Teoría de Lenguajes

Gramáticas Libres de Contexto

DC-UBA

1er cuatrimestre 2024

Agenda de hoy

1. Definiciones
2. Ejercicios entre todos
3. Propiedades
4. Ejercicio más difícil

Parte I

Repaso y definiciones

Ejemplo motivador

Las gramáticas son un formalismo para expresar lenguajes con una notación recursiva. Recordemos informalmente cómo eran con un ejemplo

Ejemplo

Dar una gramática para el lenguaje L_0 denotado por la expresión regular a^+ (sobre el alfabeto $\Sigma = \{a\}$).
Ejemplos de cadenas:

Ejemplo motivador

Las gramáticas son un formalismo para expresar lenguajes con una notación recursiva. Recordemos informalmente cómo eran con un ejemplo

Ejemplo

Dar una gramática para el lenguaje L_0 denotado por la expresión regular a^+ (sobre el alfabeto $\Sigma = \{a\}$).
Ejemplos de cadenas: $a, aa, aaa, aaaa, aaaaa, \ldots$

Ejemplo motivador

¿Cómo podemos definir *recursivamente* las cadenas que pertenecen al lenguaje $L_0 = L(a^+)$?

Ejemplo motivador

¿Cómo podemos definir *recursivamente* las cadenas que pertenecen al lenguaje $L_0 = L(a^+)$?

- **Caso base**: $a \in L_0$.
- **Caso recursivo**: Si $\alpha \in L_0$, entonces $a\alpha \in L_0$.

Ejemplo motivador

¿Cómo podemos definir *recursivamente* las cadenas que pertenecen al lenguaje $L_0 = L(a^+)$?

- **Caso base**: $a \in L_0$.
- **Caso recursivo**: Si $\alpha \in L_0$, entonces $a\alpha \in L_0$.

¿Y cómo lo notamos formalmente con una gramática?

$$\begin{aligned} S &\to a && \text{(base)} \\ S &\to aS && \text{(recursivo)} \end{aligned}$$

Podemos generar la cadena $aaaa \in L_0$ de la siguiente forma,

$$\begin{aligned} S &\Rightarrow aS && (S \to aS) \\ &\Rightarrow aaS && (S \to aS) \\ &\Rightarrow aaaS && (S \to aS) \\ &\Rightarrow aaaa && (S \to a) \end{aligned}$$

Ejemplo motivador

¿Y si queremos generar a^* en lugar de a^+?
Para a^+ teníamos

$$S \to a$$
$$S \to aS$$

Ejemplo motivador

¿Y si queremos generar a^* en lugar de a^+?
Para a^+ teníamos

$$S \to a$$
$$S \to aS$$

Alcanza con cambiar el caso base,

$$S \to \lambda$$
$$S \to aS$$

Definición de gramática

Gramática

Formalmente, una gramática se describe con una 4-upla

$$G = \langle V_N, V_T, P, S \rangle$$

donde,

- V_N son los **símbolos no terminales**.
- V_T son los **símbolos terminales**, disjuntos de V_N.
- P son las **producciones**.
- $S \in V_N$ es el **símbolo distinguido** *(start)*.

Definición de gramática

Gramática

Formalmente, una gramática se describe con una 4-upla

$$G = \langle V_N, V_T, P, S \rangle$$

donde,

- V_N son los **símbolos no terminales**.
- V_T son los **símbolos terminales**, disjuntos de V_N.
- P son las **producciones**.
- $S \in V_N$ es el **símbolo distinguido** *(start)*.

Ejemplo

En el ejemplo de a^+, tenemos

$$G = \langle \{S\}, \{a\}, P, S \rangle$$
$$P : S \rightarrow aS \mid a$$

Tipos de gramáticas

Se clasifican según la forma de sus producciones, ya que esto determina su poder expresivo:

Regulares

$$P : A \rightarrow w \mid wB$$

donde $w \in V_T^*$ y $A, B \in V_N$

Tipos de gramáticas

Se clasifican según la forma de sus producciones, ya que esto determina su poder expresivo:

Regulares

$$P : A \rightarrow w \mid wB$$

donde $w \in V_T^*$ y $A, B \in V_N$

Libres de contexto

$$P : A \rightarrow \alpha$$

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

Tipos de gramáticas

Se clasifican según la forma de sus producciones, ya que esto determina su poder expresivo:

Regulares

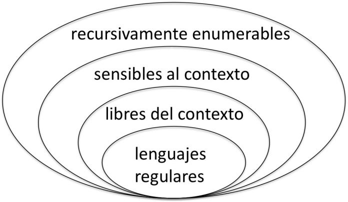
$$P : A \rightarrow w \mid wB$$

donde $w \in V_T^*$ y $A, B \in V_N$

Libres de contexto

$$P : A \rightarrow \alpha$$

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



Jerarquía de Chomsky

Tipos de gramáticas

Tipos de gramáticas

Regulares

$$P : A \rightarrow w \mid wB$$

donde $w \in V_T^*$ y $A, B \in V_N$

Libres de contexto

$$P : A \rightarrow \alpha$$

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

De las definiciones se desprende que toda gramática regular es libre de contexto, pero no al revés. **¿Por qué?**

Tipos de gramáticas

Regulares

$$P : A \rightarrow w \mid wB$$

donde $w \in V_T^*$ y $A, B \in V_N$

Libres de contexto

$$P : A \rightarrow \alpha$$

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

De las definiciones se desprende que toda gramática regular es libre de contexto, pero no al revés. **¿Por qué?**

Ejemplo

¿De qué tipo era la gramática del ejemplo?

$$G = \langle \{S\}, \{a\}, P, S \rangle$$
$$P : S \rightarrow aS \mid a$$

Tipos de gramáticas

Tipos de gramáticas

Regulares

$$P : A \rightarrow w \mid wB$$

donde $w \in V_T^*$ y $A, B \in V_N$

Libres de contexto

$$P : A \rightarrow \alpha$$

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

De las definiciones se desprende que toda gramática regular es libre de contexto, pero no al revés. **¿Por qué?**

Ejemplo

¿De qué tipo era la gramática del ejemplo? **Regular**

$$G = \langle \{S\}, \{a\}, P, S \rangle$$
$$P : S \rightarrow aS \mid a$$

Derivaciones

Para generar una cadena, vamos a partir del símbolo distinguido reemplazando símbolos no terminales que coincidan con la *cabeza* de alguna producción por su *cuerpo*. Para una producción $A \rightarrow \beta$, tenemos

$$\overbrace{A}^{\text{cabeza}} \rightarrow \overbrace{\beta}^{\text{cuerpo}}$$

Derivaciones

Para generar una cadena, vamos a partir del símbolo distinguido reemplazando símbolos no terminales que coincidan con la *cabeza* de alguna producción por su *cuerpo*. Para una producción $A \to \beta$, tenemos

$$\overbrace{A}^{\text{cabeza}} \to \overbrace{\beta}^{\text{cuerpo}}$$

Definimos formalmente el *reemplazo de símbolos según producciones* mediante la relación $\Rightarrow$.

