x$ ou por $|\gamma| > |x|$.
  - Para cada $x \in \Sigma^\star$, a procura **termina** sempre, dando uma resposta ("sim" ou "não") à questão "$x \in \mathcal{L}(\mathcal{G})$?". O algoritmo é **ineficiente**.
  - Para ver que termina, note-se que, na FN Chomsky, nenhuma variável produz $\varepsilon$ nem uma só variável. E, sempre que usamos uma regra $A \to BC$, aumentamos o comprimento da forma final em pelo menos 1 unidade. Logo, no pior caso, terminamos uma derivação se $|\gamma| > |x|$.
- Existem algoritmos melhores, como o **Algoritmo CYK** (algoritmo de Cocke-Younger-Kasami), que introduzimos a seguir.

# Algoritmos de decisão para $x \in \mathcal{L}(G)$

Como decidir se $x \in \mathcal{L}(\mathcal{G})$, quando $\mathcal{G}$ está na forma normal de Chomsky?

Se $\mathcal{G}$ não estiver na FN Chomsky e quisermos decidir se $x \in \mathcal{L}(\mathcal{G})$, podemos começar por converter $\mathcal{G}$ à FN Chomsky.

- **Algoritmo de pesquisa exaustiva "força-bruta"**: efetuar *derivações pela esquerda a partir de* $S$ até obter $x$ ou concluir que $x$ já não pode ser obtida, interrompendo derivações $S \Rightarrow^\star \gamma \Rightarrow \ldots$ que não podem gerar $x$, por $\gamma$ conter terminais não compatíveis com $x$ ou por $|\gamma| > |x|$.
  - Para cada $x \in \Sigma^\star$, a procura **termina** sempre, dando uma resposta ("sim" ou "não") à questão "$x \in \mathcal{L}(\mathcal{G})$?". O algoritmo é **ineficiente**.
  - Para ver que termina, note-se que, na FN Chomsky, nenhuma variável produz $\varepsilon$ nem uma só variável. E, sempre que usamos uma regra $A \to BC$, aumentamos o comprimento da forma final em pelo menos 1 unidade. Logo, no pior caso, terminamos uma derivação se $|\gamma| > |x|$.
- Existem algoritmos melhores, como o **Algoritmo CYK** (algoritmo de Cocke-Younger-Kasami), que introduzimos a seguir.

# Algoritmos de decisão para $x \in \mathcal{L}(G)$

Como decidir se $x \in \mathcal{L}(\mathcal{G})$, quando $\mathcal{G}$ está na forma normal de Chomsky?

Se $\mathcal{G}$ não estiver na FN Chomsky e quisermos decidir se $x \in \mathcal{L}(\mathcal{G})$, podemos começar por converter $\mathcal{G}$ à FN Chomsky.

- **Algoritmo de pesquisa exaustiva "força-bruta"**: efetuar *derivações pela esquerda a partir de* $S$ até obter $x$ ou concluir que $x$ já não pode ser obtida, interrompendo derivações $S \Rightarrow^\star \gamma \Rightarrow \ldots$ que não podem gerar $x$, por $\gamma$ conter terminais não compatíveis com $x$ ou por $|\gamma| > |x|$.
  - Para cada $x \in \Sigma^\star$, a procura **termina** sempre, dando uma resposta ("sim" ou "não") à questão "$x \in \mathcal{L}(\mathcal{G})$?". O algoritmo é **ineficiente**.
  - Para ver que termina, note-se que, na FN Chomsky, nenhuma variável produz $\varepsilon$ nem uma só variável. E, sempre que usamos uma regra $A \to BC$, aumentamos o comprimento da forma final em pelo menos 1 unidade. Logo, no pior caso, terminamos uma derivação se $|\gamma| > |x|$.
- Existem algoritmos melhores, como o **Algoritmo CYK** (algoritmo de Cocke-Younger-Kasami), que introduzimos a seguir.

# Algoritmos de decisão para $x \in \mathcal{L}(G)$

Como decidir se $x \in \mathcal{L}(\mathcal{G})$, quando $\mathcal{G}$ está na forma normal de Chomsky?

Se $\mathcal{G}$ não estiver na FN Chomsky e quisermos decidir se $x \in \mathcal{L}(\mathcal{G})$, podemos começar por converter $\mathcal{G}$ à FN Chomsky.

- **Algoritmo de pesquisa exaustiva "força-bruta"**: efetuar *derivações pela esquerda a partir de* $S$ até obter $x$ ou concluir que $x$ já não pode ser obtida, interrompendo derivações $S \Rightarrow^\star \gamma \Rightarrow \ldots$ que não podem gerar $x$, por $\gamma$ conter terminais não compatíveis com $x$ ou por $|\gamma| > |x|$.
  - Para cada $x \in \Sigma^\star$, a procura **termina** sempre, dando uma resposta ("sim" ou "não") à questão "$x \in \mathcal{L}(\mathcal{G})$?". O algoritmo é **ineficiente**.
  - Para ver que termina, note-se que, na FN Chomsky, nenhuma variável produz $\varepsilon$ nem uma só variável. E, sempre que usamos uma regra $A \to BC$, aumentamos o comprimento da forma final em pelo menos 1 unidade. Logo, no pior caso, terminamos uma derivação se $|\gamma| > |x|$.

- Existem algoritmos melhores, como o **Algoritmo CYK** (algoritmo de Cocke-Younger-Kasami), que introduzimos a seguir.

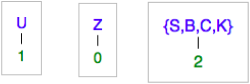
# Ideia do Algoritmo de Cocke-Younger-Kasami (CYK)

$\boxed{\text{Será que } 102 \in \mathcal{L}(\mathcal{G})?}$ **Tentemos construir a árvore das folhas para a raíz.**

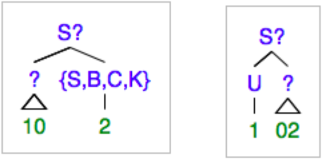
$$
\begin{array}{lcl}
S & \to & ZS \mid UB \mid MU \\
S & \to & ZU \mid CK \mid 2 \\
B & \to & UB \mid MU \mid CK \\
B & \to & ZU \mid 2 \mid ZS \\
C & \to & MU \mid ZU \mid CK \\
C & \to & 2 \mid ZS \mid UB \\
D & \to & MU \mid ZU \\
Z & \to & 0 \\
U & \to & 1 \\
K & \to & 2 \\
M & \to & ZD
\end{array}
$$

$S$ **é o símbolo inicial**


Sumariando as alternativas para 2 ...



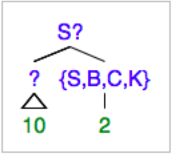

Se $102 \in \mathcal{L}(\mathcal{G})$, que tipo de estrutura teria?

