Tema 4: Gramáticas libres del contexto

Autómatas y Lenguajes Formales
Segundo de Grado en Informática

Dpto. de Ingeniería de la Información y las Comunicaciones

UNIVERSIDAD DE MURCIA

Definición formal de GLC

Las **gramáticas** (*Noam Chomsky*) son formalismos para describir lenguajes. Un tipo son las las *gramáticas libres del contexto*, más potente que las expresiones regulares.

Una **gramática libre del contexto (GLC)** viene dada por una cuádrupla $\boxed{G = (V_N, V_T, S, P)}$ donde:

- V_N es el alfabeto de **variables** o "no-terminales".
- V_T es el alfabeto de **símbolos terminales**.
- S es el **símbolo inicial** y $S \in V_N$.
- P es un conjunto finito de **reglas de producción** de la forma:

$$\boxed{A \to \alpha} \quad \text{donde } A \in V_N,\ \alpha \in (V_N \cup V_T)^*$$

Definición formal de GLC

Las **gramáticas** (*Noam Chomsky*) son formalismos para describir lenguajes. Un tipo son las las *gramáticas libres del contexto*, más potente que las expresiones regulares.

Una **gramática libre del contexto (GLC)** viene dada por una cuádrupla $G = (V_N, V_T, S, P)$ donde:

- V_N es el alfabeto de **variables** o "no-terminales".
- V_T es el alfabeto de **símbolos terminales**.
- S es el **símbolo inicial** y $S \in V_N$.
- P es un conjunto finito de **reglas de producción** de la forma:

  $A \to \alpha$ donde $A \in V_N,\ \alpha \in (V_N \cup V_T)^*$

Ejemplo de gramática y representación compacta

Sea la gramática dada por el símbolo inicial S, conjunto de variables $V_N = \{S\}$, conjunto de terminales $V_T = \{a, b\}$ y conjunto de reglas $P = \{S \to aSa,\ S \to bSb,\ S \to a,\ S \to b,\ S \to \lambda\}$

Se representa de forma compacta: $S \to aSa \mid bSb \mid a \mid b \mid \lambda$

Derivación directa: aplicación de una regla de producción

- Una regla del tipo $A \to \gamma$ **es aplicable** a la cadena α si y sólo si en α aparece al menos una ocurrencia de la variable A.
- La **aplicación de la regla** consiste en sustituir en α la variable A por γ.
- **Derivación directa:** decimos que α **deriva directamente** en β y se denota $\boxed{\alpha \Rightarrow \beta}$ si β se puede obtener a partir de α por aplicación de una regla.

Derivación directa: aplicación de una regla de producción

- Una regla del tipo $A \to \gamma$ **es aplicable** a la cadena α si y sólo si en α aparece al menos una ocurrencia de la variable A.
- La **aplicación de la regla** consiste en sustituir en α la variable A por γ.
- **Derivación directa:** decimos que α **deriva directamente** en β y se denota $\boxed{\alpha \Rightarrow \beta}$ si β se puede obtener a partir de α por aplicación de una regla.

Ejemplo de derivación directa

Considerando la gramática $S \to aSa \mid bSb \mid a \mid b \mid \lambda$ y la cadena $\alpha = \mathbf{abSba}$, cualquiera de las reglas para la variable S son aplicables a la cadena $abSba$.

$\rightrightarrows$ Aplicando la regla $\mathbf{S \to aSa}$ se tiene que:

$$ab\,\mathbf{S}\,ba \Rightarrow$$

Derivación directa: aplicación de una regla de producción

- Una regla del tipo $A \to \gamma$ **es aplicable** a la cadena α si y sólo si en α aparece al menos una ocurrencia de la variable A.
- La **aplicación de la regla** consiste en sustituir en α la variable A por γ.
- **Derivación directa:** decimos que α **deriva directamente** en β y se denota $\boxed{\alpha \Rightarrow \beta}$ si β se puede obtener a partir de α por aplicación de una regla.

Ejemplo de derivación directa

Considerando la gramática $S \to aSa \mid bSb \mid a \mid b \mid \lambda$ y la cadena $\alpha =$ **abSba**, cualquiera de las reglas para la variable S son aplicables a la cadena $abSba$.

$\rightrightarrows$ Aplicando la regla $\mathbf{S} \to \mathbf{aSa}$ se tiene que:

$$ab\,\mathbf{S}\,ba \Rightarrow ab\,\mathbf{aSa}\,ba$$

Derivación: aplicación de varias reglas de producción

- Con una gramática **interesa generar cadenas que contengan sólo símbolos terminales a partir del símbolo inicial**, normalmente aplicando varias reglas $\rightrightarrows$ extender el concepto de derivación directa.

Derivación: aplicación de varias reglas de producción

- Con una gramática **interesa generar cadenas que contengan sólo símbolos terminales a partir del símbolo inicial**, normalmente aplicando varias reglas $\rightrightarrows$ extender el concepto de derivación directa.
- **Derivación:** decimos que α **deriva en** β y se denota $\alpha \Rightarrow^* \beta$ si

  1. $\alpha = \beta$, ($\alpha \Rightarrow^* \alpha$ en **cero pasos**), o bien,
  2. **Existe una secuencia de derivaciones directas:** $\alpha \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \cdots \Rightarrow \gamma_n = \beta$ tal que $n \geq 1$ ($\alpha \Rightarrow^* \beta$ en n **pasos**).

Derivación: aplicación de varias reglas de producción

- Con una gramática **interesa generar cadenas que contengan sólo símbolos terminales a partir del símbolo inicial**, normalmente aplicando varias reglas $\rightrightarrows$ extender el concepto de derivación directa.
- **Derivación:** decimos que α **deriva en** β y se denota $\alpha \Rightarrow^* \beta$ si

  1. $\alpha = \beta$, ($\alpha \Rightarrow^* \alpha$ en **cero pasos**), o bien,
  2. **Existe una secuencia de derivaciones directas:** $\alpha \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \cdots \Rightarrow \gamma_n = \beta$ tal que $n \geq 1$ ($\alpha \Rightarrow^* \beta$ en n **pasos**).

Ejemplo de derivación

Dada $S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$, tenemos que $S \Rightarrow^* bbabb$ porque:

S

Derivación: aplicación de varias reglas de producción

- Con una gramática **interesa generar cadenas que contengan sólo símbolos terminales a partir del símbolo inicial**, normalmente aplicando varias reglas $\Rightarrow$ extender el concepto de derivación directa.
- **Derivación:** decimos que α **deriva en** β y se denota $\alpha \Rightarrow^* \beta$ si

  1. $\alpha = \beta$, ($\alpha \Rightarrow^* \alpha$ en **cero pasos**), o bien,
  2. **Existe una secuencia de derivaciones directas:** $\alpha \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \cdots \Rightarrow \gamma_n = \beta$ tal que $n \geq 1$ ($\alpha \Rightarrow^* \beta$ en n **pasos**).

Ejemplo de derivación

Dada $S \to aSa \mid bSb \mid a \mid b \mid \lambda$, tenemos que $S \Rightarrow^* bbabb$ porque:

$$S \overbrace{\Rightarrow}^{S \to bSb} b\boxed{S}b$$

Derivación: aplicación de varias reglas de producción

- Con una gramática **interesa generar cadenas que contengan sólo símbolos terminales a partir del símbolo inicial**, normalmente aplicando varias reglas $\Rightarrow$ extender el concepto de derivación directa.
- **Derivación:** decimos que α **deriva en** β y se denota $\alpha \Rightarrow^* \beta$ si

  1. $\alpha = \beta$, ($\alpha \Rightarrow^* \alpha$ en **cero pasos**), o bien,
  2. **Existe una secuencia de derivaciones directas:** $\alpha \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \cdots \Rightarrow \gamma_n = \beta$ tal que $n \geq 1$ ($\alpha \Rightarrow^* \beta$ en n **pasos**).

Ejemplo de derivación

Dada $S \to aSa \mid bSb \mid a \mid b \mid \lambda$, tenemos que $S \Rightarrow^* bbabb$ porque:

$$S \overbrace{\Rightarrow}^{S \to bSb} b\boxed{S}b \overbrace{\Rightarrow}^{S \to bSb} bb\boxed{S}bb$$

Derivación: aplicación de varias reglas de producción

- Con una gramática **interesa generar cadenas que contengan sólo símbolos terminales a partir del símbolo inicial**, normalmente aplicando varias reglas $\rightrightarrows$ extender el concepto de derivación directa.
- **Derivación:** decimos que α **deriva en** β y se denota $\alpha \Rightarrow^* \beta$ si

  1. $\alpha = \beta$, ($\alpha \Rightarrow^* \alpha$ en **cero pasos**), o bien,
  2. **Existe una secuencia de derivaciones directas:**
     $\alpha \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \cdots \Rightarrow \gamma_n = \beta$ tal que $n \geq 1$
     ($\alpha \Rightarrow^* \beta$ en n **pasos**).

Ejemplo de derivación

Dada $S \to aSa \mid bSb \mid a \mid b \mid \lambda$, tenemos que $S \Rightarrow^* bbabb$ porque:

$S \overset{S\to bSb}{\overbrace{\Rightarrow}} b\boxed{S}b \overset{S\to bSb}{\overbrace{\Rightarrow}} bb\boxed{S}bb \overset{S\to a}{\overbrace{\Rightarrow}} bbabb$ luego $S \Rightarrow^* bbabb$ en 3 pasos

Propiedades de las relación de derivación

- **Extiende la relación de derivación directa** porque si $\alpha \Rightarrow \beta$ se tiene que $\alpha \Rightarrow^* \beta$.
- Es **reflexiva** porque $\alpha \Rightarrow^* \alpha$.
- Es **transitiva** porque si $\alpha \Rightarrow^* \beta$ y $\beta \Rightarrow^* \gamma$ entonces $\alpha \Rightarrow^* \gamma$.
- **Regla de inferencia de derivaciones**:
  Si tenemos la cadena $\alpha A \beta$ y sabemos que $A \Rightarrow^* \gamma$ entonces se puede deducir que $\alpha A \beta \Rightarrow^* \alpha \gamma \beta$.
  $\rightrightarrows$ **generaliza** la aplicación de una regla.

Propiedades de las relación de derivación

- **Extiende la relación de derivación directa** porque si $\alpha \Rightarrow \beta$ se tiene que $\alpha \Rightarrow^* \beta$.
- Es **reflexiva** porque $\alpha \Rightarrow^* \alpha$.
- Es **transitiva** porque si $\alpha \Rightarrow^* \beta$ y $\beta \Rightarrow^* \gamma$ entonces $\alpha \Rightarrow^* \gamma$.
- **Regla de inferencia de derivaciones**:
  Si tenemos la cadena $\alpha A \beta$ y sabemos que $A \Rightarrow^* \gamma$ entonces se puede deducir que $\alpha A \beta \Rightarrow^* \alpha \gamma \beta$.
  $\rightrightarrows$ **generaliza** la aplicación de una regla.

Ejemplo de derivación

Dada la GLC $S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$, podemos deducir que $S \Rightarrow^* a^n\, bbabb\, a^n$ para todo $n > 0$ porque:

Propiedades de las relación de derivación

- **Extiende la relación de derivación directa** porque si $\alpha \Rightarrow \beta$ se tiene que $\alpha \Rightarrow^* \beta$.
- Es **reflexiva** porque $\alpha \Rightarrow^* \alpha$.
- Es **transitiva** porque si $\alpha \Rightarrow^* \beta$ y $\beta \Rightarrow^* \gamma$ entonces $\alpha \Rightarrow^* \gamma$.
- **Regla de inferencia de derivaciones**:
  Si tenemos la cadena $\alpha A \beta$ y sabemos que $A \Rightarrow^* \gamma$ entonces se puede deducir que $\alpha A \beta \Rightarrow^* \alpha\gamma\beta$.
  $\rightrightarrows$ **generaliza** la aplicación de una regla.

Ejemplo de derivación

Dada la GLC $S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$, podemos deducir que
$S \Rightarrow^* a^n\, bbabb\, a^n$ para todo $n > 0$ porque:
1: $S \Rightarrow^* a^n S a^n$ por aplicación de la regla $S \rightarrow aSa$ $n > 0$ veces.

Propiedades de las relación de derivación

- **Extiende la relación de derivación directa** porque si $\alpha \Rightarrow \beta$ se tiene que $\alpha \Rightarrow^* \beta$.
- Es **reflexiva** porque $\alpha \Rightarrow^* \alpha$.
- Es **transitiva** porque si $\alpha \Rightarrow^* \beta$ y $\beta \Rightarrow^* \gamma$ entonces $\alpha \Rightarrow^* \gamma$.
- **Regla de inferencia de derivaciones**:
  Si tenemos la cadena $\alpha A \beta$ y sabemos que $A \Rightarrow^* \gamma$ entonces se puede deducir que $\alpha A \beta \Rightarrow^* \alpha \gamma \beta$.
  $\rightrightarrows$ **generaliza** la aplicación de una regla.

Ejemplo de derivación

Dada la GLC $S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$, podemos deducir que $S \Rightarrow^* a^n \, bbabb \, a^n$ para todo $n > 0$ porque:

1: $S \Rightarrow^* a^n S a^n$ por aplicación de la regla $S \rightarrow aSa$ $n > 0$ veces.

2: Se tiene por otra parte que $S \Rightarrow^* bbabb$ (probado antes).

Propiedades de las relación de derivación

- **Extiende la relación de derivación directa** porque si $\alpha \Rightarrow \beta$ se tiene que $\alpha \Rightarrow^* \beta$.
- Es **reflexiva** porque $\alpha \Rightarrow^* \alpha$.
- Es **transitiva** porque si $\alpha \Rightarrow^* \beta$ y $\beta \Rightarrow^* \gamma$ entonces $\alpha \Rightarrow^* \gamma$.
- **Regla de inferencia de derivaciones**:
  Si tenemos la cadena $\alpha A \beta$ y sabemos que $A \Rightarrow^* \gamma$ entonces se puede deducir que $\alpha A \beta \Rightarrow^* \alpha \gamma \beta$.
  $\rightrightarrows$ **generaliza** la aplicación de una regla.

Ejemplo de derivación

Dada la GLC $S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$, podemos deducir que
$S \Rightarrow^* a^n\, bbabb\, a^n$ para todo $n > 0$ porque:
1: $S \Rightarrow^* a^n S a^n$ por aplicación de la regla $S \rightarrow aSa$ $n > 0$ veces.
2: Se tiene por otra parte que $S \Rightarrow^* bbabb$ (probado antes).
3: De 1 y 2 se deduce que: $S \Rightarrow^* a^n S a^n \Rightarrow^* a^n\, bbabb\, a^n$
luego $S \Rightarrow^* a^n\, bbabb\, a^n$ para todo $n > 0$.

Forma sentencial, sentencia y lenguaje generado

- **Forma sentencial:** es una cadena α que es derivable del símbolo inicial S (cumple $S \Rightarrow^* \alpha$)
- **Sentencia:** es una cadena w de símbolos terminales derivable de S (cumple $S \Rightarrow^* w$ y $w \in V_T^*$)

Forma sentencial, sentencia y lenguaje generado

- **Forma sentencial:** es una cadena α que es derivable del símbolo inicial S (cumple $S \Rightarrow^* \alpha$)
- **Sentencia:** es una cadena w de símbolos terminales derivable de S (cumple $S \Rightarrow^* w$ y $w \in V_T^*$)

  Ej.- Para la GLC $S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$ se tiene que

  $S \overbrace{\Rightarrow}^{S \rightarrow aSa}$

Forma sentencial, sentencia y lenguaje generado

- **Forma sentencial:** es una cadena α que es derivable del símbolo inicial S (cumple $S \Rightarrow^* \alpha$)
- **Sentencia:** es una cadena w de símbolos terminales derivable de S (cumple $S \Rightarrow^* w$ y $w \in V_T^*$)

  Ej.- Para la GLC $S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$ se tiene que

  $S \overbrace{\Rightarrow}^{S \rightarrow aSa} aSa \overbrace{\Rightarrow}^{S \rightarrow aSa}$

Forma sentencial, sentencia y lenguaje generado

- **Forma sentencial:** es una cadena α que es derivable del símbolo inicial S (cumple $S \Rightarrow^* \alpha$)
- **Sentencia:** es una cadena w de símbolos terminales derivable de S (cumple $S \Rightarrow^* w$ y $w \in V_T^*$)

  Ej.- Para la GLC $S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$ se tiene que

  $S \overbrace{\Rightarrow}^{S \rightarrow aSa} aSa \overbrace{\Rightarrow}^{S \rightarrow aSa} aaSaa \overbrace{\Rightarrow}^{S \rightarrow \lambda}$

Forma sentencial, sentencia y lenguaje generado

- **Forma sentencial:** es una cadena α que es derivable del símbolo inicial S (cumple $S \Rightarrow^* \alpha$)
- **Sentencia:** es una cadena w de símbolos terminales derivable de S (cumple $S \Rightarrow^* w$ y $w \in V_T^*$)

  Ej.- Para la GLC $S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$ se tiene que

  $S \overbrace{\Rightarrow}^{S \rightarrow aSa} aSa \overbrace{\Rightarrow}^{S \rightarrow aSa} aaSaa \overbrace{\Rightarrow}^{S \rightarrow \lambda} aaaa$, luego $S \Rightarrow^* aaaa$, por tanto $aaaa$ **es una sentencia**.