Definición de la relación $\Rightarrow$ (derivación)

Sean $G = \langle V_N, V_T, P, S \rangle$ una GLC, $A \to \beta \in P$ y $\alpha, \beta, \gamma \in (V_N \cup V_T)^*$. Decimos que $\alpha A \gamma$ **deriva directamente** en $\alpha \beta \gamma$ y lo notamos como

$$\alpha A \gamma \underset{G}{\Rightarrow} \alpha \beta \gamma$$

Si está claro quien es G, podemos omitirlo y usar directamente $\Rightarrow$.

Lenguaje generado

- Una cadena $\alpha \in (V_N \cup V_T)^*$ es una **forma sentencial** si se deriva del símbolo distinguido ($S \overset{*}{\Rightarrow} \alpha$)

Lenguaje generado

- Una cadena $\alpha \in (V_N \cup V_T)^*$ es una **forma sentencial** si se deriva del símbolo distinguido ($S \overset{*}{\Rightarrow} \alpha$)
- El **lenguaje generado** por una gramática $G = \langle V_N, V_T, P, S \rangle$ son las cadenas de solo terminales a las que se puede llegar partiendo del símbolo distinguido (formas sentenciales de terminales). Es decir,

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

Donde $\overset{*}{\Rightarrow}$ es la *clausura de Kleene* de la relación $\Rightarrow$, que representa derivar en 0 o más pasos.

Lenguaje generado

- Una cadena $\alpha \in (V_N \cup V_T)^*$ es una **forma sentencial** si se deriva del símbolo distinguido ($S \stackrel{*}{\Rightarrow} \alpha$)
- El **lenguaje generado** por una gramática $G = \langle V_N, V_T, P, S \rangle$ son las cadenas de solo terminales a las que se puede llegar partiendo del símbolo distinguido (formas sentenciales de terminales). Es decir,

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

  Donde $\stackrel{*}{\Rightarrow}$ es la *clausura de Kleene* de la relación $\Rightarrow$, que representa derivar en 0 o más pasos.
- Los lenguajes generados por las GLCs son los *lenguajes libres de contexto*.

Lenguaje generado

- Una cadena $\alpha \in (V_N \cup V_T)^*$ es una **forma sentencial** si se deriva del símbolo distinguido ($S \overset{*}{\Rightarrow} \alpha$)
- El **lenguaje generado** por una gramática $G = \langle V_N, V_T, P, S \rangle$ son las cadenas de solo terminales a las que se puede llegar partiendo del símbolo distinguido (formas sentenciales de terminales). Es decir,

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

  Donde $\overset{*}{\Rightarrow}$ es la *clausura de Kleene* de la relación $\Rightarrow$, que representa derivar en 0 o más pasos.
- Los lenguajes generados por las GLCs son los *lenguajes libres de contexto*.
- Una observación clave es que las GLCs **generan** lenguajes, a diferencia de los formalismos que vimos anteriormente que los *reconocían* o *denotaban*.

Volviendo al ejemplo

Ejemplo completo $L_0 = L(a^+)$

$$G = \langle \{S\}, \{a\}, P, S \rangle$$
$$P : S \to aS \mid a$$

$$\begin{aligned} S &\Rightarrow aS & (S \to aS) \\ &\Rightarrow aaS & (S \to aS) \\ &\Rightarrow aaaS & (S \to aS) \\ &\Rightarrow aaaa & (S \to a) \end{aligned} \qquad \rightsquigarrow \qquad S \overset{*}{\Rightarrow} aaaa$$

$$L(G) = \{a, aa, aaa, aaaa, \ldots\} = L(a^+)$$

Por lo general,

- Usamos letras minúsculas (a, b, c), dígitos (0, 1, ..., 9) y otros caracteres ($+$, paréntesis, corchetes) para **terminales**

$$a \in V_T$$

Convenciones

Por lo general,

- Usamos letras minúsculas (a, b, c), dígitos (0, 1, ..., 9) y otros caracteres (+, paréntesis, corchetes) para **terminales**

$$a \in V_T$$

- Usamos letras mayúsculas (A, B, C) para **no terminales**

$$A \in V_N$$

Convenciones

Por lo general,

- Usamos letras minúsculas (a, b, c), dígitos (0, 1, ..., 9) y otros caracteres (+, paréntesis, corchetes) para **terminales**

$$a \in V_T$$

- Usamos letras mayúsculas (A, B, C) para **no terminales**

$$A \in V_N$$

- Usamos letras minúsculas del final del alfabeto (x, w, z) para cadenas de terminales

$$w \in V_T^*$$

Convenciones

Por lo general,

- Usamos letras minúsculas (a, b, c), dígitos (0, 1, ..., 9) y otros caracteres ($+$, paréntesis, corchetes) para **terminales**

$$a \in V_T$$

- Usamos letras mayúsculas (A, B, C) para **no terminales**

$$A \in V_N$$

- Usamos letras minúsculas del final del alfabeto (x, w, z) para cadenas de terminales

$$w \in V_T^*$$

- Usamos letras griegas minúsculas (α, β, γ) para cadenas compuestas de terminales y no terminales

$$\alpha \in (V_N \cup V_T)^*$$

Parte II

Ejercicios

Ejercicio 1

Sea $L_1 = \{a^n b^n \mid n \in \mathbb{N}_0\}$, dar una gramática libre de contexto para L_1. Ejemplos de cadenas:

- $\lambda, ab, aabb, aaabbb, aaaabbbb \in L$ y
- $a, b, aab, abb, bbaa, aabb\boldsymbol{ab} \notin L$

Ejercicio 1

Sea $L_1 = \{a^n b^n \mid n \in \mathbb{N}_0\}$, dar una gramática libre de contexto para L_1. Ejemplos de cadenas:

- $\lambda, ab, aabb, aaabbb, aaaabbbb \in L$ y
- $a, b, aab, abb, bbaa, aabb\boldsymbol{ab} \notin L$

$$G_1 = \langle \{S\}, \{a, b\}, P, S \rangle$$
$$P : S \rightarrow aSb \mid \lambda$$

Veamos, por ejemplo, que la cadena $aabb \in L(G_1)$:

Ejercicio 1

Sea $L_1 = \{a^n b^n \mid n \in \mathbb{N}_0\}$, dar una gramática libre de contexto para L_1. Ejemplos de cadenas:

- $\lambda, ab, aabb, aaabbb, aaaabbbb \in L$ y
- $a, b, aab, abb, bbaa, aabb\boldsymbol{ab} \notin L$

$$G_1 = \langle \{S\}, \{a, b\}, P, S \rangle$$
$$P : S \to aSb \mid \lambda$$

Veamos, por ejemplo, que la cadena $aabb \in L(G_1)$:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aa\lambda bb = aabb$$

Se puede demostrar formalmente que $L(G_1) = \{a^n b^n \mid n \in \mathbb{N}_0\}$ (con la doble inclusión), pero no vamos a hacerlo en la materia.

Ejercicio 2

Sea $L_2 = \{a^n b^m \mid n, m \in \mathbb{N}_0 \text{ y } n > m\}$, dar una GLC para L_2.

Ejemplos de cadenas:

Ejercicio 2

Sea $L_2 = \{a^n b^m \mid n, m \in \mathbb{N}_0 \text{ y } n > m\}$, dar una GLC para L_2.
Ejemplos de cadenas:

- $a, aab, aaa, aaab, aaabb, aaaab \in L$
- $\lambda, ab, b, abb, aab\boldsymbol{a} \notin L$

Ejercicio 2

Sea $L_2 = \{a^n b^m \mid n, m \in \mathbb{N}_0 \text{ y } n > m\}$, dar una GLC para L_2.
Ejemplos de cadenas:

- $a, aab, aaa, aaab, aaabb, aaaab \in L$
- $\lambda, ab, b, abb, aab\boldsymbol{a} \notin L$

$$
\begin{aligned}
G_2 &= \langle \{S, S', A\}, \{a, b\}, P, S \rangle \\
P &: \; S \to AS' \\
&\quad\; S' \to aS'b \mid \lambda \\
&\quad\; A \to aA \mid a
\end{aligned}
$$

$$
\begin{aligned}
G_2' &= \langle \{S\}, \{a, b\}, P, S \rangle \\
P &: \; S \to aSb \mid aS \mid a
\end{aligned}
$$

Veamos, por ejemplo, que la cadena $aaab \in L(G_2)$:

$$S \Rightarrow AS' \Rightarrow AaS'b \Rightarrow Aa\lambda b \Rightarrow aAab \Rightarrow aaabbb$$

Ejercicio 2 - Validando solución

Ejercicio 2

Sea $L_2 = \{a^n b^m \mid n, m \in \mathbb{N}_0 \text{ y } n > m\}$, dar una GLC para L_2.

Para convencernos de que la gramática genera el lenguaje que queremos, podemos pensar intuitivamente en el *lenguaje generado* por cada símbolo no terminal.

Ejercicio 2

Sea $L_2 = \{a^n b^m \mid n, m \in \mathbb{N}_0 \text{ y } n > m\}$, dar una GLC para L_2.

Para convencernos de que la gramática genera el lenguaje que queremos, podemos pensar intuitivamente en el *lenguaje generado* por cada símbolo no terminal.

$$
\begin{aligned}
G_2 &= \langle \{S, S', A\}, \{a, b\}, P, S \rangle \\
P &: \quad S \to AS' && (a^+ a^n b^n) \\
&\qquad S' \to aS'b \mid \lambda && (a^n b^n) \\
&\qquad A \to aA \mid a && (a^+)
\end{aligned}
$$

luego, podemos interpretar a^+ como $a^k, k > 0$. Con lo que

$$a^+ a^n b^n = a^k a^n b^n = a^{k+n} b^n = a^m b^n$$

y $m = n + k > n \iff k > 0$ ✓.

Ejercicio 2 - Casos de test

Usemos las cadenas que pertenecen o no al lenguaje como casos de test para darnos más confianza de que es correcta.

Cadena	¿Generada?
a	✓
aab	✓
aaa	✓
$aaab$	✓
$aaabb$	✓
λ	✗
ab	✗
b	✗
$abbb$	✗

$$
\begin{aligned}
G_2 &= \langle \{S, S', A\}, \{a, b\}, P, S \rangle \\
P \; &: \; S \rightarrow AS' \\
&\quad\; S' \rightarrow aS'b \mid \lambda \\
&\quad\; A \rightarrow aA \mid a
\end{aligned}
$$

Ejercicio 3

Sea $L_3 = \{a^n b^n c^m d^m \mid n, m \in \mathbb{N}_0\}$, dar una GLC para L_3.
Ejemplos de cadenas:

Ejercicio 3

Ejercicio 3

Sea $L_3 = \{a^n b^n c^m d^m \mid n, m \in \mathbb{N}_0\}$, dar una GLC para L_3.
Ejemplos de cadenas:

- $\lambda, abcd, aabbcd, ab, cd, aaabbbccdd \in L_3$
- $ad, bc, aabcd, aabbcdd, aabbcd\boldsymbol{ab} \notin L_3$

Ejercicio 3

Ejercicio 3

Sea $L_3 = \{a^n b^n c^m d^m \mid n, m \in \mathbb{N}_0\}$, dar una GLC para L_3.
Ejemplos de cadenas:

- $\lambda, abcd, aabbcd, ab, cd, aaabbbccdd \in L_3$
- $ad, bc, aabcd, aabbcdd, aabbcd\boldsymbol{ab} \notin L_3$

$$
\begin{aligned}
G_3 &= \langle \{S, N, M\}, \{a, b, c, d\}, P, S \rangle \\
P &: \; S \rightarrow NM \\
&\quad\;\, N \rightarrow aNb \mid \lambda \\
&\quad\;\, M \rightarrow cMd \mid \lambda
\end{aligned}
$$

Veamos, por ejemplo, que la cadena $aabbcd \in L(G_3)$:

Ejercicio 3

Sea $L_3 = \{a^n b^n c^m d^m \mid n, m \in \mathbb{N}_0\}$, dar una GLC para L_3.
Ejemplos de cadenas:

- $\lambda, abcd, aabbcd, ab, cd, aaabbbccdd \in L_3$
- $ad, bc, aabcd, aabbcdd, aabbcd\boldsymbol{ab} \notin L_3$

$$
\begin{aligned}
G_3 &= \langle \{S, N, M\}, \{a, b, c, d\}, P, S\rangle \\
P &: \ S \to NM \\
&\quad\ N \to aNb \mid \lambda \\
&\quad\ M \to cMd \mid \lambda
\end{aligned}
$$

Veamos, por ejemplo, que la cadena $aabbcd \in L(G_3)$:

$S \Rightarrow NM \Rightarrow aNbM \Rightarrow aaNbbM \Rightarrow aabbM \Rightarrow aabbcMd \Rightarrow aabbcd$

¿Hay otras derivaciones posibles?

$$S \to NM \quad N \to aNb \mid \lambda \quad M \to cMd \mid \lambda$$

Hay más de una derivación para la cadena $aabbcd$:

Reemplazando siempre el primer
no terminal de la **izquierda**:

$$S \underset{L}{\Rightarrow} NM \underset{L}{\Rightarrow} aNbM$$
$$\underset{L}{\Rightarrow} aaNbbM$$
$$\underset{L}{\Rightarrow} aa\lambda bbM$$
$$\underset{L}{\Rightarrow} aabbcMd$$
$$\underset{L}{\Rightarrow} aabbc\lambda d = aabbcd$$

Derivaciones L y R

$$S \to NM \quad N \to aNb \mid \lambda \quad M \to cMd \mid \lambda$$

Hay más de una derivación para la cadena $aabbcd$:

Reemplazando siempre el primer no terminal de la **izquierda**:

$$\begin{aligned} S &\underset{L}{\Rightarrow} NM \underset{L}{\Rightarrow} aNbM \\ &\underset{L}{\Rightarrow} aaNbbM \\ &\underset{L}{\Rightarrow} aa\lambda bbM \\ &\underset{L}{\Rightarrow} aabbcMd \\ &\underset{L}{\Rightarrow} aabbc\lambda d = aabbcd \end{aligned}$$

Reemplazando siempre el de la **derecha**:

$$\begin{aligned} S &\underset{R}{\Rightarrow} NM \underset{R}{\Rightarrow} NcMd \\ &\underset{R}{\Rightarrow} Nc\lambda d \\ &\underset{R}{\Rightarrow} aNbcd \\ &\underset{R}{\Rightarrow} aaNbbcd \\ &\underset{R}{\Rightarrow} aa\lambda bbcd = aabbcd \end{aligned}$$

Derivaciones L y R

$$S \to NM \quad N \to aNb \mid \lambda \quad M \to cMd \mid \lambda$$

Hay más de una derivación para la cadena *aabbcd*:

Reemplazando siempre el primer no terminal de la **izquierda**:

$$\begin{aligned} S &\underset{L}{\Rightarrow} NM \underset{L}{\Rightarrow} aNbM \\ &\underset{L}{\Rightarrow} aaNbbM \\ &\underset{L}{\Rightarrow} aa\lambda bbM \\ &\underset{L}{\Rightarrow} aabbcMd \\ &\underset{L}{\Rightarrow} aabbc\lambda d = aabbcd \end{aligned}$$

Reemplazando siempre el de la **derecha**:

$$\begin{aligned} S &\underset{R}{\Rightarrow} NM \underset{R}{\Rightarrow} NcMd \\ &\underset{R}{\Rightarrow} Nc\lambda d \\ &\underset{R}{\Rightarrow} aNbcd \\ &\underset{R}{\Rightarrow} aaNbbcd \\ &\underset{R}{\Rightarrow} aa\lambda bbcd = aabbcd \end{aligned}$$

Y otras más, por ejemplo:

$$\begin{aligned} S &\Rightarrow NM \Rightarrow NcMd \Rightarrow aNbcMd \\ &\Rightarrow aaNbbcMd \Rightarrow aaNbbc\lambda d \Rightarrow aab\lambda bcd \end{aligned}$$

Derivaciones L y R

Definición de derivaciones L y R

Sea $G = \langle V_N, V_T, P, S\rangle$ una gramática libre de contexto.

- Una **derivación más a la izquierda** es una derivación $wA\alpha \underset{L}{\Rightarrow} w\beta\alpha$ tal que
  - $A \rightarrow \beta \in P$,
  - $w \in V_T^*,\ \alpha \in (V_T \cup V_N)^*$,
  - A es el primer símbolo no terminal desde la izquierda.

Definición de derivaciones L y R

Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto.

- Una **derivación más a la izquierda** es una derivación $wA\alpha \underset{L}{\Rightarrow} w\beta\alpha$ tal que
  - $A \rightarrow \beta \in P$,
  - $w \in V_T^*, \ \alpha \in (V_T \cup V_N)^*$,
  - A es el primer símbolo no terminal desde la izquierda.
- Una **derivación más a la derecha** es una derivación $\alpha Aw \underset{R}{\Rightarrow} \alpha\beta w$ tal que
  - $A \rightarrow \beta \in P$,
  - $w \in V_T^*, \ \alpha \in (V_T \cup V_N)^*$,
  - A es el primer símbolo no terminal desde la derecha.

Árboles de derivación

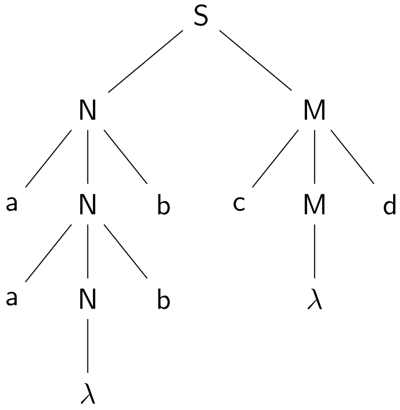
Volviendo al ejemplo, ambas derivaciones se pueden representar con un único **árbol** que *abstrae el orden* en el cuál se aplicaron.