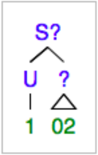


# Ideia do Algoritmo de Cocke-Younger-Kasami (CYK)

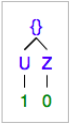
$$
\begin{aligned}
S &\rightarrow ZS \mid UB \mid MU \\
S &\rightarrow ZU \mid CK \mid 2 \\
B &\rightarrow UB \mid MU \mid CK \\
B &\rightarrow ZU \mid 2 \mid ZS \\
C &\rightarrow MU \mid ZU \mid CK \\
C &\rightarrow 2 \mid ZS \mid UB \\
D &\rightarrow MU \mid ZU \\
Z &\rightarrow 0 \\
U &\rightarrow 1 \\
K &\rightarrow 2 \\
M &\rightarrow ZD
\end{aligned}
$$

Se $102 \in \mathcal{L}(\mathcal{G})$, que tipo de estrutura teria?

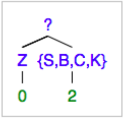



A da esquerda é impossível pois 10 é $UZ$ e nenhuma variável produz $UZ$.




E, a da direita? Que tipo pode ter 02?




Que variáveis podem produzir elementos de $\{Z\}\{S, B, C, K\} = \{ZS, ZB, ZC, ZK\}$?

# Ideia do Algoritmo de Cocke-Younger-Kasami (CYK)

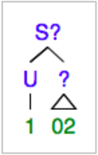
$$
\begin{array}{lcl}
S & \to & ZS \mid UB \mid MU \\
S & \to & ZU \mid CK \mid 2 \\
B & \to & UB \mid MU \mid CK \\
B & \to & ZU \mid 2 \mid ZS \\
C & \to & MU \mid ZU \mid CK \\
C & \to & 2 \mid ZS \mid UB \\
D & \to & MU \mid ZU \\
Z & \to & 0 \\
U & \to & 1 \\
K & \to & 2 \\
M & \to & ZD
\end{array}
$$

$S$ é o símbolo inicial de $\mathcal{G}$.

Vimos que 02 teria alguma forma de $\{Z\}\{S, B, C, K\}$ que é $\{ZS, ZB, ZC, ZK\}$.
Só $ZS$ ocorre nas produções. Pode ser $S$, $B$, ou $C$.

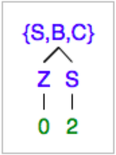
**O que concluimos para 102?**

102



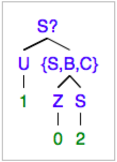

$\wedge$

02



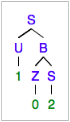

$\rightsquigarrow$

102




102 é $\{U\}\{S, B, C\} = \{US, UB, UC\}$. Só $UB$ ocorre nas regras. É produzido por $S$, $B$ e $C$. $\therefore S \Rightarrow^{\star}_{\mathcal{G}} 102$.

$102 \in \mathcal{L}(\mathcal{G})$.
A **árvore de derivação** de 102:



# Ideia do Algoritmo de Cocke-Younger-Kasami (CYK)

No **Algoritmo CYK**, aplicado à palavra $x_1x_2x_3 = 102$, construimos uma tabela.

$$
\begin{array}{lcl}
S & \rightarrow & ZS \mid UB \mid MU \\
S & \rightarrow & ZU \mid CK \mid 2 \\
B & \rightarrow & UB \mid MU \mid CK \\
B & \rightarrow & ZU \mid 2 \mid ZS \\
C & \rightarrow & MU \mid ZU \mid CK \\
C & \rightarrow & 2 \mid ZS \mid UB \\
D & \rightarrow & MU \mid ZU \\
Z & \rightarrow & 0 \\
U & \rightarrow & 1 \\
K & \rightarrow & 2 \\
M & \rightarrow & ZD
\end{array}
$$

$S$ é o símbolo inicial de $\mathcal{G}$.

| | | | |
|---|---|---|---|
| #3 | $x_1x_2x_3 = 102$ $\{S, B, C\}$ | | |
| #2 | $x_1x_2 = 10$ $\emptyset$ | $x_2x_3 = 02$ $\{S, B, C\}$ | |
| #1 | $x_1 = 1$ $\{U\}$ | $x_2 = 0$ $\{Z\}$ | $x_3 = 2$ $\{S, B, C, K\}$ |
| | 1 | 0 | 2 |

**Passos da construção:**

- Linha 1:

  Para $i = 1, 2, 3$, colocar na célula de $x_i$ o conjunto $\{X \mid X \rightarrow x_i$ é regra$\}$,

- Linha 2:

  $x_1x_2$ é $\{U\}\{Z\} = \{UZ\}$. Mas, $UZ$ não ocorre nas regras. Coloca $\emptyset$.

  $x_2x_3$ é $\{Z\}\{S, B, C, K\}$, que é $\{ZS, ZB, ZC, ZK\}$. Só $ZS$ ocorre nas regras, e tem-se $S \rightarrow ZS$, $B \rightarrow ZS$, e $C \rightarrow ZS$. Coloca $\{S, B, C\}$.

- Linha 3:

  $x_1x_2x_3$ pode **ser decomposto** como $x_1|x_2x_3$ **ou** $x_1x_2|x_3$.

  $x_1|x_2x_3$ é $\{U\}\{S, B, C\}$, ou seja $\{US, UB, UC\}$. Só $UB$ ocorre nas regras. $S$, $B$ e $C$ produzem $UB$.

  $x_1x_2|x_3$ é $\emptyset\{S, B, C, K\} = \emptyset$.

  Coloca $\{S, B, C\} \cup \{\,\} = \{S, B, C\}$.

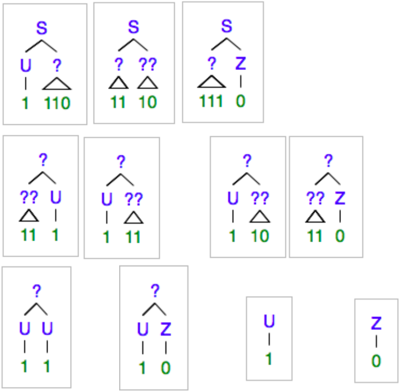
- **Conclusão:** $x_1x_2x_3 = 102$ pode ser $S$. Ou seja, $S \Rightarrow^{\star}_{\mathcal{G}} 102$. $102 \in \mathcal{L}(\mathcal{G})$.

# Ideia do Algoritmo de Cocke-Younger-Kasami (CYK)

Será que $1110 \in \mathcal{L}(\mathcal{G})$?

$$
\begin{array}{lcl}
S & \rightarrow & ZS \mid UB \mid MU \\
S & \rightarrow & ZU \mid CK \mid 2 \\
B & \rightarrow & UB \mid MU \mid ZU \mid CK \\
B & \rightarrow & 2 \mid ZS \\
C & \rightarrow & MU \mid ZU \mid CK \\
C & \rightarrow & 2 \mid ZS \mid UB \\
D & \rightarrow & MU \mid ZU \\
Z & \rightarrow & 0 \\
U & \rightarrow & 1 \\
K & \rightarrow & 2 \\
M & \rightarrow & ZD
\end{array}
$$

$S$ é o símbolo inicial.




Não há variáveis que produzam $UU$ nem $UZ$.

$\therefore$ $X \not\Rightarrow^\star 111$ e $X \not\Rightarrow^\star 110$, para todo $X \in V$.