Forma sentencial, sentencia y lenguaje generado

- **Forma sentencial:** es una cadena α que es derivable del símbolo inicial S (cumple $S \Rightarrow^* \alpha$)
- **Sentencia:** es una cadena w de símbolos terminales derivable de S (cumple $S \Rightarrow^* w$ y $w \in V_T^*$)

  Ej.- Para la GLC $S \to aSa \mid bSb \mid a \mid b \mid \lambda$ se tiene que

  $S \overbrace{\Rightarrow}^{S \to aSa} aSa \overbrace{\Rightarrow}^{S \to aSa} aaSaa \overbrace{\Rightarrow}^{S \to \lambda} aaaa$, luego $S \Rightarrow^* aaaa$, por tanto $aaaa$ **es una sentencia**.

  Ej.- S, aSa, $aaSaa$, son **formas sentenciales** y $aaaa$ también.

Forma sentencial, sentencia y lenguaje generado

- **Forma sentencial:** es una cadena α que es derivable del símbolo inicial S (cumple $S \Rightarrow^* \alpha$)
- **Sentencia:** es una cadena w de símbolos terminales derivable de S (cumple $S \Rightarrow^* w$ y $w \in V_T^*$)

  Ej.- Para la GLC $S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$ se tiene que

  $S \overbrace{\Rightarrow}^{S \rightarrow aSa} aSa \overbrace{\Rightarrow}^{S \rightarrow aSa} aaSaa \overbrace{\Rightarrow}^{S \rightarrow \lambda} aaaa$, luego $S \Rightarrow^* aaaa$, por tanto $aaaa$ **es una sentencia**.

  Ej.- S, aSa, $aaSaa$, son **formas sentenciales** y $aaaa$ también.
- Se llama **lenguaje generado** por una gramática G al lenguaje formado por todas las sentencias de G:

$$\boxed{L(G) = \{w \in V_T^* \mid S \Rightarrow^* w\}}$$

Forma sentencial, sentencia y lenguaje generado

- **Forma sentencial:** es una cadena α que es derivable del símbolo inicial S (cumple $S \Rightarrow^* \alpha$)
- **Sentencia:** es una cadena w de símbolos terminales derivable de S (cumple $S \Rightarrow^* w$ y $w \in V_T^*$)

  Ej.- Para la GLC $S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$ se tiene que

  $S \overbrace{\Rightarrow}^{S \rightarrow aSa} aSa \overbrace{\Rightarrow}^{S \rightarrow aSa} aaSaa \overbrace{\Rightarrow}^{S \rightarrow \lambda} aaaa$, luego $S \Rightarrow^* aaaa$, por tanto $aaaa$ **es una sentencia**.

  Ej.- S, aSa, $aaSaa$, son **formas sentenciales** y $aaaa$ también.
- Se llama **lenguaje generado** por una gramática G al lenguaje formado por todas las sentencias de G:

$$\boxed{L(G) = \{w \in V_T^* \mid S \Rightarrow^* w\}}$$

- **¿Cual es el lenguaje generado por** $S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$**?**

Descripción de lenguajes mediante gramáticas

Una gramática G **es correcta** para describir la sintaxis de cierto lenguaje LEN (se cumple $LEN = L(G)$) si y sólo si:

- G **no es demasiado estricta:** cualquier cadena del lenguaje LEN es derivable del símbolo inicial (se cumple $LEN \subseteq L(G)$).
- G **no es demasiado general:** ninguna cadena fuera del lenguaje LEN es derivable del símbolo inicial (se cumple $L(G) \subseteq LEN$).

Ejemplo de justificación de corrección de una GLC

Partimos de $L_{aba} = \{a^{2i}b^{j+1}a^{2k} \mid i,j,k \geq 0\}$ y obtenemos G dada por:
$\mathbf{S} \rightarrow \mathbf{ABA}$, $\mathbf{A} \rightarrow \mathbf{aAa} \mid \lambda$, $\mathbf{B} \rightarrow \mathbf{bB} \mid \mathbf{b}$.
Justificamos que **es correcta** para ese lenguaje, **considerando todas las posibles derivaciones para cada variable**.

Ejemplo de justificación de corrección de una GLC

Partimos de $L_{aba} = \{a^{2i}b^{j+1}a^{2k} \mid i, j, k \geq 0\}$ y obtenemos G dada por:
$\mathbf{S} \rightarrow \mathbf{ABA}$, $\mathbf{A} \rightarrow \mathbf{aAa} \mid \lambda$, $\mathbf{B} \rightarrow \mathbf{bB} \mid \mathbf{b}$.
Justificamos que **es correcta** para ese lenguaje, **considerando todas las posibles derivaciones para cada variable**.

- Cadenas derivables de **A**:

$$A \overbrace{\Rightarrow^*}^{i \geq 0 \text{ veces } A \rightarrow aAa} a^iAa^i \overbrace{\Rightarrow}^{A \rightarrow \lambda} a^ia^i, (i \geq 0)$$

Por tanto $\boxed{A \Rightarrow^* a^{2i}, (i \geq 0)}$

Ejemplo de justificación de corrección de una GLC

Partimos de $L_{aba} = \{a^{2i} b^{j+1} a^{2k} \mid i, j, k \geq 0\}$ y obtenemos G dada por:
$\mathbf{S} \rightarrow \mathbf{ABA}$, $\mathbf{A} \rightarrow \mathbf{aAa} \mid \lambda$, $\mathbf{B} \rightarrow \mathbf{bB} \mid \mathbf{b}$.
Justificamos que **es correcta** para ese lenguaje, **considerando todas las posibles derivaciones para cada variable**.

- Cadenas derivables de **A**: $A \Rightarrow^* a^{2i}$, $(i \geq 0)$
- Cadenas derivables de **B**:

Ejemplo de justificación de corrección de una GLC

Partimos de $L_{aba} = \{a^{2i} b^{j+1} a^{2k} \mid i, j, k \geq 0\}$ y obtenemos G dada por:
$\mathbf{S} \rightarrow \mathbf{ABA}$, $\mathbf{A} \rightarrow \mathbf{aAa} \mid \lambda$, $\mathbf{B} \rightarrow \mathbf{bB} \mid \mathbf{b}$.
Justificamos que **es correcta** para ese lenguaje, **considerando todas las posibles derivaciones para cada variable**.

- <u>Cadenas derivables de **A**</u>: $A \Rightarrow^* a^{2i}, \ (i \geq 0)$
- <u>Cadenas derivables de **B**</u>:

$$B \quad \overbrace{\Rightarrow^*}^{j \geq 0 \text{ veces } B \rightarrow bB} \quad b^j B \overbrace{\Rightarrow}^{B \rightarrow b} b^j b, \ (j \geq 0)$$

Por tanto $B \Rightarrow^* b^{j+1}, \ (j \geq 0)$

Ejemplo de justificación de corrección de una GLC

Partimos de $L_{aba} = \{a^{2i}b^{j+1}a^{2k} \mid i, j, k \geq 0\}$ y obtenemos G dada por:
S $\rightarrow$ **ABA**, **A** $\rightarrow$ **aAa** $\mid \lambda$, **B** $\rightarrow$ **bB** $\mid$ **b**.
Justificamos que es **correcta** para ese lenguaje, **considerando todas las posibles derivaciones para cada variable**.

- Cadenas derivables de **A**: $A \Rightarrow^* a^{2i},\ (i \geq 0)$
- Cadenas derivables de **B**: $B \Rightarrow^* b^{j+1},\ (j \geq 0)$
- Cadenas derivables de **S**: $S \Rightarrow^* a^{2i}b^{j+1}a^{2k},\ (i, j, k \geq 0)$ porque:

Ejemplo de justificación de corrección de una GLC

Partimos de $L_{aba} = \{a^{2i}b^{j+1}a^{2k} \mid i,j,k \geq 0\}$ y obtenemos G dada por:
$\mathbf{S} \rightarrow \mathbf{ABA}$, $\mathbf{A} \rightarrow \mathbf{aAa} \mid \lambda$, $\mathbf{B} \rightarrow \mathbf{bB} \mid \mathbf{b}$.
Justificamos que es **correcta** para ese lenguaje, **considerando todas las posibles derivaciones para cada variable**.

- Cadenas derivables de **A**: $A \Rightarrow^* a^{2i},\ (i \geq 0)$
- Cadenas derivables de **B**: $B \Rightarrow^* b^{j+1},\ (j \geq 0)$
- Cadenas derivables de **S**: $S \Rightarrow^* a^{2i}b^{j+1}a^{2k},\ (i,j,k \geq 0)$ porque:

$$S \Rightarrow ABA \overbrace{\Rightarrow^*}^{\text{por A}}$$

Ejemplo de justificación de corrección de una GLC

Partimos de $L_{aba} = \{a^{2i}b^{j+1}a^{2k} \mid i,j,k \geq 0\}$ y obtenemos G dada por:
$\mathbf{S} \rightarrow \mathbf{ABA},\ \mathbf{A} \rightarrow \mathbf{aAa} \mid \lambda,\ \mathbf{B} \rightarrow \mathbf{bB} \mid \mathbf{b}$.
Justificamos que es **correcta** para ese lenguaje, **considerando todas las posibles derivaciones para cada variable**.

- Cadenas derivables de **A**: $A \Rightarrow^* a^{2i},\ (i \geq 0)$
- Cadenas derivables de **B**: $B \Rightarrow^* b^{j+1},\ (j \geq 0)$
- Cadenas derivables de **S**: $S \Rightarrow^* a^{2i}b^{j+1}a^{2k},\ (i,j,k \geq 0)$ porque:

$$S \Rightarrow ABA \overbrace{\Rightarrow^*}^{\text{por A}} a^{2i}BA \overbrace{\Rightarrow^*}^{\text{por B}}$$

Ejemplo de justificación de corrección de una GLC

Partimos de $L_{aba} = \{a^{2i}b^{j+1}a^{2k} \mid i,j,k \geq 0\}$ y obtenemos G dada por:
$\mathbf{S} \rightarrow \mathbf{ABA}$, $\mathbf{A} \rightarrow \mathbf{aAa} \mid \lambda$, $\mathbf{B} \rightarrow \mathbf{bB} \mid \mathbf{b}$.
Justificamos que es **correcta** para ese lenguaje, **considerando todas las posibles derivaciones para cada variable**.

- Cadenas derivables de **A**: $A \Rightarrow^* a^{2i}$, $(i \geq 0)$
- Cadenas derivables de **B**: $B \Rightarrow^* b^{j+1}$, $(j \geq 0)$
- Cadenas derivables de **S**: $S \Rightarrow^* a^{2i}b^{j+1}a^{2k}$, $(i,j,k \geq 0)$ porque:

$$S \Rightarrow ABA \overbrace{\Rightarrow^*}^{\text{por A}} a^{2i}BA \overbrace{\Rightarrow^*}^{\text{por B}} a^{2i}b^{j+1}A \overbrace{\Rightarrow^*}^{\text{por A}}$$

Ejemplo de justificación de corrección de una GLC

Partimos de $L_{aba} = \{a^{2i} b^{j+1} a^{2k} \mid i, j, k \geq 0\}$ y obtenemos G dada por:
$\mathbf{S} \rightarrow \mathbf{ABA},\ \mathbf{A} \rightarrow \mathbf{aAa} \mid \lambda,\ \mathbf{B} \rightarrow \mathbf{bB} \mid \mathbf{b}$.
Justificamos que es **correcta** para ese lenguaje, **considerando todas las posibles derivaciones para cada variable**.

- <u>Cadenas derivables de **A**</u>: $A \Rightarrow^* a^{2i},\ (i \geq 0)$
- <u>Cadenas derivables de **B**</u>: $B \Rightarrow^* b^{j+1},\ (j \geq 0)$
- <u>Cadenas derivables de **S**</u>: $S \Rightarrow^* a^{2i} b^{j+1} a^{2k},\ (i, j, k \geq 0)$ porque:

$$S \Rightarrow ABA \overbrace{\Rightarrow^*}^{\text{por A}} a^{2i} BA \overbrace{\Rightarrow^*}^{\text{por B}} a^{2i} b^{j+1} A \overbrace{\Rightarrow^*}^{\text{por A}} a^{2i} b^{j+1} a^{2k},\ (i, j, k \geq 0)$$

Ejemplo de justificación de corrección de una GLC

Partimos de $L_{aba} = \{a^{2i} b^{j+1} a^{2k} \mid i, j, k \geq 0\}$ y obtenemos G dada por:
$\mathbf{S} \rightarrow \mathbf{ABA}$, $\mathbf{A} \rightarrow \mathbf{aAa} \mid \lambda$, $\mathbf{B} \rightarrow \mathbf{bB} \mid \mathbf{b}$.
Justificamos que es **correcta** para ese lenguaje, **considerando todas las posibles derivaciones para cada variable**.

- Cadenas derivables de **A**: $A \Rightarrow^* a^{2i},\ (i \geq 0)$
- Cadenas derivables de **B**: $B \Rightarrow^* b^{j+1},\ (j \geq 0)$
- Cadenas derivables de **S**: $S \Rightarrow^* a^{2i} b^{j+1} a^{2k},\ (i, j, k \geq 0)$ porque:

$$S \Rightarrow ABA \overbrace{\Rightarrow^*}^{\text{por A}} a^{2i} BA \overbrace{\Rightarrow^*}^{\text{por B}} a^{2i} b^{j+1} A \overbrace{\Rightarrow^*}^{\text{por A}} a^{2i} b^{j+1} a^{2k},\ (i, j, k \geq 0)$$

Luego: $L(G) = \{a^{2i} b^{j+1} a^{2k} \mid i, j, k \geq 0\}$ y por tanto $\boxed{L(G) = L_{aba}}$ como queríamos probar.