- $S \underset{L}{\Rightarrow} NM \underset{L}{\Rightarrow} aNbM \underset{L}{\Rightarrow} aaNbbM \underset{L}{\Rightarrow} aabbM \overset{2}{\underset{L}{\Rightarrow}} aabbcd$
- $S \underset{R}{\Rightarrow} NM \underset{R}{\Rightarrow} NcMd \underset{R}{\Rightarrow} Ncd \underset{R}{\Rightarrow} aNbcd \underset{R}{\Rightarrow} aaNbbcd \underset{R}{\Rightarrow} aabbcd$

Árboles de derivación

Volviendo al ejemplo, ambas derivaciones se pueden representar con un único **árbol** que *abstrae el orden* en el cuál se aplicaron.

- $S \underset{L}{\Rightarrow} NM \underset{L}{\Rightarrow} aNbM \underset{L}{\Rightarrow} aaNbbM \underset{L}{\Rightarrow} aabbM \underset{L}{\overset{2}{\Rightarrow}} aabbcd$
- $S \underset{R}{\Rightarrow} NM \underset{R}{\Rightarrow} NcMd \underset{R}{\Rightarrow} Ncd \underset{R}{\Rightarrow} aNbcd \underset{R}{\Rightarrow} aaNbbcd \underset{R}{\Rightarrow} aabbcd$

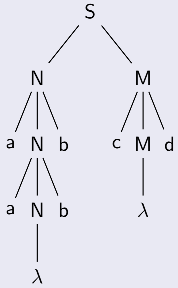


$$
\begin{aligned}
G_3 &= \langle \{S, N, M\}, \{a, b\}, P, S \rangle \\
P &: \quad S \rightarrow NM \\
& \quad N \rightarrow aNb \mid \lambda \\
& \quad M \rightarrow cMd \mid \lambda
\end{aligned}
$$

Definición de árbol de derivación

Un árbol de derivación de una gramática $G = \langle V_N, V_T, P, S \rangle$ es aquel que cumple con las siguientes condiciones.

- La raíz es el símbolo distinguido S.
- Cada nodo interno es un no terminal.
- Cada hoja es o un terminal, o λ.
- Si un nodo interno es A y sus hijos $X_1, X_2, \dots, X_k$ entonces $A \rightarrow X_1 X_2 \dots X_k \in P$.
- Si un vértice es λ, entonces es una hoja y es el único hijo de su padre.

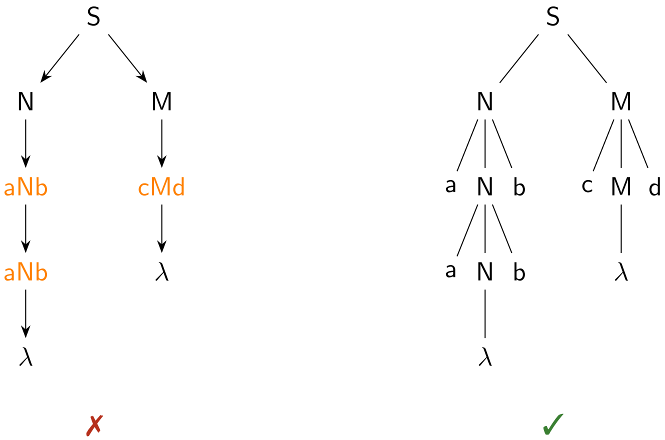


Hay una correspondencia uno a uno entre árboles de derivación, derivaciones más a la izquierda y derivaciones más a la derecha.

Errores comunes en árboles de derivación

¡Ojo! Errores comunes

- Los dibujamos como ejes y no arcos dirigidos
- Hay un nodo por símbolo



¿Para qué sirven las GLCs?

- Las gramáticas libres de contexto se suelen usar en los compiladores para describir la *sintaxis* de **lenguajes de programación**.

¿Para qué sirven las GLCs?

- Las gramáticas libres de contexto se suelen usar en los compiladores para describir la *sintaxis* de **lenguajes de programación**.
- Los árboles de derivación son usados para representar la *estructura* de los programas.

¿Para qué sirven las GLCs?

- Las gramáticas libres de contexto se suelen usar en los compiladores para describir la *sintaxis* de **lenguajes de programación**.
- Los árboles de derivación son usados para representar la *estructura* de los programas.
- Existe una forma automática de generar a partir de una descripción de una gramática un *parser*, el componente del compilador que descubre la estructura del programa y verifica que sea sintácticamente correcto.

¿Para qué sirven las GLCs?

- Las gramáticas libres de contexto se suelen usar en los compiladores para describir la *sintaxis* de **lenguajes de programación**.
- Los árboles de derivación son usados para representar la *estructura* de los programas.
- Existe una forma automática de generar a partir de una descripción de una gramática un *parser*, el componente del compilador que descubre la estructura del programa y verifica que sea sintácticamente correcto.
- La gramática tiene que expresar algo, los árboles que genera tienen que capturar la **estructura** de la cadena.

¿Para qué sirven las GLCs?

- Las gramáticas libres de contexto se suelen usar en los compiladores para describir la *sintaxis* de **lenguajes de programación**.
- Los árboles de derivación son usados para representar la *estructura* de los programas.
- Existe una forma automática de generar a partir de una descripción de una gramática un *parser*, el componente del compilador que descubre la estructura del programa y verifica que sea sintácticamente correcto.
- La gramática tiene que expresar algo, los árboles que genera tienen que capturar la **estructura** de la cadena.
- Nos vamos a enfocar en esto en esta segunda mitad de la materia.

Ejercicio 4

Sea L_4 el lenguaje de expresiones aritméticas con variables sobre el alfabeto $\{id, (,), +, \times\}$, donde los paréntesis son opcionales. Dar una GLC que lo genere.

Ejemplos de cadenas que pertenecen a L_4:

- id
- $id + id$, $id \times id$
- $id + id \times id$, $(id + id) \times (id + id)$

Ejercicio 4

Sea L_4 el lenguaje de expresiones aritméticas con variables sobre el alfabeto $\{id, (,), +, \times\}$, donde los paréntesis son opcionales. Dar una GLC que lo genere.

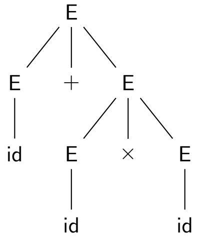
Ejemplos de cadenas que pertenecen a L_4:

- id
- $id + id$, $id \times id$
- $id + id \times id$, $(id + id) \times (id + id)$

$$
\begin{aligned}
G_4 &= \langle \{E\}, \{id, (,), +, \times\}, P, E \rangle \\
P : E &\rightarrow E + E \\
&\mid\ E \times E \\
&\mid\ (E) \\
&\mid\ id
\end{aligned}
$$

$$G_4 = \langle \{E\}, \{id, (,), +, \times\}, P, E\rangle$$
$$P \; : \; E \rightarrow E + E \mid E \times E \mid (E) \mid id$$

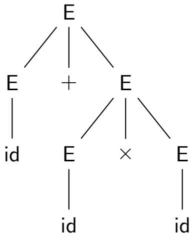
Veamos una derivación para $id + id \times id$.