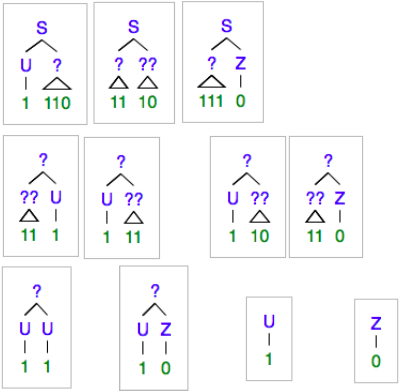
Logo, $S \not\Rightarrow^\star 1110$. Portanto, $1110 \notin \mathcal{L}(\mathcal{G})$.

# Ideia do Algoritmo de Cocke-Younger-Kasami (CYK)

Será que $1110 \in \mathcal{L}(\mathcal{G})$?

$$
\begin{array}{lcl}
S & \to & ZS \mid UB \mid MU \\
S & \to & ZU \mid CK \mid 2 \\
B & \to & UB \mid MU \mid ZU \mid CK \\
B & \to & 2 \mid ZS \\
C & \to & MU \mid ZU \mid CK \\
C & \to & 2 \mid ZS \mid UB \\
D & \to & MU \mid ZU \\
Z & \to & 0 \\
U & \to & 1 \\
K & \to & 2 \\
M & \to & ZD
\end{array}
$$

$S$ é o símbolo inicial.




Não há variáveis que produzam $UU$ nem $UZ$.

$\therefore\ X \not\Rightarrow^\star 111$ e $X \not\Rightarrow^\star 110$, para todo $X \in V$.

Logo, $S \not\Rightarrow^\star 1110$. Portanto, $1110 \notin \mathcal{L}(\mathcal{G})$.

# Ideia do Algoritmo de Cocke-Younger-Kasami (CYK)

Aplicação do **Algoritmo CYK** à palavra $x_1x_2x_3x_4 = 1110$:

$$
\begin{array}{lcl}
S & \to & ZS \mid UB \mid MU \mid ZU \mid CK \mid 2 \\
B & \to & UB \mid MU \mid CK \mid ZU \mid 2 \mid ZS \\
C & \to & MU \mid ZU \mid CK \mid 2 \mid ZS \mid UB \\
D & \to & MU \mid ZU \\
Z & \to & 0 \\
U & \to & 1 \\
K & \to & 2 \\
M & \to & ZD
\end{array}
$$

$S$ é o símbolo inicial de $\mathcal{G}$.

| | | | | |
|---|---|---|---|---|
| #4 | $x_1x_2x_3x_4$ $\emptyset$ | | | |
| #3 | $x_1x_2x_3$ $\emptyset$ | $x_2x_3x_4$ $\emptyset$ | | |
| #2 | $x_1x_2$ $\emptyset$ | $x_2x_3$ $\emptyset$ | $x_3x_4$ $\emptyset$ | |
| #1 | $x_1$ $\{U\}$ | $x_2$ $\{U\}$ | $x_3$ $\{U\}$ | $x_4$ $\{Z\}$ |
| | 1 | 1 | 1 | 0 |

**Passos da construção:**

- Linhas 1 a 3 preenchidas como no exemplo anterior.
- Linha 4:

  $x_1x_2x_3x_4$ pode **ser decomposto** como $x_1|x_2x_3x_4$ **ou** $x_1x_2|x_3x_4$ **ou** $x_1x_2x_3|x_4$.

  $x_1|x_2x_3x_4$ é $\{U\}\emptyset = \emptyset$.
  $x_1x_2|x_3x_4$ é $\emptyset\emptyset = \emptyset$.
  $x_1x_2x_3|x_4$ é $\emptyset\{Z\} = \emptyset$.

  Coloca $\emptyset$ em $x_1x_2x_3x_4$.

- **Conclusão:** $x_1x_2x_3x_4$ não pode ser $S$. Como $S \not\Rightarrow^{\star}_{\mathcal{G}} 1110$, concluimos que $1110 \notin \mathcal{L}(\mathcal{G})$.

**Observação:** Quando a **Linha 2** tem apenas $\emptyset$, não é possível construir a árvore para a palavra. Podemos terminar e dar a resposta $x \notin \mathcal{L}(\mathcal{G})$.

# Ideia do Algoritmo de Cocke-Younger-Kasami (CYK)

Aplicação do **Algoritmo CYK** à palavra $x_1x_2x_3x_4x_5 = 22012$:

| | | | | | |
|---|---|---|---|---|---|
| #5 | $x_1x_2x_3x_4x_5$ | | | | |
| #4 | $x_1x_2x_3x_4$ | $x_2x_3x_4x_5$ | | | |
| #3 | $x_1x_2x_3$ $\{S, B, C\}$ | $x_2x_3x_4$ $\{\}$ | $x_3x_4x_5$ $\{S, B, C\}$ | | |
| #2 | $x_1x_2$ $\{S, B, C\}$ | $x_2x_3$ $\{\}$ | $x_3x_4$ $\{S, B, C, D\}$ | $x_4x_5$ $\{S, B, C\}$ | |
| #1 | $x_1$ $\{S, B, C, K\}$ | $x_2$ $\{S, B, C, K\}$ | $x_3$ $\{Z\}$ | $x_4$ $\{U\}$ | $x_5$ $\{S, B, C, K\}$ |
| | 2 | 2 | 0 | 1 | 2 |

$$
\begin{array}{lcl}
S & \to & 2 \mid CK \mid MU \\
S & \to & UB \mid ZS \mid ZU \\
B & \to & 2 \mid CK \mid MU \\
B & \to & UB \mid ZU \mid ZS \\
C & \to & 2 \mid CK \mid MU \\
C & \to & UB \mid ZS \mid ZU \\
D & \to & MU \mid ZU \\
M & \to & ZD \\
Z & \to & 0 \\
U & \to & 1 \\
K & \to & 2
\end{array}
$$

- **Linha 2**

  $x_1x_2$ é $\{S, B, C, K\}\{S, B, C, K\}$. Só $CK$ existe. Coloca $\{S, B, C\}$.

  $x_2x_3$ é $\{S, B, C, K\}\{Z\} = \{SZ, BZ, CZ, KZ\}$. Não existem. Coloca $\emptyset$.

  $x_3x_4$ é $\{Z\}\{U\} = \{ZU\}$. Coloca $\{S, B, C, D\}$.

  $x_4x_5$ é $\{U\}\{S, B, C, K\} = \{US, UB, UC, UK\}$. Coloca $\{S, B, C\}$.

- **Linha 3**

  $x_1x_2x_3$ é $x_1|x_2x_3$ ou $x_1x_2|x_3$. $\{S, B, C, K\}\{\} \cup \{S, B, C\}\{S, B, C, K\}$. Só existe $CK$ e é produzido por $S$, $B$ e $C$. Coloca $\{S, B, C\}$.

  $x_2x_3x_4$ é $x_2|x_3x_4$ ou $x_2x_3|x_4$. $\{S, B, C, K\}\{S, B, C, D\} \cup \{\}\{U\}$. Nenhum ocorre. Coloca $\emptyset$.

  $x_3x_4x_5$ é $x_3|x_4x_5$ ou $x_3x_4|x_5$. $\{Z\}\{S, B, C\} \cup \{S, B, C, D\}\{S, B, C, K\}$. Só $ZS$ e $CK$ existem. Coloca $\{S, B, C\}$.