Aplicación de GLC en descripción de sintaxis del lenguajes de programación

1. **Un programa comienza por la palabra clave *begin*, va seguido de un bloque de sentencias y termina con la palabra clave *end*.**

Ejemplo de gramática en BNF para el lenguaje anterior

$\langle$programa$\rangle \rightarrow$ **"begin"**$\langle$bloque-sentencias$\rangle$**"end"**
$\langle$bloque-sentencias$\rangle \rightarrow \langle$sentencia$\rangle \mid \langle$sentencia$\rangle\langle$bloque-sentencias$\rangle$
$\langle$sentencia$\rangle \rightarrow \langle$sent-if$\rangle \mid \langle$sent-while$\rangle \mid \langle$sent-asig$\rangle$
$\langle$sent-asig$\rangle \rightarrow \ldots$
$\ldots$

Aplicación de GLC en descripción de sintaxis del lenguajes de programación

1. Un programa comienza por la palabra clave *begin*, va seguido de un bloque de sentencias y termina con la palabra clave *end*.
2. **En un bloque de sentencias debe haber al menos una sentencia.**

Ejemplo de gramática en BNF para el lenguaje anterior

$\langle$programa$\rangle \rightarrow$ **"begin"**$\langle$bloque-sentencias$\rangle$**"end"**
$\langle$bloque-sentencias$\rangle \rightarrow \langle$sentencia$\rangle \mid \langle$sentencia$\rangle\langle$bloque-sentencias$\rangle$
$\langle$sentencia$\rangle \rightarrow \langle$sent-if$\rangle \mid \langle$sent-while$\rangle \mid \langle$sent-asig$\rangle$
$\langle$sent-asig$\rangle \rightarrow \ldots$
$\ldots$

Aplicación de GLC en descripción de sintaxis del lenguajes de programación

1. Un programa comienza por la palabra clave *begin*, va seguido de un bloque de sentencias y termina con la palabra clave *end*.
2. En un bloque de sentencias debe haber al menos una sentencia.
3. **Una sentencia puede ser una sentencia de asignación o una sentencia *if* o una sentencia *while*.**

Ejemplo de gramática en BNF para el lenguaje anterior

$\langle$programa$\rangle \rightarrow$ **"begin"**$\langle$bloque-sentencias$\rangle$**"end"**
$\langle$bloque-sentencias$\rangle \rightarrow \langle$sentencia$\rangle \mid \langle$sentencia$\rangle\langle$bloque-sentencias$\rangle$
$\langle$sentencia$\rangle \rightarrow \langle$sent-if$\rangle \mid \langle$sent-while$\rangle \mid \langle$sent-asig$\rangle$
$\langle$sent-asig$\rangle \rightarrow \ldots$
$\ldots$

Aplicación de GLC en descripción de sintaxis del lenguajes de programación

1. Un programa comienza por la palabra clave *begin*, va seguido de un bloque de sentencias y termina con la palabra clave *end*.
2. En un bloque de sentencias debe haber al menos una sentencia.
3. Una sentencia puede ser una sentencia de asignación o una sentencia *if* o una sentencia *while*.
4. **Un sentencia de asignación consiste en ...**

Ejemplo de gramática en BNF para el lenguaje anterior

$\langle$programa$\rangle \rightarrow$ **“begin”**$\langle$bloque-sentencias$\rangle$**“end”**
$\langle$bloque-sentencias$\rangle \rightarrow \langle$sentencia$\rangle$ | $\langle$sentencia$\rangle\langle$bloque-sentencias$\rangle$
$\langle$sentencia$\rangle \rightarrow \langle$sent-if$\rangle$ | $\langle$sent-while$\rangle$ | $\langle$sent-asig$\rangle$
$\langle$sent-asig$\rangle \rightarrow \ldots$
...

Aplicación de GLC en descripción de sintaxis del lenguajes de programación

1. Un programa comienza por la palabra clave *begin*, va seguido de un bloque de sentencias y termina con la palabra clave *end*.
2. En un bloque de sentencias debe haber al menos una sentencia.
3. Una sentencia puede ser una sentencia de asignación o una sentencia *if* o una sentencia *while*.
4. Un sentencia de asignación consiste en ...
5. ...

Ejemplo de gramática en BNF para el lenguaje anterior

$\langle \text{programa} \rangle \rightarrow$ **“begin”**$\langle \text{bloque-sentencias} \rangle$**“end”**
$\langle \text{bloque-sentencias} \rangle \rightarrow \langle \text{sentencia} \rangle \mid \langle \text{sentencia} \rangle \langle \text{bloque-sentencias} \rangle$
$\langle \text{sentencia} \rangle \rightarrow \langle \text{sent-if} \rangle \mid \langle \text{sent-while} \rangle \mid \langle \text{sent-asig} \rangle$
$\langle \text{sent-asig} \rangle \rightarrow \ldots$
...

Gramáticas para definiciones recursivas de lenguajes

Un lenguaje formal se puede especificar mediante una **definición recursiva**:

- Se describen las cadenas más simples que pertenecen al lenguaje (**caso base**)
- Se usan reglas recursivas para describir cadenas más complejas en función de otras cadenas más simples del lenguaje (**caso/s recursivo/s**).

Gramáticas para definiciones recursivas de lenguajes

Un lenguaje formal se puede especificar mediante una **definición recursiva**:

- Se describen las cadenas más simples que pertenecen al lenguaje (**caso base**)
- Se usan reglas recursivas para describir cadenas más complejas en función de otras cadenas más simples del lenguaje (**caso/s recursivo/s**).

Ejemplo de lenguaje definido recursivamente

Sea el lenguaje B_i, que se define recursivamente de la siguiente forma:

CASO BASE: $\lambda \in B_i$
CASO RECURSIVO 1: si $w \in B_i$ entonces $0w1 \in B_i$
CASO RECURSIVO 2: si $w \in B_i$ entonces $1w0 \in B_i$
CASO RECURSIVO 3: si $x, y \in B_i$ entonces $xy \in B_i$

Ejemplo: deducir lenguaje a partir de la definición recursiva

CASO BASE: $\lambda \in B_i$

CASO RECURSIVO 1: si $w \in B_i$ entonces $0w1 \in B_i$

CASO RECURSIVO 2: si $w \in B_i$ entonces $1w0 \in B_i$

CASO RECURSIVO 3: si $x, y \in B_i$ entonces $xy \in B_i$

Generamos cadenas según definición recursiva:

Ejemplo: deducir lenguaje a partir de la definición recursiva

CASO BASE: $\lambda \in B_i$
CASO RECURSIVO 1: si $w \in B_i$ entonces $0w1 \in B_i$
CASO RECURSIVO 2: si $w \in B_i$ entonces $1w0 \in B_i$
CASO RECURSIVO 3: si $x, y \in B_i$ entonces $xy \in B_i$

Generamos cadenas según definición recursiva:

- **Cadena(s) de menor longitud** $\rightrightarrows$ caso base: λ

Ejemplo: deducir lenguaje a partir de la definición recursiva

CASO BASE: $\lambda \in B_i$

CASO RECURSIVO 1: si $w \in B_i$ entonces $0w1 \in B_i$

CASO RECURSIVO 2: si $w \in B_i$ entonces $1w0 \in B_i$

CASO RECURSIVO 3: si $x, y \in B_i$ entonces $xy \in B_i$

Generamos cadenas según definición recursiva:

- **Cadena(s) de menor longitud** $\rightrightarrows$ caso base: λ
- **Siguientes cadenas en longitud:**
  λ + caso 1 $\rightrightarrows$ para $w = \lambda$ se obtiene $0\lambda 1 = 01$

Ejemplo: deducir lenguaje a partir de la definición recursiva

CASO BASE: $\lambda \in B_i$

CASO RECURSIVO 1: si $w \in B_i$ entonces $0w1 \in B_i$

CASO RECURSIVO 2: si $w \in B_i$ entonces $1w0 \in B_i$

CASO RECURSIVO 3: si $x, y \in B_i$ entonces $xy \in B_i$

Generamos cadenas según definición recursiva:

- **Cadena(s) de menor longitud** $\rightrightarrows$ caso base: λ
- **Siguientes cadenas en longitud:**
  λ + caso 1 $\rightrightarrows$ para $w = \lambda$ se obtiene $0\lambda1 = 01$
  λ + caso 2 $\rightrightarrows$ para $w = \lambda$ se obtiene $1\lambda0 = 10$

Ejemplo: deducir lenguaje a partir de la definición recursiva

CASO BASE: $\lambda \in B_i$

CASO RECURSIVO 1: si $w \in B_i$ entonces $0w1 \in B_i$

CASO RECURSIVO 2: si $w \in B_i$ entonces $1w0 \in B_i$

CASO RECURSIVO 3: si $x, y \in B_i$ entonces $xy \in B_i$

Generamos cadenas según definición recursiva:

- **Cadena(s) de menor longitud** $\rightrightarrows$ caso base: λ
- **Siguientes cadenas en longitud:**
  λ + caso 1 $\rightrightarrows$ para $w = \lambda$ se obtiene $0\lambda 1 = 01$
  λ + caso 2 $\rightrightarrows$ para $w = \lambda$ se obtiene $1\lambda 0 = 10$
  Cadenas ya generadas + caso 3 $\rightrightarrows$ si x, y son $01/10$ se obtiene:
  $01 \cdot 01 = 0101, 01 \cdot 10 = 0110, 10 \cdot 01 = 1001, 10 \cdot 10 = 1010$

Ejemplo: deducir lenguaje a partir de la definición recursiva

CASO BASE: $\lambda \in B_i$
CASO RECURSIVO 1: si $w \in B_i$ entonces $0w1 \in B_i$
CASO RECURSIVO 2: si $w \in B_i$ entonces $1w0 \in B_i$
CASO RECURSIVO 3: si $x, y \in B_i$ entonces $xy \in B_i$

Generamos cadenas según definición recursiva:

- **Cadena(s) de menor longitud** $\rightrightarrows$ caso base: λ
- **Siguientes cadenas en longitud:**
  λ + <u>caso 1</u> $\rightrightarrows$ para $w = \lambda$ se obtiene $0\lambda1 = 01$
  λ + <u>caso 2</u> $\rightrightarrows$ para $w = \lambda$ se obtiene $1\lambda0 = 10$
  Cadenas ya generadas + <u>caso 3</u> $\rightrightarrows$ si x, y son 01/10 se obtiene:
  $01 \cdot 01 = 0101, 01 \cdot 10 = 0110, 10 \cdot 01 = 1001, 10 \cdot 10 = 1010$
  Cadenas ya generadas + <u>casos 1/2</u> $\rightrightarrows$ si w es 01/10 se obtiene:
  $0 \cdot 01 \cdot 1 = 0011$ y $1 \cdot 10 \cdot 0 = 1100$.

Ejemplo: deducir lenguaje a partir de la definición recursiva

CASO BASE: $\lambda \in B_i$
CASO RECURSIVO 1: si $w \in B_i$ entonces $0w1 \in B_i$
CASO RECURSIVO 2: si $w \in B_i$ entonces $1w0 \in B_i$
CASO RECURSIVO 3: si $x, y \in B_i$ entonces $xy \in B_i$

Generamos cadenas según definición recursiva:

- **Cadena(s) de menor longitud** $\rightrightarrows$ caso base: λ
- **Siguientes cadenas en longitud:**
  λ + caso 1 $\rightrightarrows$ para $w = \lambda$ se obtiene $0\lambda 1 = 01$
  λ + caso 2 $\rightrightarrows$ para $w = \lambda$ se obtiene $1\lambda 0 = 10$
  Cadenas ya generadas + caso 3 $\rightrightarrows$ si x, y son $01/10$ se obtiene:
  $01 \cdot 01 = 0101, 01 \cdot 10 = 0110, 10 \cdot 01 = 1001, 10 \cdot 10 = 1010$
  Cadenas ya generadas + casos 1/2 $\rightrightarrows$ si w es $01/10$ se obtiene:
  $0 \cdot 01 \cdot 1 = 0011$ y $1 \cdot 10 \cdot 0 = 1100$.
  Y así sucesivamente...

¿Cómo es B_i?

Ejemplo: deducir lenguaje a partir de la definición recursiva

CASO BASE: $\lambda \in B_i$

CASO RECURSIVO 1: si $w \in B_i$ entonces $0w1 \in B_i$

CASO RECURSIVO 2: si $w \in B_i$ entonces $1w0 \in B_i$

CASO RECURSIVO 3: si $x, y \in B_i$ entonces $xy \in B_i$

Generamos cadenas según definición recursiva:

- **Cadena(s) de menor longitud** $\rightrightarrows$ caso base: λ
- **Siguientes cadenas en longitud:**
  λ + caso 1 $\rightrightarrows$ para $w = \lambda$ se obtiene $0\lambda 1 = 01$
  λ + caso 2 $\rightrightarrows$ para $w = \lambda$ se obtiene $1\lambda 0 = 10$
  Cadenas ya generadas + caso 3 $\rightrightarrows$ si x, y son $01/10$ se obtiene:
  $01 \cdot 01 = 0101, 01 \cdot 10 = 0110, 10 \cdot 01 = 1001, 10 \cdot 10 = 1010$
  Cadenas ya generadas + casos 1/2 $\rightrightarrows$ si w es $01/10$ se obtiene:
  $0 \cdot 01 \cdot 1 = 0011$ y $1 \cdot 10 \cdot 0 = 1100$.
  Y así sucesivamente...

¿Cómo es B_i? $B_i = \{w \in \{0,1\}^* \mid ceros(w) = unos(w)\}$

Ejemplo: obtener GLC a partir de definición recursiva

CASO BASE: $\lambda \in B_i$

CASO RECURSIVO 1: si $w \in B_i$ entonces $0w1 \in B_i$

CASO RECURSIVO 2: si $w \in B_i$ entonces $1w0 \in B_i$

CASO RECURSIVO 3: si $x, y \in B_i$ entonces $xy \in B_i$

- **Base:** Incluir la regla $S \to \lambda$ para que $S \Rightarrow^* \lambda$, ya que $\lambda \in B_i$.

Ejemplo: obtener GLC a partir de definición recursiva

CASO BASE: $\lambda \in B_i$

CASO RECURSIVO 1: si $w \in B_i$ entonces $0w1 \in B_i$

CASO RECURSIVO 2: si $w \in B_i$ entonces $1w0 \in B_i$

CASO RECURSIVO 3: si $x, y \in B_i$ entonces $xy \in B_i$

- **Base:** Incluir la regla $S \rightarrow \lambda$ para que $S \Rightarrow^* \lambda$, ya que $\lambda \in B_i$.
- Se supone que $w \in B_i$ equivale a $S \Rightarrow^* w$ y según definición $0w1 \in B_i$. Por eso incluimos la regla $S \rightarrow 0S1$
  Así $S \Rightarrow 0S1 \Rightarrow^* 0w1$, lo que asegura que $0w1$ es derivable de S.

Ejemplo: obtener GLC a partir de definición recursiva

CASO BASE: $\lambda \in B_i$

CASO RECURSIVO 1: si $w \in B_i$ entonces $0w1 \in B_i$

CASO RECURSIVO 2: si $w \in B_i$ entonces $1w0 \in B_i$

CASO RECURSIVO 3: si $x, y \in B_i$ entonces $xy \in B_i$

- **Base:** Incluir la regla $S \rightarrow \lambda$ para que $S \Rightarrow^* \lambda$, ya que $\lambda \in B_i$.
- Se supone que $w \in B_i$ equivale a $S \Rightarrow^* w$ y según definición $0w1 \in B_i$. Por eso incluimos la regla $S \rightarrow 0S1$
  Así $S \Rightarrow 0S1 \Rightarrow^* 0w1$, lo que asegura que $0w1$ es derivable de S.
- Se supone que $w \in B_i$ equivale a $S \Rightarrow^* w$ y según definición $1w0 \in B_i$. Por eso incluimos la regla $S \rightarrow 1S0$
  Así $S \Rightarrow 1S0 \Rightarrow^* 1w0$, lo que asegura que $1w0$ es derivable de S.

Ejemplo: obtener GLC a partir de definición recursiva

CASO BASE: $\lambda \in B_i$
CASO RECURSIVO 1: si $w \in B_i$ entonces $0w1 \in B_i$
CASO RECURSIVO 2: si $w \in B_i$ entonces $1w0 \in B_i$
CASO RECURSIVO 3: si $x, y \in B_i$ entonces $xy \in B_i$

- **Base:** Incluir la regla $S \to \lambda$ para que $S \Rightarrow^* \lambda$, ya que $\lambda \in B_i$.
- Se supone que $w \in B_i$ equivale a $S \Rightarrow^* w$ y según definición $0w1 \in B_i$. Por eso incluimos la regla $S \to 0S1$
  Así $S \Rightarrow 0S1 \Rightarrow^* 0w1$, lo que asegura que $0w1$ es derivable de S.
- Se supone que $w \in B_i$ equivale a $S \Rightarrow^* w$ y según definición $1w0 \in B_i$. Por eso incluimos la regla $S \to 1S0$
  Así $S \Rightarrow 1S0 \Rightarrow^* 1w0$, lo que asegura que $1w0$ es derivable de S.
- Se supone que $x, y \in B_i$ equivale a $S \Rightarrow^* x$, $S \Rightarrow^* y$ y según definición $xy \in B_i$. Por eso incluimos la regla $S \to SS$
  Así $S \Rightarrow SS \Rightarrow^* xS \Rightarrow^* xy$, por lo que xy es derivable de S.