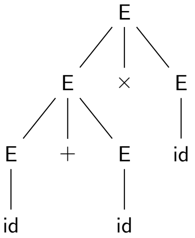


Ejercicio 4

$$G_4 = \langle \{E\}, \{id, (,), +, \times\}, P, E \rangle$$
$$P : E \rightarrow E + E \mid E \times E \mid (E) \mid id$$

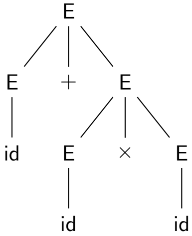
Veamos una derivación para $id + id \times id$.



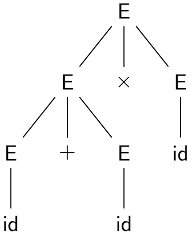


$$G_4 = \langle \{E\}, \{id, (,), +, \times\}, P, E \rangle$$
$$P : E \to E + E \mid E \times E \mid (E) \mid id$$

Veamos una derivación para $id + id \times id$.



$id + (id \times id)$

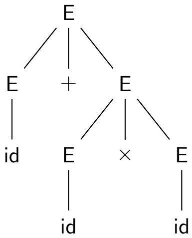


$(id + id) \times id$

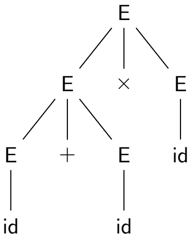
Ejercicio 4

$$G_4 = \langle \{E\}, \{id, (,), +, \times\}, P, E \rangle$$
$$P \ : \ E \rightarrow E + E \mid E \times E \mid (E) \mid id$$

Veamos una derivación para $id + id \times id$. ¡Hay más de una! Y una sola representa la precedencia usual



$id + (id \times id)$

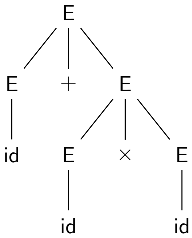


$(id + id) \times id$

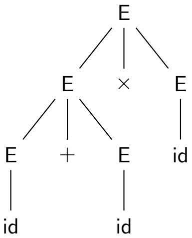
Ejercicio 4

$$G_4 = \langle \{E\}, \{id, (,), +, \times\}, P, E \rangle$$
$$P \;:\; E \rightarrow E + E \mid E \times E \mid (E) \mid id$$

Veamos una derivación para $id + id \times id$. ¡Hay más de una! Y una sola representa la precedencia usual



$id + (id \times id)$



$(id + id) \times id$

Ambigüedad

Gramática ambigua

Decimos que una gramática libre de contexto G es **ambigua** si existe $\alpha \in L(G)$ para la cuál hay más de un árbol de derivación distinto (o, equivalentemente, más de una derivación L / R).

¿Cómo probamos que es una gramática ambigua? Basta con exhibir dos árboles de derivación distintos para *alguna* cadena.

Gramática ambigua

Decimos que una gramática libre de contexto G es **ambigua** si existe $\alpha \in L(G)$ para la cuál hay más de un árbol de derivación distinto (o, equivalentemente, más de una derivación L / R).

¿Cómo probamos que es una gramática ambigua? Basta con exhibir dos árboles de derivación distintos para *alguna* cadena.

¡G_4 es ambigua! Mostramos que tiene dos árboles de derivación diferentes para la cadena $id + id \times id$.

¿Por qué es un problema?

- Como dijimos antes, para el proceso de *parsing* los árboles de derivación representarán la **estructura de la cadena**.
- A partir de ellos vamos a querer hacer cómputos.
- Si fueramos a reemplazar los identificadores por números, dependiendo de la estructura que nos dé la gramática **obtendríamos resultados diferentes**, lo cual no es deseable para un lenguaje de programación.

¿Cómo desambiguamos G_4?

$$\begin{aligned} G_4 &= \langle \{E\}, \{id, (,), +, \times\}, P, E\rangle \\ P &: \quad E \to E + E \mid E \times E \mid (E) \mid id \end{aligned}$$

Problema: No define ni la **precedencia** ni la **asociatividad**.
Las siguientes cadenas generan ambigüedad:

- $id + id + id$ y $id \times id \times id$ (asociatividad)
- $id + id \times id$ y $id \times id + id$ (precedencia)

Repaso precedencia y asociatividad

¿Qué era la precedencia?

Nos dice cuales operadores se aplican primero. Por ejemplo, $\times$ tiene mayor precedencia que $+$, por lo que $1 + 2 \times 3 = 1 + (2 \times 3)$.

¿Qué era la asociatividad?

Nos dice en qué dirección asocian. Usualmente todos los operadores aritméticos asocian a izquierda, por lo que

$$1 + 2 + 3 = (1 + 2) + 3.$$

Para $+$ y $\times$ no cambia el resultado ya que ambas operaciones son asociativas (aunque generen ambiguedad). Pero si tuvieramos la resta,

$$1 - 3 - 2 = (1 - 3) - 2 = -4 \neq 1 - (3 - 2) = 0$$

Desambiguación

Queremos que la gramática nos de árboles que cumplan con las siguientes precedencias y asociatividades, ordenadas de menos a más precedentes.

Operador	**Asociatividad**
$+$	Izquierda
$\times$	Izquierda
$(\cdot)$, **id**	-

- Queremos restringir la gramática para forzarlo.
- Intuitivamente, queremos que los operadores con menor precedencia aparezcan "más arriba" en el árbol de derivación (para que se evalúen más tarde) y las de mayor precedencia "más abajo" (para que se evalúen primero). Los paréntesis reinician la precedencia.
- Para la asociatividad a la izquierda, queremos permitir que dentro del mismo nivel de precedencia se "expandan" solamente hacia la izquierda.