# Ideia do Algoritmo de Cocke-Younger-Kasami (CYK)

Aplicação do **Algoritmo CYK** à palavra $x_1x_2x_3x_4x_5 = 22012$ (cont.):

| | | | | | |
|---|---|---|---|---|---|
| #5 | $x_1x_2x_3x_4x_5$ $\{\}$ | | | | |
| #4 | $x_1x_2x_3x_4$ $\{\}$ | $x_2x_3x_4x_5$ $\{\}$ | | | |
| #3 | $x_1x_2x_3$ $\{S, B, C\}$ | $x_2x_3x_4$ $\{\}$ | $x_3x_4x_5$ $\{S, B, C\}$ | | |
| #2 | $x_1x_2$ $\{S, B, C\}$ | $x_2x_3$ $\{\}$ | $x_3x_4$ $\{S, B, C, D\}$ | $x_4x_5$ $\{S, B, C\}$ | |
| #1 | $x_1$ $\{S, B, C, K\}$ | $x_2$ $\{S, B, C, K\}$ | $x_3$ $\{Z\}$ | $x_4$ $\{U\}$ | $x_5$ $\{S, B, C, K\}$ |
| | 2 | 2 | 0 | 1 | 2 |

$$
\begin{array}{lcl}
S & \rightarrow & 2 \mid CK \mid MU \\
S & \rightarrow & UB \mid ZS \mid ZU \\
B & \rightarrow & 2 \mid CK \mid MU \\
B & \rightarrow & UB \mid ZU \mid ZS \\
C & \rightarrow & 2 \mid CK \mid MU \\
C & \rightarrow & UB \mid ZS \mid ZU \\
D & \rightarrow & MU \mid ZU \\
M & \rightarrow & ZD \\
Z & \rightarrow & 0 \\
U & \rightarrow & 1 \\
K & \rightarrow & 2
\end{array}
$$

- **Linha 4**

  $x_1x_2x_3x_4$ é $x_1 | x_2x_3x_4$ ou $x_1x_2 | x_3x_4$ ou $x_1x_2x_3 | x_4$.
  $\{S, B, C, K\}\{\} \cup \{S, B, C\}\{S, B, C, D\} \cup \{S, B, C\} \cup \{U\}$. Não existem.

  $x_2x_3x_4x_5$ é $x_2 | x_3x_4x_5$ ou $x_2x_3 | x_4x_5$ ou $x_2x_3x_4 | x_5$.
  $\{S, B, C, K\}\{S, B, C\} \cup \{\}\{S, B, C\} \cup \{\}\{S, B, C, K\}$. Não existem.

- **Linha 5**

  $x_2x_3x_4x_5x_5$ é $x_1 | x_2x_3x_4x_5$ ou $x_1x_2 | x_3x_4x_5$ ou $x_1x_2x_3 | x_4x_5$ ou $x_1x_2x_3x_4 | x_5$.
  $\{S, B, C, K\}\{\} \cup \{S, B, C\}\{S, B, C\} \cup \{S, B, C\}\{S, B, C\} \cup \{\}\{S, B, C, K\} = \{S, B, C\}\{S, B, C\} = \{SS, SB, SC, BS, BB, BC, CS, CB, CC\}$. Não ocorrem.

  Portanto, $22012 \notin \mathcal{L}(\mathcal{G})$.

**Observação:** A **Linha 4** tem apenas $\emptyset$. Mas, se alguma das formas indicadas pudesse ser produzida, a Linha 5 não teria $\emptyset$.

# Algoritmo de Cocke-Younger-Kasami (CYK)

Dada $G = (V, \Sigma, P, S)$ e $x = x_1 x_2 \ldots x_n$, seja $N[i, i+t]$ o conjunto de variáveis que geram $x_i \ldots x_{i+t}$, i.e., $N[i, i+t] = \{A \mid A \in V,\ A \Rightarrow^{\star}_{\mathcal{G}} x_i \ldots x_{i+t}\}$.

AlgoritmoCYK($x$)

Para $i \leftarrow 1$ até $n$ fazer $N[i,i] := \{A \mid A \in V,\ A \rightarrow x_i\}$;
Para todo $(i,j)$, com $i \neq j$ e $1 \leq i \leq n$ e $1 \leq j \leq n$ fazer
    $N[i,j] := \emptyset$;
Para $t \leftarrow 1$ até $n-1$ fazer
    Para $i \leftarrow 1$ até $n-t$ fazer
        Para $k \leftarrow i$ até $i+t-1$ fazer
            Para todo $BC \in N[i,k]N[k+1, i+t]$ fazer
                Para todo $A$ tal que $(A \rightarrow BC) \in P$ fazer
                    $N[i, i+t] := N[i, i+t] \cup \{A\}$
Se $S \in N[1,n]$ então retorna "Sim"; senão retorna "Não";

**Observação:** Para $G = (V, \Sigma, P, S)$ fixa, a complexidade temporal do Algoritmo CYK é $O(|x|^3)$, i.e., é cúbica no comprimento da palavra que se quer analisar.

# Algoritmo de Cocke-Younger-Kasami (CYK)

Dada $G = (V, \Sigma, P, S)$ e $x = x_1 x_2 \ldots x_n$, seja $N[i, i+t]$ o conjunto de variáveis que geram $x_i \ldots x_{i+t}$, i.e., $N[i, i+t] = \{A \mid A \in V,\ A \Rightarrow^{\star}_{\mathcal{G}} x_i \ldots x_{i+t}\}$.

$\text{ALGORITMOCYK}(x)$

Para $i \leftarrow 1$ até $n$ fazer $N[i,i] := \{A \mid A \in V,\ A \rightarrow x_i\}$;
Para todo $(i,j)$, com $i \neq j$ e $1 \leq i \leq n$ e $1 \leq j \leq n$ fazer
    $N[i,j] := \emptyset$;
Para $t \leftarrow 1$ até $n-1$ fazer
    Para $i \leftarrow 1$ até $n-t$ fazer
        Para $k \leftarrow i$ até $i+t-1$ fazer
            Para todo $BC \in N[i,k]N[k+1, i+t]$ fazer
                Para todo $A$ tal que $(A \rightarrow BC) \in P$ fazer
                    $N[i, i+t] := N[i, i+t] \cup \{A\}$
Se $S \in N[1, n]$ então retorna "Sim"; senão retorna "Não";

**Observação:** Para $G = (V, \Sigma, P, S)$ fixa, a complexidade temporal do Algoritmo CYK é $O(|x|^3)$, i.e., é cúbica no comprimento da palavra que se quer analisar.

# Algoritmo de Cocke-Younger-Kasami (CYK)

Dada $G = (V, \Sigma, P, S)$ e $x = x_1 x_2 \ldots x_n$, seja $N[i, i+t]$ o conjunto de variáveis que geram $x_i \ldots x_{i+t}$, i.e., $N[i, i+t] = \{A \mid A \in V,\ A \Rightarrow^\star_{\mathcal{G}} x_i \ldots x_{i+t}\}$.