Ejemplo: obtener GLC a partir de definición recursiva

CASO BASE: $\lambda \in B_i$
CASO RECURSIVO 1: si $w \in B_i$ entonces $0w1 \in B_i$
CASO RECURSIVO 2: si $w \in B_i$ entonces $1w0 \in B_i$
CASO RECURSIVO 3: si $x, y \in B_i$ entonces $xy \in B_i$

- **Base:** Incluir la regla $S \to \lambda$ para que $S \Rightarrow^* \lambda$, ya que $\lambda \in B_i$.
- Se supone que $w \in B_i$ equivale a $S \Rightarrow^* w$ y según definición $0w1 \in B_i$. Por eso incluimos la regla $S \to 0S1$
  Así $S \Rightarrow 0S1 \Rightarrow^* 0w1$, lo que asegura que $0w1$ es derivable de S.
- Se supone que $w \in B_i$ equivale a $S \Rightarrow^* w$ y según definición $1w0 \in B_i$. Por eso incluimos la regla $S \to 1S0$
  Así $S \Rightarrow 1S0 \Rightarrow^* 1w0$, lo que asegura que $1w0$ es derivable de S.
- Se supone que $x, y \in B_i$ equivale a $S \Rightarrow^* x$, $S \Rightarrow^* y$ y según definición $xy \in B_i$. Por eso incluimos la regla $S \to SS$
  Así $S \Rightarrow SS \Rightarrow^* xS \Rightarrow^* xy$, por lo que xy es derivable de S.

Ejemplo: obtener GLC a partir de definición recursiva

En problema resueltos: **lenguaje de los paréntesis balanceados (PB)**

CASO BASE: $() \in PB$

CASO RECURSIVO 1: si $w \in PB$ entonces $(w) \in PB$

CASO RECURSIVO 3: si $x, y \in PB$ entonces $xy \in PB$

Ejemplo: obtener GLC a partir de definición recursiva

En problema resueltos: **lenguaje de los paréntesis balanceados (PB)**

CASO BASE: $() \in PB$

CASO RECURSIVO 1: si $w \in PB$ entonces $(w) \in PB$

CASO RECURSIVO 3: si $x, y \in PB$ entonces $xy \in PB$

- **Base:** Incluir la regla $S \rightarrow ()$

Ejemplo: obtener GLC a partir de definición recursiva

En problema resueltos: **lenguaje de los paréntesis balanceados (PB)**

CASO BASE: $() \in PB$

CASO RECURSIVO 1: si $w \in PB$ entonces $(w) \in PB$

CASO RECURSIVO 3: si $x, y \in PB$ entonces $xy \in PB$

- **Base:** Incluir la regla $S \to ()$
- Se supone que $w \in PB$ equivale a $S \Rightarrow^* w$ y según definición $(w) \in PB$. Por eso incluimos la regla $S \to (S)$

Ejemplo: obtener GLC a partir de definición recursiva

En problema resueltos: **lenguaje de los paréntesis balanceados (PB)**

CASO BASE: $() \in PB$

CASO RECURSIVO 1: si $w \in PB$ entonces $(w) \in PB$

CASO RECURSIVO 3: si $x, y \in PB$ entonces $xy \in PB$

- **Base:** Incluir la regla $S \to ()$
- Se supone que $w \in PB$ equivale a $S \Rightarrow^* w$ y según definición $(w) \in PB$. Por eso incluimos la regla $S \to (S)$
- Se supone que $x, y \in PB$ equivale a $S \Rightarrow^* x$, $S \Rightarrow^* y$ y según definición $xy \in PB$. Por eso incluimos la regla $S \to SS$

Ejemplo: obtener GLC a partir de definición recursiva

En problema resueltos: **lenguaje de los paréntesis balanceados (PB)**

CASO BASE: $() \in PB$
CASO RECURSIVO 1: si $w \in PB$ entonces $(w) \in PB$
CASO RECURSIVO 3: si $x, y \in PB$ entonces $xy \in PB$

- **Base:** Incluir la regla $S \to ()$
- Se supone que $w \in PB$ equivale a $S \Rightarrow^* w$ y según definición $(w) \in PB$. Por eso incluimos la regla $S \to (S)$
- Se supone que $x, y \in PB$ equivale a $S \Rightarrow^* x$, $S \Rightarrow^* y$ y según definición $xy \in PB$. Por eso incluimos la regla $S \to SS$

Análisis sintáctico

Consiste en comprobar si una cadena es una sentencia de la gramática, es decir, comprobar si es derivable del símbolo inicial.

Los **algoritmos de análisis sintáctico** se estudian en Compiladores.

Análisis sintáctico

Consiste en comprobar si una cadena es una sentencia de la gramática, es decir, comprobar si es derivable del símbolo inicial.
Los **algoritmos de análisis sintáctico** se estudian en Compiladores.

Ejemplo de cadenas sintácticamente correctas e incorrectas

Sea la gramática G con reglas $S \to ABA$, $A \to aAa \mid \lambda$, $B \to bB \mid b$.

- Comprobamos que **aabaaaa es sintácticamente correcta** según la gramática. Debemos probar que $S \Rightarrow^* aabaaaa$:

Análisis sintáctico

Consiste en comprobar si una cadena es una sentencia de la gramática, es decir, comprobar si es derivable del símbolo inicial.
Los **algoritmos de análisis sintáctico** se estudian en Compiladores.

Ejemplo de cadenas sintácticamente correctas e incorrectas

Sea la gramática G con reglas $S \to ABA$, $A \to aAa \mid \lambda$, $B \to bB \mid b$.

- Comprobamos que **aabaaaa es sintácticamente correcta** según la gramática. Debemos probar que $S \Rightarrow^* aabaaaa$:

$$S \overbrace{\Rightarrow}^{S \to ABA} ABA$$

Análisis sintáctico

Consiste en comprobar si una cadena es una sentencia de la gramática, es decir, comprobar si es derivable del símbolo inicial.
Los **algoritmos de análisis sintáctico** se estudian en Compiladores.

Ejemplo de cadenas sintácticamente correctas e incorrectas

Sea la gramática G con reglas $S \to ABA$, $A \to aAa \mid \lambda$, $B \to bB \mid b$.

- Comprobamos que **aabaaaa es sintácticamente correcta** según la gramática. Debemos probar que $S \Rightarrow^* aabaaaa$:

$$S \overbrace{\Rightarrow}^{S \to ABA} ABA \overbrace{\Rightarrow}^{A \to aAa}$$

Análisis sintáctico

Consiste en comprobar si una cadena es una sentencia de la gramática, es decir, comprobar si es derivable del símbolo inicial.
Los **algoritmos de análisis sintáctico** se estudian en Compiladores.

Ejemplo de cadenas sintácticamente correctas e incorrectas

Sea la gramática G con reglas $S \to ABA$, $A \to aAa \mid \lambda$, $B \to bB \mid b$.

- Comprobamos que **aabaaaa es sintácticamente correcta** según la gramática. Debemos probar que $S \Rightarrow^* aabaaaa$:

$$S \overbrace{\Rightarrow}^{S \to ABA} ABA \overbrace{\Rightarrow}^{A \to aAa} aAaBA \overbrace{\Rightarrow}^{A \to \lambda}$$

Análisis sintáctico

Consiste en comprobar si una cadena es una sentencia de la gramática, es decir, comprobar si es derivable del símbolo inicial.
Los **algoritmos de análisis sintáctico** se estudian en Compiladores.

Ejemplo de cadenas sintácticamente correctas e incorrectas

Sea la gramática G con reglas $S \to ABA$, $A \to aAa \mid \lambda$, $B \to bB \mid b$.

- Comprobamos que **aabaaaa es sintácticamente correcta** según la gramática. Debemos probar que $S \Rightarrow^* aabaaaa$:

$$S \overbrace{\Rightarrow}^{S\to ABA} ABA \overbrace{\Rightarrow}^{A\to aAa} aAaBA \overbrace{\Rightarrow}^{A\to\lambda} aaBA \overbrace{\Rightarrow}^{B\to b}$$

Análisis sintáctico

Consiste en comprobar si una cadena es una sentencia de la gramática, es decir, comprobar si es derivable del símbolo inicial.
Los **algoritmos de análisis sintáctico** se estudian en Compiladores.

Ejemplo de cadenas sintácticamente correctas e incorrectas

Sea la gramática G con reglas $S \to ABA,\ A \to aAa \mid \lambda,\ B \to bB \mid b$.

- Comprobamos que **aabaaaa es sintácticamente correcta** según la gramática. Debemos probar que $S \Rightarrow^* aabaaaa$:

$$S \overbrace{\Rightarrow}^{S\to ABA} ABA \overbrace{\Rightarrow}^{A\to aAa} aAaBA \overbrace{\Rightarrow}^{A\to\lambda} aaBA \overbrace{\Rightarrow}^{B\to b} aabA \overbrace{\Rightarrow}^{A\to aAa}$$

Análisis sintáctico

Consiste en comprobar si una cadena es una sentencia de la gramática, es decir, comprobar si es derivable del símbolo inicial.
Los **algoritmos de análisis sintáctico** se estudian en Compiladores.

Ejemplo de cadenas sintácticamente correctas e incorrectas

Sea la gramática G con reglas $S \to ABA,\ A \to aAa \mid \lambda,\ B \to bB \mid b$.

- Comprobamos que **aabaaaa es sintácticamente correcta** según la gramática. Debemos probar que $S \Rightarrow^* aabaaaa$:

$$S \overbrace{\Rightarrow}^{S\to ABA} ABA \overbrace{\Rightarrow}^{A\to aAa} aAaBA \overbrace{\Rightarrow}^{A\to\lambda} aaBA \overbrace{\Rightarrow}^{B\to b} aabA \overbrace{\Rightarrow}^{A\to aAa} aabaAa$$
$$\overbrace{\Rightarrow}^{A\to aAa}$$

Análisis sintáctico

Consiste en comprobar si una cadena es una sentencia de la gramática, es decir, comprobar si es derivable del símbolo inicial.
Los **algoritmos de análisis sintáctico** se estudian en Compiladores.

Ejemplo de cadenas sintácticamente correctas e incorrectas

Sea la gramática G con reglas $S \to ABA$, $A \to aAa \mid \lambda$, $B \to bB \mid b$.

- Comprobamos que **aabaaaa es sintácticamente correcta** según la gramática. Debemos probar que $S \Rightarrow^* aabaaaa$:

$$S \overbrace{\Rightarrow}^{S\to ABA} ABA \overbrace{\Rightarrow}^{A\to aAa} aAaBA \overbrace{\Rightarrow}^{A\to\lambda} aaBA \overbrace{\Rightarrow}^{B\to b} aabA \overbrace{\Rightarrow}^{A\to aAa} aabaAa$$
$$\overbrace{\Rightarrow}^{A\to aAa} aabaaAaa$$

Análisis sintáctico

Consiste en comprobar si una cadena es una sentencia de la gramática, es decir, comprobar si es derivable del símbolo inicial.
Los **algoritmos de análisis sintáctico** se estudian en Compiladores.

Ejemplo de cadenas sintácticamente correctas e incorrectas

Sea la gramática G con reglas $S \to ABA,\ A \to aAa \mid \lambda,\ B \to bB \mid b$.

- Comprobamos que **aabaaaa es sintácticamente correcta** según la gramática. Debemos probar que $S \Rightarrow^* aabaaaa$:

$$S \overbrace{\Rightarrow}^{S\to ABA} ABA \overbrace{\Rightarrow}^{A\to aAa} aAaBA \overbrace{\Rightarrow}^{A\to\lambda} aaBA \overbrace{\Rightarrow}^{B\to b} aabA \overbrace{\Rightarrow}^{A\to aAa} aabaAa$$
$$\overbrace{\Rightarrow}^{A\to aAa} aabaaAaa \overbrace{\Rightarrow}^{A\to\lambda} aabaaaa\ (S \Rightarrow^* aabaaaa \text{ en 7 pasos})$$

Análisis sintáctico

Consiste en comprobar si una cadena es una sentencia de la gramática, es decir, comprobar si es derivable del símbolo inicial.
Los **algoritmos de análisis sintáctico** se estudian en Compiladores.

Ejemplo de cadenas sintácticamente correctas e incorrectas

Sea la gramática G con reglas $S \to ABA$, $A \to aAa \mid \lambda$, $B \to bB \mid b$.

- Comprobamos que **aabaaaa es sintácticamente correcta** según la gramática. Debemos probar que $S \Rightarrow^* aabaaaa$:

$$S \overbrace{\Rightarrow}^{S\to ABA} ABA \overbrace{\Rightarrow}^{A\to aAa} aAaBA \overbrace{\Rightarrow}^{A\to\lambda} aaBA \overbrace{\Rightarrow}^{B\to b} aabA \overbrace{\Rightarrow}^{A\to aAa} aabaAa$$
$$\overbrace{\Rightarrow}^{A\to aAa} aabaaAaa \overbrace{\Rightarrow}^{A\to\lambda} aabaaaa \; (S \Rightarrow^* aabaaaa \text{ en 7 pasos})$$

- La cadena λ no es una sentencia de la gramática.

Análisis sintáctico

Consiste en comprobar si una cadena es una sentencia de la gramática, es decir, comprobar si es derivable del símbolo inicial.
Los **algoritmos de análisis sintáctico** se estudian en Compiladores.

Ejemplo de cadenas sintácticamente correctas e incorrectas

Sea la gramática G con reglas $S \to ABA,\ A \to aAa \mid \lambda,\ B \to bB \mid b$.

- Comprobamos que **aabaaaa es sintácticamente correcta** según la gramática. Debemos probar que $S \Rightarrow^* aabaaaa$:

$$S \overbrace{\Rightarrow}^{S\to ABA} ABA \overbrace{\Rightarrow}^{A\to aAa} aAaBA \overbrace{\Rightarrow}^{A\to\lambda} aaBA \overbrace{\Rightarrow}^{B\to b} aabA \overbrace{\Rightarrow}^{A\to aAa} aabaAa$$
$$\overbrace{\Rightarrow}^{A\to aAa} aabaaAaa \overbrace{\Rightarrow}^{A\to\lambda} aabaaaa \ (S \Rightarrow^* aabaaaa \text{ en 7 pasos})$$

- La cadena λ no es una sentencia de la gramática.
- La cadena $babb$ no es una sentencia.

Árbol de derivación

Un **árbol de derivación** **describe gráficamente la estructura de una sentencia**. Los árboles de derivación se usan en **análisis sintáctico y en generación de código** en diversos tipos de traductores.

Árbol de derivación

Un árbol de derivación **describe gráficamente la estructura de una sentencia**. Los árboles de derivación se usan en **análisis sintáctico y en generación de código** en diversos tipos de traductores.

Un árbol para una cadena w **se construye** teniendo en cuenta que:

- El **nodo raíz** se etiqueta con el símbolo inicial S y cada **nodo hoja** se etiquetará con un símbolo de w.
- Cada variable que aparece en una forma sentencial en la derivación de w está representada en un nodo **nodo-variable**.
- Cada nodo-variable tiene asociado un **subárbol** cuyas hojas representan la porción de la sentencia que se puede derivar a partir de esa variable.