Precedencia

- El problema es la producción $E \to E \times E$, que nos permite tener el $\times$ más arriba y dentro expandir alguna de las E por $E \to E + E$.
- Para evitarlo, jerarquizamos la gramática mediante no terminales. Primero generamos los $+$ (expresiones), luego los $\times$ (términos) y finalmente los identificadores o paréntesis (factores)

$$
\begin{aligned}
E &\to E + E \mid T && \text{(expresiones)} \\
T &\to T \times T \mid F && \text{(términos)} \\
F &\to (E) \mid id && \text{(factores)}
\end{aligned}
$$

Queda como ejercicio verificar que de esta forma no tenemos problemas con la precedencia (cadenas $id + id \times id$ y $id \times id + id$)

Asociatividad

Tenemos

$$\begin{aligned} E &\to E + E \mid T && \text{(expresiones)} \\ T &\to T \times T \mid F && \text{(términos)} \\ F &\to (E) \mid id && \text{(factores)} \end{aligned}$$

Asociatividad

Tenemos

$$\begin{aligned} E &\to E + E \mid T & \quad & \text{(expresiones)} \\ T &\to T \times T \mid F & & \text{(términos)} \\ F &\to (E) \mid id & & \text{(factores)} \end{aligned}$$

- Pero nos sigue generando ambigüedades con la asociatividad, por las producciones $E \to E + E$ para la suma y $T \to T \times T$ para la multiplicación.
- Nos quedamos con las que asocian a izquierda, $E \to E + T$ y $T \to T \times F$.

Gramática desambiguada

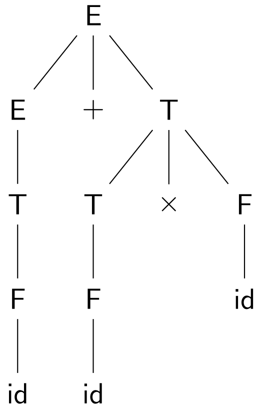
La gramática final queda

$$
\begin{aligned}
G_4' &= \langle \{E, T, F\}, \{id, (,), +, \times\}, P, E \rangle \\
P \;:\; & E \to E + T \mid T \qquad \text{(expresiones)} \\
& T \to T \times F \mid F \qquad \text{(términos)} \\
& F \to (E) \mid id \qquad \text{(factores)}
\end{aligned}
$$

Seguimos necesitando las producciones $E \to T$ y $T \to F$ para permitir expresiones sin $+$ y sin $\times$ respectivamente.

Desambiguación

Derivación de $id + id \times id$



$$
\begin{aligned}
G_4' &= \langle \{E, T, F\}, \{id, (,), +, \times\}, P, E \rangle \\
P &: \quad E \rightarrow E + T \mid T \\
& \qquad T \rightarrow T \times F \mid F \\
& \qquad F \rightarrow (E) \mid id
\end{aligned}
$$

Ambigüedad intrínseca

Definición de ambigüedad intrínseca

Decimos que un *lenguaje* libre de contexto (¡no gramática!) es **intrínsecamente ambiguo** si toda gramática que lo genera es ambigua.

Consecuencia

¡No siempre vamos a poder desambiguar una gramática y mantener lenguaje generado!

Ambigüedad intrínseca

Definición de ambigüedad intrínseca

Decimos que un *lenguaje* libre de contexto (¡no gramática!) es **intrínsecamente ambiguo** si toda gramática que lo genera es ambigua.

Consecuencia

¡No siempre vamos a poder desambiguar una gramática y mantener lenguaje generado!

Ejemplo

El siguiente lenguaje es intrínsecamente ambiguo

$$L = \{a^n b^n c^m d^m \mid n, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n, m \geq 1\}$$

Ejercicio 5

Dar una gramática libre de contexto para el lenguaje de cadenas de paréntesis balanceados.
Ejemplos de cadenas:
$\lambda, (), ()(), (()), ()(()), (())() \in L_5$

Ejercicio 5

Dar una gramática libre de contexto para el lenguaje de cadenas de paréntesis balanceados.
Ejemplos de cadenas:
$\lambda, (), ()(), (()), ()(()), (())() \in L_5$

$$
\begin{aligned}
G_5 &= \langle \{S\}, \{(,)\}, P, S \rangle \\
P &: \; S \rightarrow (S) \mid SS \mid \lambda
\end{aligned}
$$

Ejercicio 5

Dar una gramática libre de contexto para el lenguaje de cadenas de paréntesis balanceados.
Ejemplos de cadenas:
$\lambda, (), ()(), (()), ()(()), (())() \in L_5$

$$
\begin{aligned}
G_5 &= \langle \{S\}, \{(,)\}, P, S \rangle \\
P &: \; S \rightarrow (S) \mid SS \mid \lambda
\end{aligned}
$$

- ¿Es una gramática ambigua?

Ejercicio 5

Ejercicio 5

Dar una gramática libre de contexto para el lenguaje de cadenas de paréntesis balanceados.
Ejemplos de cadenas:
$\lambda, (), ()(), (()), ()(()), (())() \in L_5$

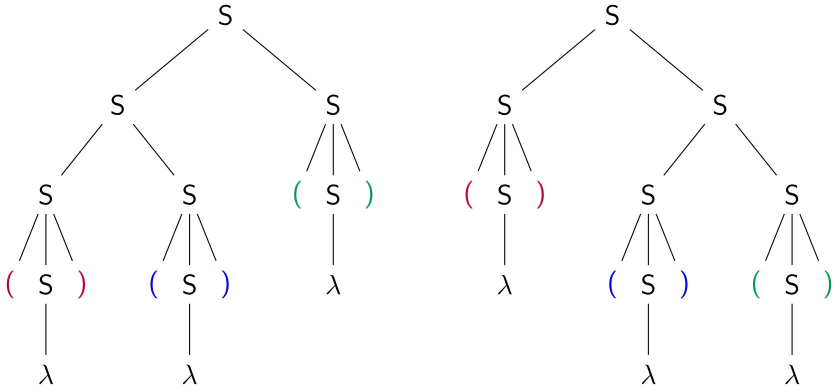
$$\begin{aligned} G_5 &= \langle \{S\}, \{(,)\}, P, S\rangle \\ P &: \ S \rightarrow (S) \mid SS \mid \lambda \end{aligned}$$

- ¿Es una gramática ambigua? **Si**, hay más de un árbol de derivación para la cadena ()()().
- Y también para la cadena λ,
  - $S \underset{L}{\Rightarrow} \lambda$
  - $S \underset{L}{\Rightarrow} SS \underset{L}{\Rightarrow} \lambda S \underset{L}{\Rightarrow} \lambda\lambda = \lambda$