$\textsc{AlgoritmoCYK}(x)$

Para $i \leftarrow 1$ até $n$ fazer $N[i, i] := \{A \mid A \in V,\ A \rightarrow x_i\}$;
Para todo $(i, j)$, com $i \neq j$ e $1 \leq i \leq n$ e $1 \leq j \leq n$ fazer
    $N[i, j] := \emptyset$;
Para $t \leftarrow 1$ até $n - 1$ fazer
    Para $i \leftarrow 1$ até $n - t$ fazer
        Para $k \leftarrow i$ até $i + t - 1$ fazer
            Para todo $BC \in N[i, k]N[k+1, i+t]$ fazer
                Para todo $A$ tal que $(A \rightarrow BC) \in P$ fazer
                    $N[i, i+t] := N[i, i+t] \cup \{A\}$
Se $S \in N[1, n]$ então retorna "Sim"; senão retorna "Não";

**Observação:** Para $G = (V, \Sigma, P, S)$ fixa, a complexidade temporal do Algoritmo CYK é $O(|x|^3)$, i.e., é cúbica no comprimento da palavra que se quer analisar.

# Algoritmo CYK - utilização de tabela

É usual usar uma matriz, com $N[k, k+t]$ na coluna $k$ e linha $\#t{+}1$.

| | | | | | | |
|---|---|---|---|---|---|---|
| $\#n$ | $N[1,n]$ | | | | | |
| $\#n{-}1$ | $N[1,n{-}1]$ | $N[2,n]$ | | | | |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | | | |
| $\#3$ | $N[1,3]$ | $N[2,4]$ | $\cdots$ | $N[n{-}2,n]$ | | |
| $\#2$ | $N[1,2]$ | $N[2,3]$ | $\cdots$ | $N[n{-}2,n{-}1]$ | $N[n{-}1,n]$ | |
| $\#1$ | $N[1,1]$ | $N[2,2]$ | $\cdots$ | $N[n{-}2,n{-}2]$ | $N[n{-}1,n{-}1]$ | $N[n,n]$ |
| | $x_1$ | $x_2$ | $\cdots$ | $x_{n-2}$ | $x_{n-1}$ | $x_n$ |

A entrada $N[k, k+t]$ da tabela apresenta o conjunto das categorias possíveis para a subpalavra $x_k \cdots x_{k+t}$ de $x$. Portanto, carateriza as categorias das subpalavras indicadas na matrix seguinte, como vimos anteriormente:

| | | | | | | |
|---|---|---|---|---|---|---|
| $\#n$ | $x_1 \cdots x_n$ | | | | | |
| $\#n{-}1$ | $x_1 \cdots x_{n-1}$ | $x_2 \cdots x_n$ | | | | |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | | | |
| $\#3$ | $x_1x_2x_3$ | $x_2x_3x_4$ | $\cdots$ | $x_{n-2}x_{n-1}x_n$ | | |
| $\#2$ | $x_1x_2$ | $x_2x_3$ | $\cdots$ | $x_{n-2}x_{n-1}$ | $x_{n-1}x_n$ | |
| $\#1$ | $x_1$ | $x_2$ | $\cdots$ | $x_{n-2}$ | $x_{n-1}$ | $x_n$ |

# Algoritmo CYK - utilização de tabela

É usual usar uma matriz, com $N[k, k+t]$ na coluna $k$ e linha $\#t{+}1$.

| | | | | | | |
|---|---|---|---|---|---|---|
| $\#n$ | $N[1,n]$ | | | | | |
| $\#n{-}1$ | $N[1,n{-}1]$ | $N[2,n]$ | | | | |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | | | |
| $\#3$ | $N[1,3]$ | $N[2,4]$ | $\cdots$ | $N[n{-}2,n]$ | | |
| $\#2$ | $N[1,2]$ | $N[2,3]$ | $\cdots$ | $N[n{-}2,n{-}1]$ | $N[n{-}1,n]$ | |
| $\#1$ | $N[1,1]$ | $N[2,2]$ | $\cdots$ | $N[n{-}2,n{-}2]$ | $N[n{-}1,n{-}1]$ | $N[n,n]$ |
| | $x_1$ | $x_2$ | $\cdots$ | $x_{n-2}$ | $x_{n-1}$ | $x_n$ |

A entrada $N[k, k+t]$ da tabela apresenta o conjunto das categorias possíveis para a subpalavra $x_k \cdots x_{k+t}$ de $x$. Portanto, carateriza as categorias das subpalavras indicadas na matrix seguinte, como vimos anteriormente:

| | | | | | | |
|---|---|---|---|---|---|---|
| $\#n$ | $x_1 \cdots x_n$ | | | | | |
| $\#n{-}1$ | $x_1 \cdots x_{n-1}$ | $x_2 \cdots x_n$ | | | | |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | | | |
| $\#3$ | $x_1x_2x_3$ | $x_2x_3x_4$ | $\cdots$ | $x_{n-2}x_{n-1}x_n$ | | |
| $\#2$ | $x_1x_2$ | $x_2x_3$ | $\cdots$ | $x_{n-2}x_{n-1}$ | $x_{n-1}x_n$ | |
| $\#1$ | $x_1$ | $x_2$ | $\cdots$ | $x_{n-2}$ | $x_{n-1}$ | $x_n$ |

# Aplicação do Algoritmo CYK

**Exemplo:** Para GIC $G$, com $V = \{E, T, F, E_1, E_2, T_1, T_2, T_3, M, S, X, Q, A, B\}$, símbolo inicial $E$, e seguintes produções:

$$\begin{array}{lll} E & \to & TE_1 \mid TE_2 \mid FT_1 \mid FT_2 \mid AT_3 \mid \mathtt{n} \\ T & \to & \mathtt{n} \mid FT_1 \mid FT_2 \mid AT_3 \\ F & \to & AT_3 \mid \mathtt{n} \\ E_1 & \to & ME \\ E_2 & \to & SE \\ T_1 & \to & XT \\ T_2 & \to & QT \end{array} \qquad \begin{array}{lll} M & \to & \texttt{+} \\ S & \to & \texttt{-} \\ X & \to & \texttt{*} \\ Q & \to & \texttt{/} \\ A & \to & \texttt{(} \\ B & \to & \texttt{)} \\ T_3 & \to & EB \end{array}$$

Tem-se $(\mathtt{n} + \mathtt{n}) * \mathtt{n} \in \mathcal{L}(\mathcal{G})$ porque o símbolo inicial $E$ ocorre no topo da tabela:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| #7 | $\{T, E\}$ | | | | | | |
| #6 | $\emptyset$ | $\emptyset$ | | | | | |
| #5 | $\{F, T, E\}$ | $\emptyset$ | $\emptyset$ | | | | |
| #4 | $\emptyset$ | $\{T_3\}$ | $\emptyset$ | $\emptyset$ | | | |
| #3 | $\emptyset$ | $\{E\}$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | | |
| #2 | $\emptyset$ | $\emptyset$ | $\{E_1\}$ | $\{T_3\}$ | $\emptyset$ | $\{T_1\}$ | |
| #1 | $\{A\}$ | $\{E, T, F\}$ | $\{M\}$ | $\{E, T, F\}$ | $\{B\}$ | $\{X\}$ | $\{E, T, F\}$ |
| | ( | n | $+$ | n | ) | $*$ | n |

# Simplificação de gramáticas

Recordar que duas gramáticas $\mathcal{G}$ e $\mathcal{G}'$ são equivalentes se e só se $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$.