Ej. de árbol y derivaciones MI y MD de una sentencia

Gramática:

E $\rightarrow$ E + E
E $\rightarrow$ E * E
E $\rightarrow$ (E)
E $\rightarrow$ **i**

árbol de derivación para $i + i * i$

E

Derivación más a la izquierda (MI):

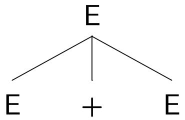
$$E$$

Ej. de árbol y derivaciones MI y MD de una sentencia

Gramática:

$E \rightarrow E + E$
$E \rightarrow E$ * E
$E \rightarrow (E)$
$E \rightarrow$ **i**

árbol de derivación para $i + i * i$



Derivación más a la izquierda (MI):

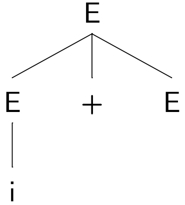
$$E \Rightarrow \boxed{E} + E$$

Ej. de árbol y derivaciones MI y MD de una sentencia

Gramática:

E $\rightarrow$ E + E
E $\rightarrow$ E * E
E $\rightarrow$ (E)
E $\rightarrow$ **i**

árbol de derivación para $i + i * i$



Derivación más a la izquierda (MI):

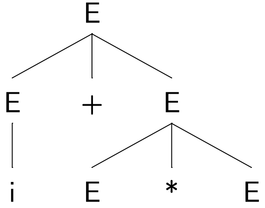
$$E \Rightarrow \boxed{E} + E \Rightarrow i + \boxed{E}$$

Ej. de árbol y derivaciones MI y MD de una sentencia

Gramática:

E → E + E
E → E * E
E → (E)
E → **i**

árbol de derivación para $i + i * i$



Derivación más a la izquierda (MI):

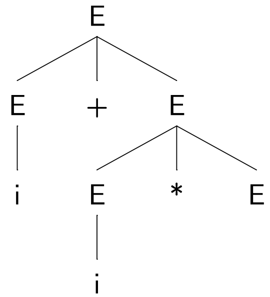
$$E \Rightarrow \boxed{E} + E \Rightarrow i + \boxed{E} \Rightarrow i + \boxed{E} * E$$

Ej. de árbol y derivaciones MI y MD de una sentencia

Gramática:

E $\rightarrow$ E + E
E $\rightarrow$ E * E
E $\rightarrow$ (E)
E $\rightarrow$ **i**

árbol de derivación para $i + i * i$



Derivación más a la izquierda (MI):

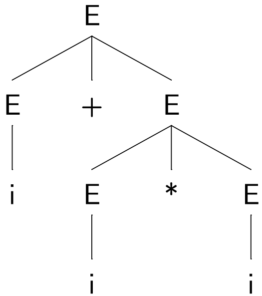
$$E \Rightarrow \boxed{E} + E \Rightarrow i + \boxed{E} \Rightarrow i + \boxed{E} * E \Rightarrow i + i * \boxed{E}$$

Ej. de árbol y derivaciones MI y MD de una sentencia

Gramática:

E $\rightarrow$ E + E
E $\rightarrow$ E * E
E $\rightarrow$ (E)
E $\rightarrow$ **i**

árbol de derivación para $i + i * i$



Derivación más a la izquierda (MI):

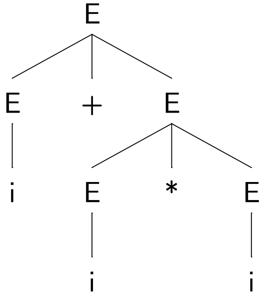
$$E \Rightarrow \boxed{E} + E \Rightarrow i + \boxed{E} \Rightarrow i + \boxed{E} * E \Rightarrow i + i * \boxed{E} \Rightarrow i + i * i$$

Ej. de árbol y derivaciones MI y MD de una sentencia

Gramática:

E $\rightarrow$ E + E
E $\rightarrow$ E * E
E $\rightarrow$ (E)
E $\rightarrow$ **i**

árbol de derivación para $i + i * i$



Derivación más a la izquierda (MI):

$$E \Rightarrow \boxed{E} + E \Rightarrow i + \boxed{E} \Rightarrow i + \boxed{E} * E \Rightarrow i + i * \boxed{E} \Rightarrow i + i * i$$

Derivación más a la derecha (MD):

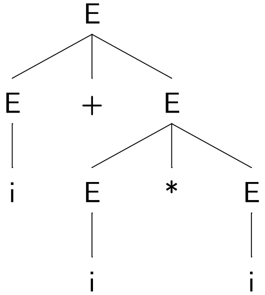
$$E \Rightarrow E + \boxed{E}$$

Ej. de árbol y derivaciones MI y MD de una sentencia

Gramática:

E $\rightarrow$ E + E
E $\rightarrow$ E * E
E $\rightarrow$ (E)
E $\rightarrow$ **i**

árbol de derivación para $i + i * i$



Derivación más a la izquierda (MI):

$$E \Rightarrow \boxed{E} + E \Rightarrow i + \boxed{E} \Rightarrow i + \boxed{E} * E \Rightarrow i + i * \boxed{E} \Rightarrow i + i * i$$

Derivación más a la derecha (MD):

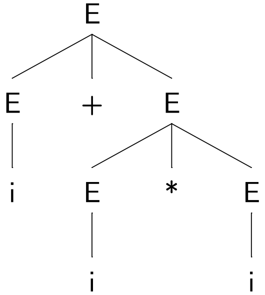
$$E \Rightarrow E + \boxed{E} \Rightarrow E + E * \boxed{E}$$

Ej. de árbol y derivaciones MI y MD de una sentencia

Gramática:

E → E + E
E → E * E
E → (E)
E → **i**

árbol de derivación para $i + i * i$



Derivación más a la izquierda (MI):

$$E \Rightarrow \boxed{E} + E \Rightarrow i + \boxed{E} \Rightarrow i + \boxed{E} * E \Rightarrow i + i * \boxed{E} \Rightarrow i + i * i$$

Derivación más a la derecha (MD):

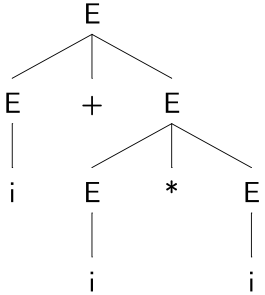
$$E \Rightarrow E + \boxed{E} \Rightarrow E + E * \boxed{E} \Rightarrow E + \boxed{E} * i$$

Ej. de árbol y derivaciones MI y MD de una sentencia

Gramática:

E → E + E
E → E * E
E → (E)
E → **i**

árbol de derivación para $i + i * i$



Derivación más a la izquierda (MI):

$$E \Rightarrow \boxed{E} + E \Rightarrow i + \boxed{E} \Rightarrow i + \boxed{E} * E \Rightarrow i + i * \boxed{E} \Rightarrow i + i * i$$

Derivación más a la derecha (MD):

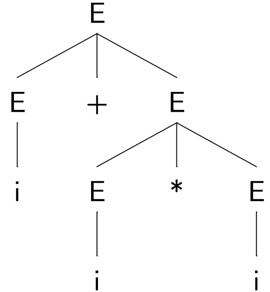
$$E \Rightarrow E + \boxed{E} \Rightarrow E + E * \boxed{E} \Rightarrow E + \boxed{E} * i \Rightarrow \boxed{E} + i * i$$

Ej. de árbol y derivaciones MI y MD de una sentencia

Gramática:

E → E + E
E → E * E
E → (E)
E → **i**

árbol de derivación para $i + i * i$



Derivación más a la izquierda (MI):

$$E \Rightarrow \boxed{E} + E \Rightarrow i + \boxed{E} \Rightarrow i + \boxed{E} * E \Rightarrow i + i * \boxed{E} \Rightarrow i + i * i$$

Derivación más a la derecha (MD):

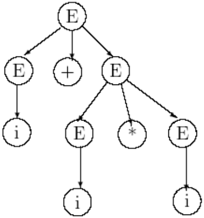
$$E \Rightarrow E + \boxed{E} \Rightarrow E + E * \boxed{E} \Rightarrow E + \boxed{E} * i \Rightarrow \boxed{E} + i * i \Rightarrow i + i * i$$

Ambigüedad

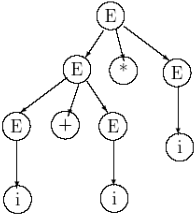
Una **gramática ambigua** es una gramática para la cual podemos encontrar al menos una **sentencia ambigua**.
Una **sentencia ambigua** es aquella que tiene **dos árboles de derivación distintos**, o equivalentemente, **dos derivaciones más a la izquierda** distintas o **dos derivaciones más a la derecha distintas**.

Ejemplo de ambigüedad

$E \rightarrow E + E \mid E * E \mid (E) \mid i$ es una **gramática ambigua** porque tiene una **sentencia ambigua**, por ej. $i + i * i$ $\rightrightarrows$ ya que tiene **dos árboles de derivación distintos**:



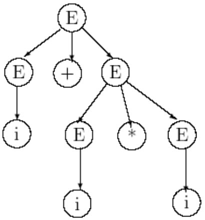
ARBOL (1)



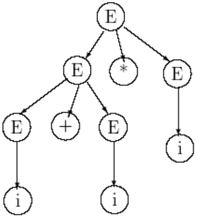
ARBOL (2)

Ejemplo de ambigüedad

$E \to E + E \mid E * E \mid (E) \mid i$ es una **gramática ambigua** porque tiene una **sentencia ambigua**, por ej. $i + i * i$ $\rightrightarrows$ ya que tiene **dos árboles de derivación distintos**:



ARBOL (1)



ARBOL (2)

También hay **dos derivaciones más a la izquierda distintas**.

Derivación más a la izquierda (1) $\rightrightarrows$ árbol (1), significado $i + (i * i)$:

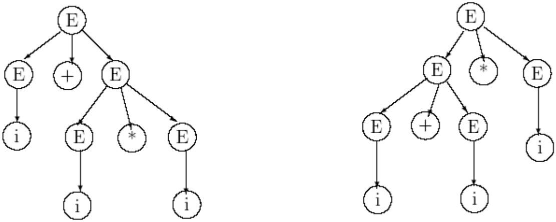
$$E$$

Derivación más a la izquierda (2) $\rightrightarrows$ árbol (2), significado $(i + i) * i$:

$$E$$

Ejemplo de ambigüedad

$E \to E + E \mid E * E \mid (E) \mid i$ es una **gramática ambigua** porque tiene una **sentencia ambigua**, por ej. $i + i * i$ $\rightrightarrows$ ya que tiene **dos árboles de derivación distintos**:



ARBOL (1) ARBOL (2)

También hay **dos derivaciones más a la izquierda distintas**.

Derivación más a la izquierda (1) $\rightrightarrows$ árbol (1), significado $i + (i * i)$:

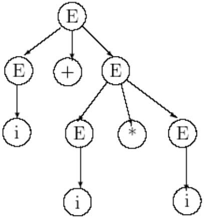
$$E \Rightarrow \boxed{E} + E$$

Derivación más a la izquierda (2) $\rightrightarrows$ árbol (2), significado $(i + i) * i$:

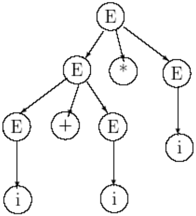
$$E \Rightarrow \boxed{E} * E$$

Ejemplo de ambigüedad

$E \rightarrow E + E \mid E * E \mid (E) \mid i$ es una **gramática ambigua** porque tiene una **sentencia ambigua**, por ej. $i + i * i$ $\rightrightarrows$ ya que tiene **dos árboles de derivación distintos**:



ARBOL (1)



ARBOL (2)

También hay **dos derivaciones más a la izquierda distintas**.

Derivación más a la izquierda (1) $\rightrightarrows$ árbol (1), significado $i + (i * i)$:

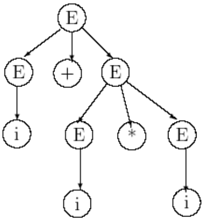
$$E \Rightarrow \boxed{E} + E \Rightarrow i + \boxed{E}$$

Derivación más a la izquierda (2) $\rightrightarrows$ árbol (2), significado $(i + i) * i$:

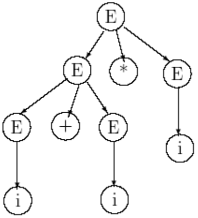
$$E \Rightarrow \boxed{E} * E \Rightarrow \boxed{E} + E * E$$

Ejemplo de ambigüedad

$E \rightarrow E + E \mid E * E \mid (E) \mid i$ es una **gramática ambigua** porque tiene una **sentencia ambigua**, por ej. $i + i * i \Rightarrow$ ya que tiene **dos árboles de derivación distintos**:



ARBOL (1)



ARBOL (2)

También hay **dos derivaciones más a la izquierda distintas**.

Derivación más a la izquierda (1) $\Rightarrow$ árbol (1), significado $i + (i * i)$:

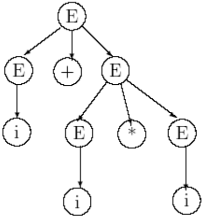
$$E \Rightarrow \boxed{E} + E \Rightarrow i + \boxed{E} \Rightarrow i + \boxed{E} * E$$

Derivación más a la izquierda (2) $\Rightarrow$ árbol (2), significado $(i + i) * i$:

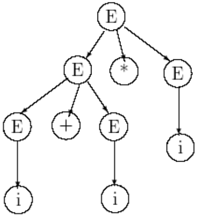
$$E \Rightarrow \boxed{E} * E \Rightarrow \boxed{E} + E * E \Rightarrow i + \boxed{E} * E$$

Ejemplo de ambigüedad

$E \rightarrow E + E \mid E * E \mid (E) \mid i$ es una **gramática ambigua** porque tiene una **sentencia ambigua**, por ej. $i + i * i$ $\rightrightarrows$ ya que tiene **dos árboles de derivación distintos**:



ARBOL (1)



ARBOL (2)

También hay **dos derivaciones más a la izquierda distintas**.

Derivación más a la izquierda (1) $\rightrightarrows$ árbol (1), significado $i + (i * i)$:

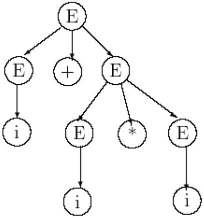
$$E \Rightarrow \boxed{E} + E \Rightarrow i + \boxed{E} \Rightarrow i + \boxed{E} * E \Rightarrow i + i * \boxed{E}$$

Derivación más a la izquierda (2) $\rightrightarrows$ árbol (2), significado $(i + i) * i$:

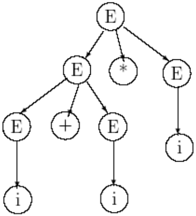
$$E \Rightarrow \boxed{E} * E \Rightarrow \boxed{E} + E * E \Rightarrow i + \boxed{E} * E \Rightarrow i + i * \boxed{E}$$

Ejemplo de ambigüedad

$E \to E + E \mid E * E \mid (E) \mid i$ es una **gramática ambigua** porque tiene una **sentencia ambigua**, por ej. $i + i * i$ $\rightrightarrows$ ya que tiene **dos árboles de derivación distintos**:



ARBOL (1)



ARBOL (2)

También hay **dos derivaciones más a la izquierda distintas**.

Derivación más a la izquierda (1) $\rightrightarrows$ árbol (1), significado $i + (i * i)$:

$$E \Rightarrow \boxed{E} + E \Rightarrow i + \boxed{E} \Rightarrow i + \boxed{E} * E \Rightarrow i + i * \boxed{E} \Rightarrow i + i * i$$

Derivación más a la izquierda (2) $\rightrightarrows$ árbol (2), significado $(i + i) * i$:

$$E \Rightarrow \boxed{E} * E \Rightarrow \boxed{E} + E * E \Rightarrow i + \boxed{E} * E \Rightarrow i + i * \boxed{E} \Rightarrow i + i * i$$

La ambigüedad puede evitarse a veces

- La ambigüedad es una **característica no deseable** para una GLC, ya que esto supone que una misma sentencia puede tener **significados distintos**.

La ambigüedad puede evitarse a veces

- La ambigüedad es una **característica no deseable** para una GLC, ya que esto supone que una misma sentencia puede tener **significados distintos**.
- A las **gramáticas de lenguajes de programación** sí que se les puede **eliminar la ambigüedad.**

  Ej.- La siguiente gramática para expresiones aritméticas **no es ambigua** porque **tiene en cuenta la precedencia de operadores y asociatividad izquierda**:

$$
\begin{aligned}
&\langle \mathit{ExprArit} \rangle \to \langle \mathit{ExprArit} \rangle + \langle \mathit{Termino} \rangle \mid \langle \mathit{Termino} \rangle \\
&\langle \mathit{Termino} \rangle \to \langle \mathit{Termino} \rangle * \langle \mathit{Factor} \rangle \mid \langle \mathit{Factor} \rangle \\
&\langle \mathit{Factor} \rangle \to i \mid (\langle \mathit{ExprArit} \rangle)
\end{aligned}
$$

  La sentencia $i + i * i$ no es ambigua, porque tiene un único árbol.

La ambigüedad puede evitarse a veces

- La ambigüedad es una **característica no deseable** para una GLC, ya que esto supone que una misma sentencia puede tener **significados distintos**.
- A las **gramáticas de lenguajes de programación** sí que se les puede **eliminar la ambigüedad.**

  Ej.- La siguiente gramática para expresiones aritméticas **no es ambigua** porque **tiene en cuenta la precedencia de operadores y asociatividad izquierda**:

$$
\begin{aligned}
&\langle \mathit{ExprArit} \rangle \rightarrow \langle \mathit{ExprArit} \rangle + \langle \mathit{Termino} \rangle \mid \langle \mathit{Termino} \rangle \\
&\langle \mathit{Termino} \rangle \rightarrow \langle \mathit{Termino} \rangle * \langle \mathit{Factor} \rangle \mid \langle \mathit{Factor} \rangle \\
&\langle \mathit{Factor} \rangle \rightarrow i \mid (\langle \mathit{ExprArit} \rangle)
\end{aligned}
$$

  La sentencia $i + i * i$ no es ambigua, porque tiene un único árbol.
- Sin embargo, **el problema de la ambigüedad es indecidible: no existe algoritmo que compruebe si una gramática arbitraria es ambigua** $\rightrightarrows$ no existe algoritmo que elimine la ambigüedad.

Ambigüedad inevitable

- Existen lenguajes libres del contexto que **no pueden ser generados por gramáticas no ambiguas** $\Rightarrow$ son **inherentemente ambiguos**

  Ej.-$L_{amb} = \{a^n b^n c^m d^m \mid n, m \geq 1\} \cup \{a^i b^j c^j d^i \mid i, j \geq 1\}$.

Ambigüedad inevitable

- Existen lenguajes libres del contexto que **no pueden ser generados por gramáticas no ambiguas** $\Rightarrow$ son **inherentemente ambiguos**

  Ej.- $L_{amb} = \{a^n b^n c^m d^m \mid n, m \geq 1\} \cup \{a^i b^j c^j d^i \mid i, j \geq 1\}$.

- Una posible gramática para L_{amb}:

$$\begin{array}{ll} S \to AB \mid aCd & C \to aCd \mid bDc \mid bc \\ A \to aAb \mid ab & D \to bDc \mid bc \\ B \to cBd \mid cd & \end{array}$$

Ambigüedad inevitable

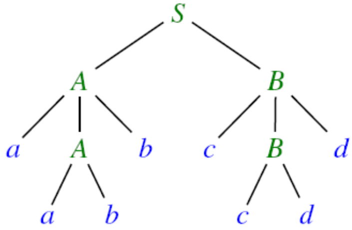
- Existen lenguajes libres del contexto que **no pueden ser generados por gramáticas no ambiguas** $\rightrightarrows$ son **inherentemente ambiguos**

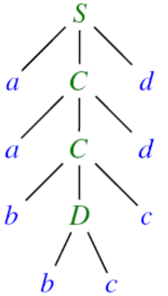
  Ej.-$L_{amb} = \{a^n b^n c^m d^m \mid n, m \geq 1\} \cup \{a^i b^j c^j d^i \mid i, j \geq 1\}$.

- Una posible gramática para L_{amb}:

$$
\begin{array}{ll}
S \rightarrow AB \mid aCd & C \rightarrow aCd \mid bDc \mid bc \\
A \rightarrow aAb \mid ab & D \rightarrow bDc \mid bc \\
B \rightarrow cBd \mid cd &
\end{array}
$$

- La sentencia **aabbccdd** es ambigua.





Combinación de GLCs

Para **obtener una gramática para un lenguaje complejo que se divide en lenguajes más simples**, a veces interesa obtener gramáticas para los lenguajes simples y **combinar las gramáticas** mediante métodos de:

- Unión
- Concatenación
- Clausura

Combinación de GLCs

Para **obtener una gramática para un lenguaje complejo que se divide en lenguajes más simples**, a veces interesa obtener gramáticas para los lenguajes simples y **combinar las gramáticas** mediante métodos de:

- Unión
- Concatenación
- Clausura

Teorema (propiedades de cierre) La unión, concatenación o clausura de lenguajes libres del contexto es un lenguaje libre del contexto $\Rightarrow$ la clase $\mathcal{L}_{LC}$ de lenguajes libres de contexto es cerrada bajo las operaciones de unión, concatenación o clausura.

Concatenación de gramáticas

Ej.- Queremos encontrar una GLC que genere el lenguaje

$$L_c = \{a^i b^i c^j d^j \mid i, j \geq 1\}$$

Concatenación de gramáticas

Ej.- Queremos encontrar una GLC que genere el lenguaje

$$L_c = \{a^i b^i c^j d^j \mid i, j \geq 1\}$$

- Tenemos que $L_c = L_1 \cdot L_2$, donde:

$$L_1 = \{a^n b^n \mid n \geq 1\} \quad \text{y} \quad L_2 = \{c^n d^n \mid n \geq 1\}$$

Concatenación de gramáticas

Ej.- Queremos encontrar una GLC que genere el lenguaje

$$L_c = \{a^i b^i c^j d^j \mid i, j \geq 1\}$$

- Tenemos que $L_c = L_1 \cdot L_2$, donde:

  $$L_1 = \{a^n b^n \mid n \geq 1\} \quad \text{y} \quad L_2 = \{c^n d^n \mid n \geq 1\}$$

- Obtenemos G_1 para L_1 y G_2 para L_2:

  $$G_1 : \; S_1 \rightarrow aS_1b \mid ab \qquad G_2 : \; S_2 \rightarrow cS_2d \mid cd$$

Concatenación de gramáticas

Ej.- Queremos encontrar una GLC que genere el lenguaje

$$L_c = \{a^i b^i c^j d^j \mid i,j \geq 1\}$$

- Tenemos que $L_c = L_1 \cdot L_2$, donde:

$$L_1 = \{a^n b^n \mid n \geq 1\} \quad \text{y} \quad L_2 = \{c^n d^n \mid n \geq 1\}$$

- Obtenemos G_1 para L_1 y G_2 para L_2:

$$G_1 : \; S_1 \to aS_1b \mid ab \qquad G_2 : \; S_2 \to cS_2d \mid cd$$

- Una gramática G_c que cumple $L(G_c) = L_1 \cdot L_2 = L_c$ es:

$$S \to S_1S_2$$
$$S_1 \to aS_1b \mid ab$$
$$S_2 \to cS_2d \mid cd$$

Unión de gramáticas

Ej.- Queremos encontrar una GLC que genere el lenguaje

$$L_u = \{w \in \{a, b, c, d\}^* \mid (w = a^n b^n \vee w = c^n d^n) \wedge n \geq 1\}$$

Unión de gramáticas

Ej.- Queremos encontrar una *GLC* que genere el lenguaje

$$L_u = \{w \in \{a, b, c, d\}^* \mid (w = a^n b^n \vee w = c^n d^n) \wedge n \geq 1\}$$

- Tenemos que $L_u = L_1 \cup L_2$, donde:

$$L_1 = \{a^n b^n \mid n \geq 1\} \quad \text{y} \quad L_2 = \{c^n d^n \mid n \geq 1\}$$

Unión de gramáticas

Ej.- Queremos encontrar una GLC que genere el lenguaje

$$L_u = \{w \in \{a, b, c, d\}^* \mid (w = a^n b^n \vee w = c^n d^n) \wedge n \geq 1\}$$

- Tenemos que $L_u = L_1 \cup L_2$, donde:

$$L_1 = \{a^n b^n \mid n \geq 1\} \quad \text{y} \quad L_2 = \{c^n d^n \mid n \geq 1\}$$

- Obtenemos G_1 para L_1 y G_2 para L_2:

$$G_1 : \ S_1 \rightarrow aS_1b \mid ab \qquad G_2 : \ S_2 \rightarrow cS_2d \mid cd$$

Unión de gramáticas

Ej.- Queremos encontrar una GLC que genere el lenguaje

$$L_u = \{w \in \{a, b, c, d\}^* \mid (w = a^n b^n \vee w = c^n d^n) \wedge n \geq 1\}$$

- Tenemos que $L_u = L_1 \cup L_2$, donde:

$$L_1 = \{a^n b^n \mid n \geq 1\} \quad \text{y} \quad L_2 = \{c^n d^n \mid n \geq 1\}$$

- Obtenemos G_1 para L_1 y G_2 para L_2:

$$G_1 : \; S_1 \to aS_1b \mid ab \qquad G_2 : \; S_2 \to cS_2d \mid cd$$

- Una gramática G_c que cumple $L(G_c) = L_1 \cup L_2 = L_c$ es:

$$S \to S_1 \mid S_2$$
$$S_1 \to aS_1b \mid ab$$
$$S_2 \to cS_2d \mid cd$$

Clausura de una gramática

Ej.- Queremos encontrar una GLC que genere el lenguaje L_u^*, donde:

$$L_u = \{w \in \{a, b, c, d\}^* \mid (w = a^n b^n \vee w = c^n d^n), n \geq 1\}$$

Clausura de una gramática

Ej.- Queremos encontrar una GLC que genere el lenguaje L_u^*, donde:

$$L_u = \{w \in \{a, b, c, d\}^* \mid (w = a^n b^n \vee w = c^n d^n), n \geq 1\}$$

- Obtenemos G_u para L_u (como antes):

$$\begin{aligned} S &\to S_1 \mid S_2 \\ S_1 &\to aS_1b \mid ab \\ S_2 &\to cS_2d \mid cd \end{aligned}$$

Clausura de una gramática

Ej.- Queremos encontrar una GLC que genere el lenguaje L_u^*, donde:

$$L_u = \{w \in \{a,b,c,d\}^* \mid (w = a^n b^n \vee w = c^n d^n), n \geq 1\}$$

- Obtenemos G_u para L_u (como antes):

$$\begin{array}{l} S \rightarrow S_1 \mid S_2 \\ S_1 \rightarrow aS_1b \mid ab \\ S_2 \rightarrow cS_2d \mid cd \end{array}$$

- Obtenemos G_s tal que $L(G_s) = L_u^*$:

$$\begin{array}{l} S' \rightarrow SS' \mid \lambda \\ S \rightarrow S_1 \mid S_2 \\ S_1 \rightarrow aS_1b \mid ab \\ S_2 \rightarrow cS_2d \mid cd \end{array}$$

Gramáticas regulares

Son un **tipo restringido de** GLC, que generan el mismo tipo de lenguajes que pueden ser descritos mediante expresiones regulares (los **lenguajes regulares**).

- Las **reglas de producción** son de la forma:

$$\boxed{\begin{array}{l} A \rightarrow bC \\ A \rightarrow b \\ A \rightarrow \lambda \end{array}}$$

Gramáticas regulares

Son un **tipo restringido de** GLC, que generan el mismo tipo de lenguajes que pueden ser descritos mediante expresiones regulares (los **lenguajes regulares**).

- Las **reglas de producción** son de la forma:

$$\boxed{\begin{array}{l} A \to bC \\ A \to b \\ A \to \lambda \end{array}}$$

- **Ej.-** Sea $N_{par} = \{xp \mid x \in V_{dig}^{*} \wedge p \in \{0, 2, 4, 6, 8\}\}$
  Una gramática regular que genera N_{npar} es:

Gramáticas regulares

Son un **tipo restringido de** GLC, que generan el mismo tipo de lenguajes que pueden ser descritos mediante expresiones regulares (los **lenguajes regulares**).

- Las **reglas de producción** son de la forma:

$$\boxed{\begin{array}{l} A \rightarrow bC \\ A \rightarrow b \\ A \rightarrow \lambda \end{array}}$$

- **Ej.-** Sea $N_{par} = \{xp \mid x \in V_{dig}^* \wedge p \in \{0, 2, 4, 6, 8\}\}$
  Una gramática regular que genera N_{npar} es:

$$S \rightarrow 0 \mid 2 \mid 4 \mid 6 \mid 8 \mid 0S \mid 1S \mid 2S \mid \cdots \mid 9S$$

Gramáticas regulares

Son un **tipo restringido de** GLC, que generan el mismo tipo de lenguajes que pueden ser descritos mediante expresiones regulares (los **lenguajes regulares**).

- Las **reglas de producción** son de la forma:

$$\boxed{\begin{array}{l} A \to bC \\ A \to b \\ A \to \lambda \end{array}}$$

- **Ej.-** Sea $N_{par} = \{xp \mid x \in V_{dig}^{*} \wedge p \in \{0, 2, 4, 6, 8\}\}$
  Una gramática regular que genera N_{npar} es:

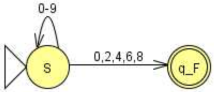
$$S \to 0 \mid 2 \mid 4 \mid 6 \mid 8 \mid 0S \mid 1S \mid 2S \mid \cdots \mid 9S$$

Teorema: todo lenguaje regular también es un lenguaje libre del contexto, pero no al contrario.

Relación entre gramáticas regulares y autómatas finitos

Se puede pasar de forma algorítmica de un autómata finito a una gramática regular y viceversa.

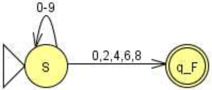
- **Método GRtoAF**.
  Ej.- de la gramática $S \rightarrow 0 \mid 2 \mid 4 \mid 6 \mid 8 \mid 0S \mid 1S \mid 2S \mid \cdots \mid 9S$ se obtiene el autómata:



Relación entre gramáticas regulares y autómatas finitos

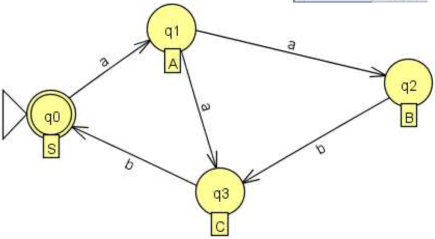
Se puede pasar de forma algorítmica de un autómata finito a una gramática regular y viceversa.

- **Método GRtoAF**.
  Ej.- de la gramática $S \to 0 \mid 2 \mid 4 \mid 6 \mid 8 \mid 0S \mid 1S \mid 2S \mid \cdots \mid 9S$ se obtiene el autómata:



- **Método AFtoGR**.

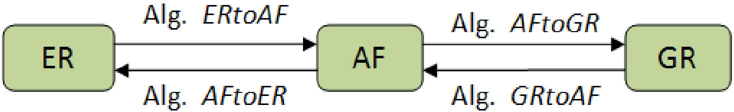
S	$\longrightarrow$	λ
A	$\longrightarrow$	aC
S	$\longrightarrow$	aA
A	$\longrightarrow$	aB
B	$\longrightarrow$	bC
C	$\longrightarrow$	bS



Gramáticas regulares, autómatas finitos y expresiones regulares

Equivalencia de formalismos para lenguajes regulares

Algoritmos de conversión



Algoritmos de transformación de GLC

A veces interesa encontrar una *GLC* equivalente a una dada ("transformar") para que ciertos **métodos de análisis sintáctico funcionen adecuadamente** y para **mejorar la eficiencia del proceso de análisis sintáctico**.

Estos algoritmos se aplican en Compiladores:

- Eliminación de símbolos inútiles.
- Transformación a gramática λ-libre.
- Eliminación de reglas unitarias.
- Transformación a gramática propia.

Eliminación de símbolos inútiles

- Una variable se dice que es **improductiva** si a partir de ella no se puede derivar una cadena de símbolos terminales. En otro caso es **productiva**.
- Un símbolo $X \in V_N \cup V_T$ (variable o terminal) se dice que es **inaccesible** si no aparece en ninguna forma sentencial de la gramática. En otro caso es **accesible**.
- Un símbolo es **inútil** si es una variable improductiva o es un símbolo inaccesible.

Eliminación de símbolos inútiles

- Una variable se dice que es **improductiva** si a partir de ella no se puede derivar una cadena de símbolos terminales. En otro caso es **productiva**.
- Un símbolo $X \in V_N \cup V_T$ (variable o terminal) se dice que es **inaccesible** si no aparece en ninguna forma sentencial de la gramática. En otro caso es **accesible**.
- Un símbolo es **inútil** si es una variable improductiva o es un símbolo inaccesible.