Na redução de $\mathcal{G}$ à FN Chomsky ou à FN Greibach, pode ser preciso simplificar $\mathcal{G}$ para garantir que não contém símbolos desnecessários nem variáveis que gerem $\varepsilon$.

### Simplificação de gramáticas

Qualquer GIC $\mathcal{G} = (V, \Sigma, P, S)$, com $\mathcal{L}(\mathcal{G}) \neq \emptyset$, pode ser transformada numa GIC equivalente $\mathcal{G}' = (V', \Sigma, P', S)$ tal que em $\mathcal{G}'$:

- **não há símbolos trivialmente desnecessários**
  - cada símbolo terminal ocorre em alguma palavra em $x \in \mathcal{L}(\mathcal{G}')$
  - cada variável ocorre em alguma derivação de algum $x \in \mathcal{L}(\mathcal{G}')$
- **nenhuma variável produz** $\varepsilon$, com possível exceção de $S$, se $\varepsilon \in \mathcal{L}(\mathcal{G}')$
- **não há regras unitárias**, isto é, regras $A \to B$, com $B$ variável.

# Eliminar variáveis que não geram sequências de terminais

Não foi dado na Aula 22

Seja $\mathcal{G} = (V, \Sigma, P, S)$ uma GIC tal que $\mathcal{L}(\mathcal{G}) \neq \emptyset$. Existe um algoritmo que obtém uma GIC $\mathcal{G}' = (V', \Sigma, P', S)$ equivalente a $\mathcal{G}$, onde cada variável $A \in V' \subseteq V$ gera alguma sequência em $\Sigma^\star$, ou seja, em $\mathcal{G}'$, para cada $A \in V'$, existe $w \in \Sigma^\star$ tal que $A \Rightarrow^\star_{\mathcal{G}'} w$.

**Ideia da prova:**
Começar por definir $V'$ como o conjunto de variáveis que trivialmente produzem sequências de terminais (por terem regras que só têm sequências de terminais no lado direito). Depois completar $V'$, acrescentando variáveis que têm produções que, no lado direito, incluem apenas variáveis que já estão em $V'$, e sucessivamente.

**Algoritmo 1:**

$W := \emptyset$
$V' := \{A \mid (A \to \alpha) \in P, \text{ para algum } \alpha \in \Sigma^\star\}$
Enquanto $(W \neq V')$ fazer
  $W := V'$
  $V' := W \cup \{A \mid (A \to \alpha) \in P, \text{ para algum } \alpha \in (\Sigma \cup W)^\star\}$.
$P' := \{A \to \alpha \mid A \in V', (A \to \alpha) \in P, \ \alpha \in (\Sigma \cup V')^\star\}$.

**Observação (corolário):** $\mathcal{L}(\mathcal{G}') = \emptyset$ sse $S \notin V'$. Este algoritmo permite decidir se uma gramática $\mathcal{G}$ gera a linguagem vazia.

# Eliminar variáveis que não geram sequências de terminais

Não foi dado na Aula 22

Seja $\mathcal{G} = (V, \Sigma, P, S)$ uma GIC tal que $\mathcal{L}(\mathcal{G}) \neq \emptyset$. Existe um algoritmo que obtém uma GIC $\mathcal{G}' = (V', \Sigma, P', S)$ equivalente a $\mathcal{G}$, onde cada variável $A \in V' \subseteq V$ gera alguma sequência em $\Sigma^\star$, ou seja, em $\mathcal{G}'$, para cada $A \in V'$, existe $w \in \Sigma^\star$ tal que $A \Rightarrow^\star_{\mathcal{G}'} w$.

**Ideia da prova:**
Começar por definir $V'$ como o conjunto de variáveis que trivialmente produzem sequências de terminais (por terem regras que só têm sequências de terminais no lado direito). Depois completar $V'$, acrescentando variáveis que têm produções que, no lado direito, incluem apenas variáveis que já estão em $V'$, e sucessivamente.

**Algoritmo 1:**

$W := \emptyset$
$V' := \{A \mid (A \to \alpha) \in P, \text{ para algum } \alpha \in \Sigma^\star\}$
Enquanto $(W \neq V')$ fazer
    $W := V'$
    $V' := W \cup \{A \mid (A \to \alpha) \in P, \text{ para algum } \alpha \in (\Sigma \cup W)^\star\}$.
$P' := \{A \to \alpha \mid A \in V', (A \to \alpha) \in P, \ \alpha \in (\Sigma \cup V')^\star\}$.

**Observação (corolário):** $\mathcal{L}(\mathcal{G}') = \emptyset$ sse $S \notin V'$. Este algoritmo permite decidir se uma gramática $\mathcal{G}$ gera a linguagem vazia.

# Eliminar variáveis que não geram sequências de terminais

Não foi dado na Aula 22

Seja $\mathcal{G} = (V, \Sigma, P, S)$ uma GIC tal que $\mathcal{L}(\mathcal{G}) \neq \emptyset$. Existe um algoritmo que obtém uma GIC $\mathcal{G}' = (V', \Sigma, P', S)$ equivalente a $\mathcal{G}$, onde cada variável $A \in V' \subseteq V$ gera alguma sequência em $\Sigma^\star$, ou seja, em $\mathcal{G}'$, para cada $A \in V'$, existe $w \in \Sigma^\star$ tal que $A \Rightarrow^\star_{\mathcal{G}'} w$.

**Ideia da prova:**

Começar por definir $V'$ como o conjunto de variáveis que trivialmente produzem sequências de terminais (por terem regras que só têm sequências de terminais no lado direito). Depois completar $V'$, acrescentando variáveis que têm produções que, no lado direito, incluem apenas variáveis que já estão em $V'$, e sucessivamente.

**Algoritmo 1:**

$W := \emptyset$
$V' := \{A \mid (A \to \alpha) \in P, \text{ para algum } \alpha \in \Sigma^\star\}$
Enquanto $(W \neq V')$ fazer
    $W := V'$
    $V' := W \cup \{A \mid (A \to \alpha) \in P, \text{ para algum } \alpha \in (\Sigma \cup W)^\star\}$.
$P' := \{A \to \alpha \mid A \in V', (A \to \alpha) \in P, \ \alpha \in (\Sigma \cup V')^\star\}$.

**Observação (corolário):** $\mathcal{L}(\mathcal{G}') = \emptyset$ sse $S \notin V'$. Este algoritmo permite decidir se uma gramática $\mathcal{G}$ gera a linguagem vazia.

# Eliminar variáveis que não ocorrem em derivações a partir do símbolo inicial

Não foi dado na Aula 22

Recordar que se $\mathcal{L}(\mathcal{G}) \neq \emptyset$, um **símbolo de $V \cup \Sigma$** é **desnecessário** se não ocorre em nenhuma derivação de palavras de $\mathcal{L}(\mathcal{G})$. Uma **variável é desnecessária** se:

- não produzir sequências de terminais, isto é, se a linguagem gerada a partir dessa variável for vazia, ou
- se não ocorrer na derivações a partir de $S$.