Algoritmo de eliminación de símbolos inútiles

ENTRADA: $G_{in} = (V_N, V_T, S, P)$

SALIDA: G_{out} equivalente y sin símbolos inútiles.

1. Eliminación de variables improductivas.
2. Eliminación de símbolos inaccesibles.

(El orden es importante: siempre 2 tras 1)

Eliminación de símbolos inútiles (1)

$\boxed{\text{Paso 1:}}$ **Eliminar variables improductivas.**

1.1 **INICIALIZAR** $Vpro$ **añadiendo las variables** A **de** V_N **para las que existe una regla A** $\rightarrow$ **w, tal que w** $\in V_T^*$**;**

Ejemplo:

$S \rightarrow aAS \mid AA$
$A \rightarrow AbB \mid ACa \mid a$
$B \rightarrow ABa \mid Ab \mid \lambda$
$C \rightarrow Cab \mid CC$
$D \rightarrow CD \mid Cb \mid e$
$E \rightarrow dA$

$Vpro =$

Eliminación de símbolos inútiles (1)

$\boxed{\text{Paso 1:}}$ **Eliminar variables improductivas.**

1.1 **INICIALIZAR** $Vpro$ **añadiendo las variables** A **de** V_N **para las que existe una regla** $\mathbf{A} \rightarrow \mathbf{w}$**, tal que w** $\in V_T^*$**;**

Ejemplo:

$S \rightarrow aAS \mid AA$
$\mathbf{A} \rightarrow AbB \mid ACa \mid \mathbf{a}$
$\mathbf{B} \rightarrow ABa \mid Ab \mid \lambda$
$C \rightarrow Cab \mid CC$
$\mathbf{D} \rightarrow CD \mid Cb \mid \mathbf{e}$
$E \rightarrow dA$

$$Vpro = \{\mathbf{A}, \mathbf{B}, \mathbf{D}\}$$

Eliminación de símbolos inútiles (1)

$\boxed{\text{Paso 1:}}$ **Eliminar variables improductivas.**

1.1 Inicializar *Vpro* añadiendo las variables A de V_N para las que existe una regla $A \to w$, tal que $w \in V_T^*$;

1.2 **REPETIR:**

Examinar cada regla B $\to \alpha$ de P y si todas las variables que aparecen en α están ya en *Vpro* entonces añadir B a *Vpro*;

HASTA que no se añadan variables nuevas a *Vpro*;

Ejemplo:

$\mathbf{S} \to aAS \mid \mathbf{AA}$
$A \to AbB \mid ACa \mid a$
$B \to ABa \mid Ab \mid \lambda$
$C \to Cab \mid CC$
$D \to CD \mid Cb \mid e$
$\mathbf{E} \to \mathbf{dA}$

$$Vpro = \{A, B, D, \mathbf{S}, \mathbf{E}\}$$

Eliminación de símbolos inútiles (1)

$\boxed{\text{Paso 1:}}$ **Eliminar variables improductivas.**

1.1 Inicializar *Vpro* añadiendo las variables A de V_N para las que existe una regla $A \to w$, tal que $w \in V_T^*$;

1.2 REPETIR:

Examinar cada regla $B \to \alpha$ de P y si todas las variables que aparecen en α están ya en *Vpro* entonces añadir B a *Vpro*;

HASTA que no se añadan variables nuevas a *Vpro*;

1.3 $V_N \leftarrow Vpro$ **(sólo variables productivas);**

Ejemplo:

$S \to aAS \mid AA$
$A \to AbB \mid ACa \mid a$
$B \to ABa \mid Ab \mid \lambda$
$C \to Cab \mid CC$
$D \to CD \mid Cb \mid e$
$E \to dA$

$V_N = \{\mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{D}, \mathbf{E}\}$
(C es improductiva)

Eliminación de símbolos inútiles (1)

$\boxed{\text{Paso 1:}}$ **Eliminar variables improductivas.**

1.1 INICIALIZAR *Vpro* añadiendo las variables A de V_N para las que existe una regla $A \to w$, tal que $w \in V_T^*$;

1.2 REPETIR:

Examinar cada regla $B \to \alpha$ de P y si todas las variables que aparecen en α están ya en *Vpro* entonces añadir B a *Vpro*;

HASTA que no se añadan variables nuevas a *Vpro*;

1.3 $V_N \leftarrow Vpro$ (sólo variables productivas);

1.4 **Eliminar de P aquellas reglas que tienen alguna variable improductiva (ya no pertenece a V_N) en la parte izda. o derecha;**

Ejemplo:

$S \to aAS \mid AA$
$A \to AbB \mid \cancel{ACa} \mid a$
$B \to ABa \mid Ab \mid \lambda$
$\cancel{C \to Cab \mid CC}$
$D \to \cancel{CD} \mid \cancel{Cb} \mid e$
$E \to dA$

$$V_N = \{S, A, B, D, E\}$$

Eliminación de símbolos inútiles (1)

$\boxed{\text{Paso 1:}}$ **Eliminar variables improductivas.**

1.1 INICIALIZAR *Vpro* añadiendo las variables A de V_N para las que existe una regla $A \to w$, tal que $w \in V_T^*$;

1.2 REPETIR:

Examinar cada regla $B \to \alpha$ de P y si todas las variables que aparecen en α están ya en *Vpro* entonces añadir B a *Vpro*;

HASTA que no se añadan variables nuevas a *Vpro*;

1.3 $V_N \leftarrow Vpro$ (sólo variables productivas);

1.4 **Eliminar de P aquellas reglas que tienen alguna variable improductiva (ya no pertenece a V_N) en la parte izda. o derecha;**

Ejemplo:

$S \to aAS \mid AA$
$A \to AbB \mid a$
$B \to ABa \mid Ab \mid \lambda$
$D \to e$
$E \to dA$

$$V_N = \{S, A, B, D, E\}$$

Gramática sin variables improductivas

Eliminación de símbolos inútiles (2)

$\boxed{\text{Paso 2:}}$ **Eliminar símbolos inaccesibles.**

2.1 **INICIALIZAR conjunto** $Vacc$ **de variables accesibles con el símbolo inicial** S**;**

Ejemplo:

$S \rightarrow aAS \mid AA$
$A \rightarrow AbB \mid a$
$B \rightarrow ABa \mid Ab \mid \lambda$
$D \rightarrow e$
$E \rightarrow dA$

$$Vacc = \{S\}$$

Eliminación de símbolos inútiles (2)

$\boxed{\text{Paso 2:}}$ **Eliminar símbolos inaccesibles.**

2.1 Inicializar conjunto *Vacc* de variables accesibles con el símbolo inicial S;

2.2 **REPETIR:**

Para cada variable nueva A en *Vacc*, examinar cada regla del tipo $\mathbf{A} \to \alpha$ en P y añadir a *Vacc* todas las variables que aparecen en α;

HASTA que no se añadan variables nuevas a *Vacc*;

Ejemplo:

$\mathbf{S} \to a\mathbf{A}S \mid \mathbf{AA}$
$A \to AbB \mid a$
$B \to ABa \mid Ab \mid \lambda$
$D \to e$
$E \to dA$

$Vacc = \{\mathbf{S}, \mathbf{A}\}$

Eliminación de símbolos inútiles (2)

$\boxed{\text{Paso 2:}}$ **Eliminar símbolos inaccesibles.**

2.1 Inicializar conjunto *Vacc* de variables accesibles con el símbolo inicial S;

2.2 **REPETIR:**

Para cada variable nueva A en *Vacc*, examinar cada regla del tipo $\mathbf{A} \rightarrow \alpha$ en P y añadir a *Vacc* todas las variables que aparecen en α;

HASTA que no se añadan variables nuevas a *Vacc*;

Ejemplo:

$S \rightarrow aAS \mid AA$
$\mathbf{A} \rightarrow Ab\mathbf{B} \mid a$
$B \rightarrow ABa \mid Ab \mid \lambda$
$D \rightarrow e$
$E \rightarrow dA$

$Vacc = \{S, \mathbf{A}, \mathbf{B}\}$

Eliminación de símbolos inútiles (2)

$\boxed{\text{Paso 2:}}$ **Eliminar símbolos inaccesibles.**

2.1 Inicializar conjunto *Vacc* de variables accesibles con el símbolo inicial S;

2.2 REPETIR:

Para cada variable nueva A en *Vacc*, examinar cada regla del tipo $A \rightarrow \alpha$ en P y añadir a *Vacc* todas las variables que aparecen en α;

HASTA que no se añadan variables nuevas a *Vacc*;

2.3 $V_N \leftarrow Vacc$ **(sólo variables accesibles);**

Ejemplo:

$S \rightarrow aAS \mid AA$
$A \rightarrow AbB \mid a$
$B \rightarrow ABa \mid Ab \mid \lambda$
$D \rightarrow e$
$E \rightarrow dA$

$\mathbf{V_N} = \{\mathbf{S}, \mathbf{A}, \mathbf{B}\}$
(D, E son variables inaccesibles)

Eliminación de símbolos inútiles (2)

$\boxed{\text{Paso 2:}}$ **Eliminar símbolos inaccesibles.**

2.1 INICIALIZAR conjunto *Vacc* de variables accesibles con el símbolo inicial S;

2.2 REPETIR:

Para cada variable nueva A en *Vacc*, examinar cada regla del tipo $A \rightarrow \alpha$ en P y añadir a *Vacc* todas las variables que aparecen en α;

HASTA que no se añadan variables nuevas a *Vacc*;

2.3 $V_N \leftarrow Vacc$ (se eliminan las inaccesibles);

2.4 **Eliminar de P aquellas reglas que tienen alguna variable inaccesible (ya no pertenece a V_N) en la parte izda. o derecha;**

2.5 **Eliminar de V_T todos aquellos símbolos terminales que no aparecen en las reglas que quedan en P (son terminales inaccesibles);**

Ejemplo:

$S \rightarrow aAS \mid AA$

$A \rightarrow AbBa$

$B \rightarrow ABa \mid Ab \mid \lambda$

$\cancel{D \rightarrow e}$

$\cancel{E \rightarrow dA}$

$V_N = \{S, A, B\}, V_T = \{a, b\}$

Eliminación de símbolos inútiles (2)

$\boxed{\text{Paso 2:}}$ **Eliminar símbolos inaccesibles.**

2.1 INICIALIZAR conjunto *Vacc* de variables accesibles con el símbolo inicial S;

2.2 REPETIR:

Para cada variable nueva A en *Vacc*, examinar cada regla del tipo $A \to \alpha$ en P y añadir a *Vacc* todas las variables que aparecen en α;

HASTA que no se añadan variables nuevas a *Vacc*;

2.3 $V_N \leftarrow Vacc$ (se eliminan las inaccesibles);

2.4 **Eliminar de P aquellas reglas que tienen alguna variable inaccesible (ya no pertenece a V_N) en la parte izda. o derecha;**

2.5 **Eliminar de V_T todos aquellos símbolos terminales que no aparecen en las reglas que quedan en P (son terminales inaccesibles);**

Ejemplo:

$S \to aAS \mid AA$
$A \to AbB \mid a$
$B \to ABa \mid Ab \mid \lambda$

$V_N = \{S, A, B\}, V_T = \{a, b\}$

Gramática sin símbolos inútiles

Transformación a gramática λ-libre

- Una GLC con símbolo inicial S es λ-LIBRE si no contiene reglas de la forma $A \to \lambda$ (λ-reglas), excepto $S \to \lambda$, siempre y cuando S no aparezca en la parte derecha de ninguna regla.

Transformación a gramática λ-libre

- Una *GLC* con símbolo inicial S es λ-LIBRE si no contiene reglas de la forma $A \to \lambda$ (λ-reglas), excepto $S \to \lambda$, siempre y cuando S no aparezca en la parte derecha de ninguna regla.
- **Teorema** para toda *GLC* existe otra *GLC* equivalente λ-libre.

Transformación a gramática λ-libre

- Una *GLC* con símbolo inicial S es λ-LIBRE si no contiene reglas de la forma $A \to \lambda$ (λ-reglas), excepto $S \to \lambda$, siempre y cuando S no aparezca en la parte derecha de ninguna regla.
- **Teorema** para toda *GLC* existe otra *GLC* equivalente λ-libre.

Algoritmo de transformación a gramática λ-libre

ENTRADA: $G_{in} = (V_N, V_T, S, P)$
SALIDA: G_{out} equivalente y λ-libre.

1. Obtener el conjunto de variables anulables.
2. Eliminar λ-reglas y añadir nuevas.
3. Añadir nuevo estado inicial y reglas adicionales si es necesario.

Transformación a gramática λ-libre (1)

$\boxed{\text{Paso 1:}}$ **Obtener conjunto de variables anulables.**
$Vanu = \{A \in V_N | A \Rightarrow^* \lambda\}$

1.1 **INICIALIZAR** $Vanu$ **con todas las variables** A **tal que** $A \rightarrow \lambda \in P$**;**

Ejemplo:

$S \rightarrow AB \mid 0S1$
$\mathbf{A} \rightarrow 0ABC \mid \lambda$
$\mathbf{B} \rightarrow B1 \mid \lambda$
$\mathbf{C} \rightarrow \lambda$

$Vanu = \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$

Transformación a gramática λ-libre (1)

$\boxed{\text{Paso 1:}}$ **Obtener conjunto de variables anulables.**
$Vanu = \{A \in V_N | A \Rightarrow^* \lambda\}$

1.1 INICIALIZAR *Vanu* con todas las variables A tal que $A \to \lambda \in P$;

1.2 **REPETIR:**

Para cada vble. $B \notin V_{anu}$ **y cada regla** $\mathbf{B \to C_1C_2 \dots C_n}$**, donde todas las variables** $\mathbf{C_i} \in Vanu$ **(son anulables) entonces se añade** B **a** $Vanu$**;**

HASTA que no se añadan variables nuevas a $Vanu$**;**

Ejemplo:

$\mathbf{S} \to \mathbf{AB} \mid 0S1$
$A \to 0ABC \mid \lambda$
$B \to B1 \mid \lambda$
$C \to \lambda$

$Vanu = \{\mathbf{S}, A, B, C\}$
(todas son anulables)

Transformación a gramática λ-libre (2)

$\boxed{\text{Paso 2:}}$ **Eliminar λ-reglas y añadir nuevas**

2.1 **Eliminar de P todas las reglas de la forma $A \to \lambda$**

Ejemplo:

$S \to AB \mid 0S1$
$\mathbf{A} \to 0ABC \mid \cancel{\lambda}$
$\mathbf{B} \to B1 \mid \cancel{\lambda}$
$\cancel{\mathbf{C} \to \lambda}$

$Vanu = \{\mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{C}\}$

Transformación a gramática λ-libre (2)

$\boxed{\text{Paso 2:}}$ **Eliminar λ-reglas y añadir nuevas**

2.1 <u>Eliminar de P</u> todas las reglas de la forma $A \to \lambda$;

2.2 **PARA CADA <u>regla con variables anulables</u> tipo $A \to \alpha$, se añaden todas las reglas (excepto $A \to \lambda$) que se generan al considerar que cada variable anulable de α se incluye en una regla y no se incluye en otra regla;**

Ejemplo:

$\mathbf{S} \to \mathbf{AB} \mid 0S1$
$A \to 0ABC$
$B \to B1$

$Vanu = \{\mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{C}\}$

$$\begin{cases} S \to AB \\ S \to A \\ S \to B \\ \cancel{S \to \lambda} \end{cases}$$

Transformación a gramática λ-libre (2)

$\boxed{\text{Paso 2:}}$ **Eliminar λ-reglas y añadir nuevas**

2.1 Eliminar de P todas las reglas de la forma $A \rightarrow \lambda$;

2.2 **PARA CADA regla con variables anulables tipo $A \rightarrow \alpha$, se añaden todas las reglas (excepto $A \rightarrow \lambda$) que se generan al considerar que cada variable anulable de α se incluye en una regla y no se incluye en otra regla;**

Ejemplo:

$\mathbf{S} \rightarrow AB \mid A \mid B \mid 0\mathbf{S}1$
$A \rightarrow 0ABC$
$B \rightarrow B1$

$Vanu = \{\mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{C}\}$

$$\begin{cases} S \rightarrow 0S1 \\ S \rightarrow 01 \end{cases}$$

Transformación a gramática λ-libre (2)