A gramática que se resulta do **Algoritmo 1** pode conter variáveis ou terminais desnecessários, embora todas as variáveis produzam linguagens não vazias.

Podemos continuar a simplificar a gramática para remover símbolos que não ocorrem em derivações a partir de $S$.

# Eliminar variáveis que não ocorrem em derivações a partir do símbolo inicial

Não foi dado na Aula 22

Recordar que se $\mathcal{L}(\mathcal{G}) \neq \emptyset$, um **símbolo de $V \cup \Sigma$** é **desnecessário** se não ocorre em nenhuma derivação de palavras de $\mathcal{L}(\mathcal{G})$. Uma **variável é desnecessária** se:

- não produzir sequências de terminais, isto é, se a linguagem gerada a partir dessa variável for vazia, ou
- se não ocorrer na derivações a partir de $S$.

A gramática que se resulta do **Algoritmo 1** pode conter variáveis ou terminais desnecessários, embora todas as variáveis produzam linguagens não vazias.

Podemos continuar a simplificar a gramática para remover símbolos que não ocorrem em derivações a partir de $S$.

# Eliminar variáveis que não ocorrem em derivações a partir do símbolo inicial

Não foi dado na Aula 22

Recordar que se $\mathcal{L}(\mathcal{G}) \neq \emptyset$, um **símbolo de $V \cup \Sigma$** é **desnecessário** se não ocorre em nenhuma derivação de palavras de $\mathcal{L}(\mathcal{G})$. Uma **variável é desnecessária** se:

- não produzir sequências de terminais, isto é, se a linguagem gerada a partir dessa variável for vazia, ou
- se não ocorrer na derivações a partir de $S$.

A gramática que se resulta do **Algoritmo 1** pode conter variáveis ou terminais desnecessários, embora todas as variáveis produzam linguagens não vazias.

Podemos continuar a simplificar a gramática para remover símbolos que não ocorrem em derivações a partir de $S$.

# Eliminar variáveis que não ocorrem em derivações a partir do símbolo inicial

Não foi dado na Aula 22

Seja $\mathcal{G} = (V, \Sigma, P, S)$ uma GIC. Existe um algoritmo que obtém uma GIC $\mathcal{G}' = (V', \Sigma', P', S)$ equivalente a $\mathcal{G}$, em que cada variável $A \in V' \subseteq V$ ocorre em alguma derivação a partir de $S$, i.e., $S \Rightarrow^{\star}_{\mathcal{G}'} w_1 A w_2$, com $w_1, w_2 \in (V' \cup \Sigma')^{\star}$.

**Algoritmo 2:**

$V' := \{S\}$;   $P' := \Sigma' := \emptyset$
Enquanto $Q \neq \emptyset$ fazer
    $X :=$ Retirar de $Q$ uma variável
    $Q' := \{$variáveis que ocorrem nas regras para $X$ e não estão em $V'\}$
    $V' := V' \cup Q'$
    $P' := P' \cup \{$regras para $X\}$
    $\Sigma' := \Sigma' \cup \{$terminais que ocorrem nas regras para $X\}$
    $Q := Q \cup Q'$

**Proposição:** Seja $\mathcal{G}$ uma GIC tal que $\mathcal{L}(\mathcal{G}) \neq \emptyset$. Se se aplicar o Algoritmo 1 e a seguir o Algoritmo 2, obtém-se uma GIC $\mathcal{G}'$ equivalente a $\mathcal{G}$ sem símbolo desnecessários.

# Nenhuma variável distinta do símbolo inicial gerará $\varepsilon$

Transformação referida na Aula 22

Seja $\mathcal{G} = (V, \Sigma, P, S)$ uma GIC. Existe um algoritmo que determina uma GIC $\mathcal{G}_1 = (V_1, \Sigma, P_1, S)$, que gera $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$, sem símbolos desnecessários e sem variáveis que produzem $\varepsilon$, ou seja, tem-se $A \not\Rightarrow^{\star}_{\mathcal{G}_1} \varepsilon$, para todo $A \in V_1$.

Ideia da prova:

**Determinar todas as variáveis $A$ em $V$ que geram $\varepsilon$**, isto é, tais que $A \Rightarrow^{\star}_{\mathcal{G}} \varepsilon$.

- para cada $A \in V$, marcar $A$ como *"variável que gera $\varepsilon$"* se $(A \to \varepsilon) \in P$;
- Marcar as restantes variáveis $A$ tais que $(A \to \alpha) \in P$ e $\alpha$ é uma sequência de variáveis já marcadas; Continuar até não ser possível marcar mais variáveis.

**Determinar o conjunto de regras $P'$:** para cada regra $(A \to X_1X_2\cdots X_n) \in P$, com $X_i \in V \cup \Sigma$, colocar em $P'$ todas as produções $A \to \alpha_1\alpha_2\cdots\alpha_n$ obtidas dessa regra e que satisfazem as condições seguintes:

- se $X_i$ for um terminal ou uma variável não marcada, então $\alpha_i = X_i$;
- se $X_i \in V$ estiver marcada, então $\alpha_i$ pode ser $\varepsilon$ ou $X_i$.
- os $\alpha_i$'s não podem ser todos $\varepsilon$.

A gramática $\mathcal{G}' = (V, \Sigma, P', S)$ gera $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$. Para eliminar símbolos desnecessários aplicar os Algoritmos 1 e 2.

# Nenhuma variável distinta do símbolo inicial gerará $\varepsilon$

Transformação referida na Aula 22

Seja $\mathcal{G} = (V, \Sigma, P, S)$ uma GIC. Existe um algoritmo que determina uma GIC $\mathcal{G}_1 = (V_1, \Sigma, P_1, S)$, que gera $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$, sem símbolos desnecessários e sem variáveis que produzem $\varepsilon$, ou seja, tem-se $A \not\Rightarrow^\star_{\mathcal{G}_1} \varepsilon$, para todo $A \in V_1$.

Ideia da prova:

**Determinar todas as variáveis $A$ em $V$ que geram $\varepsilon$**, isto é, tais que $A \Rightarrow^\star_{\mathcal{G}} \varepsilon$.

- para cada $A \in V$, marcar $A$ como *"variável que gera $\varepsilon$"* se $(A \to \varepsilon) \in P$;
- Marcar as restantes variáveis $A$ tais que $(A \to \alpha) \in P$ e $\alpha$ é uma sequência de variáveis já marcadas; Continuar até não ser possível marcar mais variáveis.

**Determinar o conjunto de regras $P'$:** para cada regra $(A \to X_1X_2\cdots X_n) \in P$, com $X_i \in V \cup \Sigma$, colocar em $P'$ todas as produções $A \to \alpha_1\alpha_2\cdots\alpha_n$ obtidas dessa regra e que satisfazem as condições seguintes:

- se $X_i$ for um terminal ou uma variável não marcada, então $\alpha_i = X_i$;
- se $X_i \in V$ estiver marcada, então $\alpha_i$ pode ser $\varepsilon$ ou $X_i$.
- os $\alpha_i$'s não podem ser todos $\varepsilon$.

A gramática $\mathcal{G}' = (V, \Sigma, P', S)$ gera $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$. Para eliminar símbolos desnecessários aplicar os Algoritmos 1 e 2.

# Eliminar regras unitárias $A \to B$, com $B$ variável

Qualquer LIC $L$, tal que $L \neq \emptyset$ e $\varepsilon \notin L$, pode ser gerada por uma GIC que não tem produções unitárias, nem símbolos que não ocorrem na derivação de palavras de $L$, nem variáveis que gerem $\varepsilon$.

**Ideia da prova**

- Existe uma GIC $\mathcal{G} = (V, \Sigma, P, S)$ que gera $L$, sem símbolos desnecessários e variáveis que gerem $\varepsilon$. Se $\mathcal{G}$ não tiver produções unitárias, nada resta fazer.
- Se não, vamos transformá-la numa GIC $\mathcal{G}'$ equivalente mas sem produções unitárias, assim:
  - enquanto houver regras unitárias, tomamos uma qualquer $A \to B$ e substituimo-la pelas regras $A \to \beta$, tais que $B \to \beta$ e $\beta \neq A$.
  - Pode acontecer que nesta substituição se criem novas regras unitárias, mas o processo terminará.
- Finalmente, simplificamos a gramática resultante, se necessário, usando os Algoritmos 1 e 2.

# Redução à forma normal de Chomsky

**Existência da Forma normal de Chomsky (FNC):** Qualquer LIC $L$ que não tenha $\varepsilon$, pode ser gerada por uma GIC na forma normal de Chomsky.

**Prova:**

- Se $L = \emptyset$, então $\mathcal{G} = (\{S\}, \Sigma, \{\}, S)$ gera $\mathcal{L}$ e está em FNC.
- Seja $L \neq \emptyset$ e tal que $\varepsilon \notin L$. Seja $\mathcal{G} = (V, \Sigma, P, S)$ uma GIC sem produções $A \to \varepsilon$ nem $A \to B$, com $A, B \in V$. Esta GIC pode ser obtida a partir de qualquer GIC que gere $L$, como vimos.
- Para as regras que não são do tipo $A \to a$, com $a \in \Sigma$, garantimos que não ocorrem terminais: introduzimos variáveis $C_a$ novas, com $C_a \to a$, com $a \in \Sigma$, e substituimos.
- Depois, substituimos regras $A \to X_1 X_2 \cdots X_n$, com $n \geq 3$, se existirem, por

  $$A \to X_1 D_1 \qquad D_1 \to X_2 D_2 \quad \ldots \qquad D_{n-2} \to X_{n-1} X_n$$

  introduzindo variáveis novas $D_1, D_2 \ldots, D_{n-1}$ para tal.

A GIC que se obtém gera $L$ e está na FNC.

# Redução à forma normal de Chomsky

**Existência da Forma normal de Chomsky (FNC):** Qualquer LIC $L$ que não tenha $\varepsilon$, pode ser gerada por uma GIC na forma normal de Chomsky.

**Prova:**

- Se $L = \emptyset$, então $\mathcal{G} = (\{S\}, \Sigma, \{\}, S)$ gera $\mathcal{L}$ e está em FNC.
- Seja $L \neq \emptyset$ e tal que $\varepsilon \notin L$. Seja $\mathcal{G} = (V, \Sigma, P, S)$ uma GIC sem produções $A \to \varepsilon$ nem $A \to B$, com $A, B \in V$. Esta GIC pode ser obtida a partir de qualquer GIC que gere $L$, como vimos.
- Para as regras que não são do tipo $A \to a$, com $a \in \Sigma$, garantimos que não ocorrem terminais: introduzimos variáveis $C_a$ novas, com $C_a \to a$, com $a \in \Sigma$, e substituimos.
- Depois, substituimos regras $A \to X_1 X_2 \cdots X_n$, com $n \geq 3$, se existirem, por

  $$A \to X_1 D_1 \qquad D_1 \to X_2 D_2 \quad \ldots \qquad D_{n-2} \to X_{n-1} X_n$$

  introduzindo variáveis novas $D_1, D_2 \ldots, D_{n-1}$ para tal.

A GIC que se obtém gera $L$ e está na FNC.

# Gramáticas regulares

- Uma GIC $G = (V, \Sigma, P, S)$ é **linear à direita** sse qualquer regra é da forma $A \to w$ ou $A \to wB$, com $w \in \Sigma^\star$ e $B \in V$.

- Uma GIC $G = (V, \Sigma, P, S)$ é **linear à esquerda** sse qualquer regra é da forma $A \to w$ ou $A \to Bw$, com $w \in \Sigma^\star$ e $B \in V$;

Qualquer GIC linear à direita (ou linear à esquerda) gera uma linguagem regular e qualquer linguagem regular pode ser gerada por uma GIC linear à direita (e por uma GIC linear à esquerda).

As GICs lineares à direita e as GICs lineares à esquerda dizem-se **gramáticas regulares**.

# Conversão de AFs para GICs lineares à direita

- **Dado um AFD** $A = (S, \Sigma, \delta, s_0, F)$, a linguagem $\mathcal{L}(A)$ é gerada pela **gramática linear à direita** $G = (\{V_s \mid s \in S\}, \Sigma, P, V_{s_0})$, em que $P$ tem a regra $V_s \to aV_{s'}$ sse $\delta(s, a) = s'$ e ainda uma regra $V_f \to \varepsilon$, **por cada estado final** $f \in F$.
- O mesmo algoritmo pode ser trivialmente adaptado se $A$ for um AFND ou AFND-$\varepsilon$.
- Um percurso no autómato do estado $s_0$ até um estado final, em que se consumiu a palavra $x$, corresponde a uma derivação da palavra $x$ em $G$ a partir de $V_{s_0}$, e vice-versa.
- A gramática obtida por aplicação do método de conversão a um **AFD é não ambígua**. Portanto, qualquer linguagem regular é não ambígua.

# Conversão de GICs lineares à direita para AFs

Não foi dado na Aula 22

- **Dada uma GIC $G$ linear à direita**, podemos **obter um AFND-$\varepsilon$** que reconhece $\mathcal{L}(G)$ pelo algoritmo seguinte.
- Começamos por substituir todas as regras da forma $X \to w$ por $X \to wX_f$, em que $X_f$ é uma variável nova (e a mesma para todas as regras), e será a única com produção $X_f \to \varepsilon$ (o AFND-$\varepsilon$ terá um único estado final).
- Depois, cada regra da forma $X \to wY$, com $|w| \geq 2$, é substituída por $|w|$ regras, $X \to a_1 Y_1$, $Y_1 \to a_2 Y_2, \ldots, Y_{k-1} \to a_k Y$, sendo $w = a_1 a_2 \ldots a_k$, e $Y_1, \ldots Y_{k-1}$ variáveis novas (criadas para cada regra).
- O **conjunto de estados do AFND-$\varepsilon$** corresponde ao conjunto de variáveis da nova gramática e $X_f$ ao **estado final**.
- Cada regra $X \to \alpha Y$, com $\alpha \in \Sigma \cup \{\varepsilon\}$, define **uma transição** do estado $X$ para o estado $Y$ por $\alpha$.