$\boxed{\text{Paso 2:}}$ **Eliminar λ-reglas y añadir nuevas**

2.1 Eliminar de P todas las reglas de la forma $A \to \lambda$;

2.2 **PARA CADA regla con variables anulables tipo $A \to \alpha$, se añaden todas las reglas (excepto $A \to \lambda$) que se generan al considerar que cada variable anulable de α se incluye en una regla y no se incluye en otra regla;**

Ejemplo:

$S \to AB \mid A \mid B \mid 0S1 \mid 01$
$\mathbf{A} \to 0\mathbf{ABC}$
$B \to B1$

$Vanu = \{\mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{C}\}$

$$\begin{cases} A \to 0ABC \\ A \to 0AB \mid 0AC \mid 0BC \\ A \to 0A \mid 0B \mid 0C \\ A \to 0 \end{cases}$$

Transformación a gramática λ-libre (2)

$\boxed{\text{Paso 2:}}$ **Eliminar λ-reglas y añadir nuevas**

2.1 Eliminar de P todas las reglas de la forma $A \rightarrow \lambda$;

2.2 **PARA CADA regla con variables anulables tipo $A \rightarrow \alpha$, se añaden todas las reglas (excepto $A \rightarrow \lambda$) que se generan al considerar que cada variable anulable de α se incluye en una regla y no se incluye en otra regla;**

Ejemplo:

$S \rightarrow AB \mid A \mid B \mid 0S1 \mid 01$
$A \rightarrow 0ABC \mid 0AB \mid 0AC \mid 0BC$
$A \rightarrow 0A \mid 0B \mid 0C \mid 0$
$\mathbf{B} \rightarrow \mathbf{B}1$

$Vanu = \{\mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{C}\}$

$$\begin{cases} B \rightarrow B1 \\ B \rightarrow 1 \end{cases}$$

Transformación a gramática λ-libre (2)

$\boxed{\text{Paso 2:}}$ **Eliminar λ-reglas y añadir nuevas**

2.1 Eliminar de P todas las reglas de la forma $A \rightarrow \lambda$;

2.2 **PARA CADA regla con variables anulables tipo $A \rightarrow \alpha$, se añaden todas las reglas (excepto $A \rightarrow \lambda$) que se generan al considerar que cada variable anulable de α se incluye en una regla y no se incluye en otra regla;**

Ejemplo:

$S \rightarrow AB \mid A \mid B \mid 0S1 \mid 01$
$A \rightarrow 0ABC \mid 0AB \mid 0AC \mid 0BC$
$A \rightarrow 0A \mid 0B \mid 0C \mid 0$
$\mathbf{B} \rightarrow B1 \mid 1$

$Vanu = \{\mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{C}\}$

Transformación a gramática λ-libre (3)

$\boxed{\text{Paso 3:}}$ **Añadir nuevo símbolo inicial y reglas adicionales <u>si es necesario</u>**

Si $S \in Vanu$ ($S \Rightarrow^* \lambda$ en la gramática de entrada) entonces:

Si S no aparece en la parte derecha entonces añadir regla $S \rightarrow \lambda$;

En otro caso

- **Considerar un nuevo símbolo inicial S' y añadir S' a V_N;**
- **Añadir las reglas $S' \rightarrow S|\lambda$ a P;**

Ejemplo:

$Vanu = \{\mathbf{S}, A, B, C\}$

$S \Rightarrow^* \lambda$ (S es anulable) y S aparece en la parte derecha.

$$S \rightarrow AB \mid 0S1 \mid A \mid B \mid 01$$
$$A \rightarrow 0ABC \mid 0AB \mid 0AC \mid 0BC \mid 0A \mid 0B \mid 0C \mid 0$$
$$B \rightarrow B1 \mid 1$$

Transformación a gramática λ-libre (3)

$\boxed{\text{Paso 3:}}$ **Añadir nuevo símbolo inicial y reglas adicionales <u>si es necesario</u>**

Si $S \in Vanu$ ($S \Rightarrow^* \lambda$ en la gramática de entrada) entonces:

Si S no aparece en la parte derecha entonces añadir regla $S \to \lambda$;

En otro caso

- **Considerar un nuevo símbolo inicial S' y añadir S' a V_N;**
- **Añadir las reglas $S' \to S|\lambda$ a P;**

Ejemplo:

<u>Nuevo símbolo inicial</u> S'

$$
\begin{array}{l}
S' \to S|\lambda \quad \text{(añadidas)} \\
S \to AB \mid 0S1 \mid A \mid B \mid 01 \\
A \to 0ABC \mid 0AB \mid 0AC \mid 0BC \mid 0A \mid 0B \mid 0C \mid 0 \\
B \to B1 \mid 1
\end{array}
$$

Gramática λ-libre equivalente. (C ha quedado improductiva)

Gramática sin reglas unitarias

Llamamos **regla unitaria** a una regla de la forma $A \to B$, con $A, B \in V_N$.

Gramática sin reglas unitarias

Llamamos **regla unitaria** a una regla de la forma $A \to B$, con $A, B \in V_N$.

Algoritmo de eliminación de reglas unitarias

ENTRADA: $G_{in} = (V_N, V_T, S, P)$
SALIDA: G_{out} equivalente a G_{in} sin reglas unitarias.

$\boxed{\text{Paso previo:}}$ obtener gramática λ-libre equivalente, si no lo es G_{in}.

1. Calcular conjunto de variables derivables de otra cualquiera por reglas unitarias.
2. Eliminar las reglas unitarias.
3. Añadir reglas para que el lenguaje generado no varíe.

Eliminación de reglas unitarias

$\boxed{\text{Paso previo:}}$ obtener gramática λ-libre equivalente, si no lo es la de entrada.

Ejemplo:

$S \to aBc \mid A \mid aAb$
$A \to B \mid cd$
$B \to ccBS \mid dc$

La gramática ya es λ-libre

Eliminación de reglas unitarias

$\boxed{\text{Paso previo:}}$ obtener gramática λ-libre equivalente, si no lo es la de entrada.

- **Para cada variable $\mathbf{A} \in V_N$ se calcula $Vuni(\mathbf{A}) = \{\mathbf{B} \in V_N \mid \mathbf{A} \Rightarrow^* \mathbf{B}, B \neq A\}$;**

Ejemplo:

$Vuni(\mathbf{S}) = \{\mathbf{A}, \mathbf{B}\}$

$S \rightarrow aBc \mid A \mid aAb$
$A \rightarrow B \mid cd$
$B \rightarrow ccBS \mid dc$

$\mathbf{S} \overbrace{\Rightarrow}^{S \rightarrow A} A \overbrace{\Rightarrow}^{A \rightarrow B} B$ luego $S \Rightarrow^* A, S \Rightarrow^* B$

Eliminación de reglas unitarias

$\boxed{\text{Paso previo:}}$ obtener gramática λ-libre equivalente, si no lo es la de entrada.

- **Para cada variable $\mathbf{A} \in V_N$ se calcula**
  $Vuni(\mathbf{A}) = \{\mathbf{B} \in V_N \mid \mathbf{A} \Rightarrow^* \mathbf{B}, B \neq A\}$**;**

Ejemplo:

$S \rightarrow aBc \mid A \mid aAb$
$A \rightarrow B \mid cd$
$B \rightarrow ccBS \mid dc$

$Vuni(S) = \{A, B\}$
$Vuni(\mathbf{A}) = \{\mathbf{B}\}$
Por la regla $A \rightarrow B$ se tiene $\mathbf{A} \Rightarrow^* \mathbf{B}$

Eliminación de reglas unitarias

$\boxed{\text{Paso previo:}}$ obtener gramática λ-libre equivalente, si no lo es la de entrada.

- **Para cada variable $\mathbf{A} \in V_N$ se calcula**
  $Vuni(\mathbf{A}) = \{\mathbf{B} \in V_N \mid \mathbf{A} \Rightarrow^* \mathbf{B}, B \neq A\}$**;**

Ejemplo:

$S \rightarrow aBc \mid A \mid aAb$
$A \rightarrow B \mid cd$
$B \rightarrow ccBS \mid dc$

$Vuni(S) = \{A, B\}$
$Vuni(A) = \{B\}$
$Vuni(\mathbf{B}) = \varnothing$

Eliminación de reglas unitarias

$\boxed{\text{Paso previo:}}$ obtener gramática λ-libre equivalente, si no lo es G_{in}.

1. Para cada variable $A \in V_N$ se calcula
   $Vuni(A) = \{B \in V_N \mid A \Rightarrow^* B, B \neq A\}$;
2. **Eliminar las reglas unitarias de P;**

Ejemplo:

$S \rightarrow aBc \mid \cancel{A} \mid aAb$
$A \rightarrow \cancel{B} \mid cd$
$B \rightarrow ccBS \mid dc$

Eliminación de reglas unitarias

$\boxed{\text{Paso previo:}}$ obtener gramática λ-libre equivalente, si no lo es G_{in}.

1. Para cada variable $A \in V_N$ se calcula
   $Vuni(A) = \{B \in V_N \mid A \Rightarrow^* B, B \neq A\}$;
2. Eliminar las reglas unitarias de P;
3. **Añadir nuevas reglas:**
   Para cada variable $\mathbf{A}$ tal que $Vuni(A) \neq \varnothing$
   Para cada variable $\mathbf{B} \in Vuni(A)$
   Para cada regla de la forma $B \rightarrow \beta$
   Añadir la regla $\mathbf{A} \rightarrow \beta$;

Ejemplo:

$S \rightarrow aBc \mid aAb$
$\mathbf{A} \rightarrow \mathbf{cd}$
$B \rightarrow ccBS \mid dc$

$Vuni(\mathbf{S}) = \{\mathbf{A}, B\}, Vuni(A) = \{B\}$
Se añade $\mathbf{S} \rightarrow \mathbf{cd}$

Eliminación de reglas unitarias

$\boxed{\text{Paso previo:}}$ obtener gramática λ-libre equivalente, si no lo es G_{in}.

1. Para cada variable $A \in V_N$ se calcula
$Vuni(A) = \{B \in V_N \mid A \Rightarrow^* B, B \neq A\}$;
2. Eliminar las reglas unitarias de P;
3. **Añadir nuevas reglas:**
Para cada variable $\mathbf{A}$ tal que $Vuni(A) \neq \varnothing$
  Para cada variable $\mathbf{B} \in Vuni(A)$
    Para cada regla de la forma $B \to \beta$
      Añadir la regla $\mathbf{A} \to \beta$;

Ejemplo:

$S \to aBc \mid aAb \mid cd$
$A \to cd$
$\mathbf{B \to ccBS \mid dc}$

$Vuni(\mathbf{S}) = \{A, \mathbf{B}\}, Vuni(A) = \{B\}$
Se añade $\mathbf{S} \to \mathbf{ccBS} \mid \mathbf{dc}$

Eliminación de reglas unitarias

$\boxed{\text{Paso previo:}}$ obtener gramática λ-libre equivalente, si no lo es G_{in}.

1. Para cada variable $A \in V_N$ se calcula
   $Vuni(A) = \{B \in V_N \mid A \Rightarrow^* B, B \neq A\}$;
2. Eliminar las reglas unitarias de P;
3. **Añadir nuevas reglas:**
   Para cada variable $\mathbf{A}$ tal que $Vuni(A) \neq \varnothing$
   Para cada variable $\mathbf{B} \in Vuni(A)$
   Para cada regla de la forma $B \to \beta$
   Añadir la regla $\mathbf{A} \to \beta$;

Ejemplo:

$S \to aBc \mid aAb \mid cd \mid ccBS \mid dc$
$A \to cd$
$\mathbf{B} \to \mathbf{ccBS} \mid \mathbf{dc}$

$Vuni(S) = \{A, B\}, Vuni(\mathbf{A}) = \{\mathbf{B}\}$
Se añade $\mathbf{A} \to \mathbf{ccBS} \mid \mathbf{dc}$

Eliminación de reglas unitarias

Paso previo: obtener gramática λ-libre equivalente, si no lo es G_{in}.

1. Para cada variable $A \in V_N$ se calcula
   $Vuni(A) = \{B \in V_N \mid A \Rightarrow^* B, B \neq A\}$;
2. Eliminar las reglas unitarias de P;
3. **Añadir nuevas reglas:**
   Para cada variable **A** tal que $Vuni(A) \neq \varnothing$
   Para cada variable **B** $\in Vuni(A)$
   Para cada regla de la forma $B \rightarrow \beta$
   Añadir la regla **A** $\rightarrow \beta$;

Ejemplo:

$S \rightarrow aBc \mid aAb \mid cd \mid ccBS \mid dc$
$A \rightarrow cd \mid ccBS \mid dc$
$B \rightarrow ccBS \mid dc$

Gramática equivalente sin reglas unitarias

Gramática propia

Una GLC es **libre de ciclos** si es imposible que se produzca una derivación de la forma $A \Rightarrow^* A$.
Una GLC es **propia** si no tiene símbolos inútiles, es λ-libre y libre de ciclos.

Gramática propia

Una GLC es **libre de ciclos** si es imposible que se produzca una derivación de la forma $A \Rightarrow^* A$.
Una GLC es **propia** si no tiene símbolos inútiles, es λ-libre y libre de ciclos.

Algoritmo de transformación a gramática propia

ENTRADA: $G = (V_N, V_T, S, P)$
SALIDA: gramática equivalente a G y propia.

1. Se aplica a G el algoritmo de **eliminación de símbolos inútiles** y se obtiene otra equivalente G_2.

Gramática propia

Una GLC es **libre de ciclos** si es imposible que se produzca una derivación de la forma $A \Rightarrow^* A$.
Una GLC es **propia** si no tiene símbolos inútiles, es λ-libre y libre de ciclos.

Algoritmo de transformación a gramática propia

ENTRADA: $G = (V_N, V_T, S, P)$
SALIDA: gramática equivalente a G y propia.

1. Se aplica a G el algoritmo de **eliminación de símbolos inútiles** y se obtiene otra equivalente G_2.
2. Se aplica a G_2 el algoritmo de **transformación a λ-libre** y se obtiene G_3.

Gramática propia

Una GLC es **libre de ciclos** si es imposible que se produzca una derivación de la forma $A \Rightarrow^* A$.
Una GLC es **propia** si no tiene símbolos inútiles, es λ-libre y libre de ciclos.

Algoritmo de transformación a gramática propia

ENTRADA: $G = (V_N, V_T, S, P)$
SALIDA: gramática equivalente a G y propia.

1. Se aplica a G el algoritmo de **eliminación de símbolos inútiles** y se obtiene otra equivalente G_2.
2. Se aplica a G_2 el algoritmo de **transformación a λ-libre** y se obtiene G_3.
3. Se aplica a G_3 el algoritmo de **eliminación de las reglas unitarias** (y con ello los ciclos) y se obtiene G_4.
4. Se aplica a G_4 el algoritmo de **eliminación de símbolos inútiles** y se obtiene G_5.
5. Devolver (G_5)

Gramática propia

- Observación: una gramática puede tener reglas unitarias y ser propia.
- **Ej.-** La gramática siguiente es propia:

$$
\begin{aligned}
&S \rightarrow aBc \mid A \mid aAb \\
&A \rightarrow B \mid cd \\
&B \rightarrow ccBS \mid dc
\end{aligned}
$$

aunque las reglas $S \rightarrow A$, $A \rightarrow B$ son unitarias.
Pero no se da: $S \Rightarrow^* S$, ni $A \Rightarrow^* A$, ni $B \Rightarrow^* B$ (no hay ciclos).

Gramática propia

- Observación: una gramática puede tener reglas unitarias y ser propia.
- **Ej.-** La gramática siguiente es propia:

$$
\begin{aligned}
S &\to aBc \mid A \mid aAb \\
A &\to B \mid cd \\
B &\to ccBS \mid dc
\end{aligned}
$$

  aunque las reglas $S \to A$, $A \to B$ son unitarias.
  Pero no se da: $S \Rightarrow^* S$, ni $A \Rightarrow^* A$, ni $B \Rightarrow^* B$ (no hay ciclos).
- **Ej.-** ver *tutorial5-GLC-jflap*

FIN TEMA 4