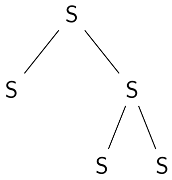
Ejercicio 5

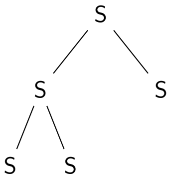
$$G_5 = \langle \{S\}, \{(,)\}, P, S \rangle$$
$$P \;:\; S \to (S) \mid SS \mid \lambda$$

Árboles de derivación para ()()()



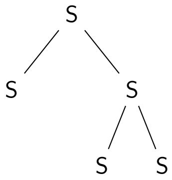
¡La producción que genera la ambigüedad es $S \to SS$! Se puede aplicar a cualquiera de los dos subárboles,

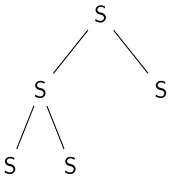




Ejercicio 5

¡La producción que genera la ambigüedad es $S \rightarrow SS$! Se puede aplicar a cualquiera de los dos subárboles,





¿Será **intrínsecamente ambiguo**?

Desambiguación de paréntesis balanceados

- Podemos pensar que hay dos operadores, el unario $(\cdot)$ y la concatenación.
- $S \rightarrow SS$ nos genera una ambigüedad de asociatividad en la concatenación (que no cambia en nada el resultado final ni la estructura de forma significativa)
- Para solucionarlo podemos separar en dos niveles, y forzar solo asociatividad a derecha

$$\begin{aligned} S &\rightarrow TS \mid \lambda \\ T &\rightarrow (S) \end{aligned}$$

- Como T no es más que un renombre podemos quitarlo

Concluimos que no es intrínsecamente ambiguo, ya que la siguiente gramática no ambigua genera el lenguaje.

$$\begin{aligned} G'_5 &= \langle \{S\}, \{(,)\}, P, S \rangle \\ P &: \quad S \rightarrow (S)S \mid \lambda \end{aligned}$$

Propiedades

Sean $L_1 = L(\langle V_{N_1}, V_{T_1}, P_1, S_1 \rangle)$, $L_2 = L(\langle V_{N_2}, V_{T_2}, P_2, S_2 \rangle)$ lenguajes libres de contexto.

- $L_1 \cup L_2$

Propiedades

Sean $L_1 = L(\langle V_{N_1}, V_{T_1}, P_1, S_1\rangle)$, $L_2 = L(\langle V_{N_2}, V_{T_2}, P_2, S_2\rangle)$ lenguajes libres de contexto.

- $L_1 \cup L_2$ es libre de contexto (cerrados por unión)
  Dem: Generado por (Con $S \notin (V_{N_1} \cup V_{N_2})$)
  $G = \langle V_{N_1} \cup V_{N_2} \cup \{S\}, V_{T_1} \cup V_{T_2}, P_1 \cup P_2 \cup \{S \rightarrow S_1 \mid S_2\}, S\rangle$
- $L_1 L_2$

Propiedades

Sean $L_1 = L(\langle V_{N_1}, V_{T_1}, P_1, S_1 \rangle)$, $L_2 = L(\langle V_{N_2}, V_{T_2}, P_2, S_2 \rangle)$ lenguajes libres de contexto.

- $L_1 \cup L_2$ es libre de contexto (cerrados por unión)
  Dem: Generado por (Con $S \notin (V_{N_1} \cup V_{N_2})$)
  $G = \langle V_{N_1} \cup V_{N_2} \cup \{S\}, V_{T_1} \cup V_{T_2}, P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}, S \rangle$
- $L_1 L_2$ es libre de contexto (cerrados por concatenación)
  Dem: Generado por (Idem S)
  $G = \langle V_{N_1} \cup V_{N_2} \cup \{S\}, V_{T_1} \cup V_{T_2}, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S \rangle$
- $L_1 \cap L_2$

Propiedades

Sean $L_1 = L(\langle V_{N_1}, V_{T_1}, P_1, S_1\rangle)$, $L_2 = L(\langle V_{N_2}, V_{T_2}, P_2, S_2\rangle)$ lenguajes libres de contexto.

- $L_1 \cup L_2$ es libre de contexto (cerrados por unión)
  Dem: Generado por (Con $S \notin (V_{N_1} \cup V_{N_2})$)
  $G = \langle V_{N_1} \cup V_{N_2} \cup \{S\}, V_{T_1} \cup V_{T_2}, P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}, S\rangle$
- $L_1 L_2$ es libre de contexto (cerrados por concatenación)
  Dem: Generado por (Idem S)
  $G = \langle V_{N_1} \cup V_{N_2} \cup \{S\}, V_{T_1} \cup V_{T_2}, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S\rangle$
- $L_1 \cap L_2$ no necesariamente es libre de contexto.
  Contraejemplo: $a^n b^n c^m \cap a^m b^n c^n = a^n b^n c^n$ no es libre de contexto (se puede ver con pumping).
- L_1^+

Propiedades

Sean $L_1 = L(\langle V_{N_1}, V_{T_1}, P_1, S_1\rangle)$, $L_2 = L(\langle V_{N_2}, V_{T_2}, P_2, S_2\rangle)$ lenguajes libres de contexto.

- $L_1 \cup L_2$ es libre de contexto (cerrados por unión)
  Dem: Generado por (Con $S \notin (V_{N_1} \cup V_{N_2})$)
  $G = \langle V_{N_1} \cup V_{N_2} \cup \{S\}, V_{T_1} \cup V_{T_2}, P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}, S\rangle$
- $L_1 L_2$ es libre de contexto (cerrados por concatenación)
  Dem: Generado por (Idem S)
  $G = \langle V_{N_1} \cup V_{N_2} \cup \{S\}, V_{T_1} \cup V_{T_2}, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S\rangle$
- $L_1 \cap L_2$ no necesariamente es libre de contexto.
  Contraejemplo: $a^n b^n c^m \cap a^m b^n c^n = a^n b^n c^n$ no es libre de contexto (se puede ver con pumping).
- L_1^+ es libre de contexto (cerrados por clausura positiva)
  Dem: Generado por (Idem S)
  $G = \langle V_{N_1} \cup V_{N_2} \cup \{S\}, V_{T_1} \cup V_{T_2}, P_1 \cup P_2 \cup \{S \to S_1 S \mid S_1\}, S\rangle$

Parte III

Ejercicio más complicado

Ejercicio más complicado

Ejercicio

Dar una GLC para $L = \{0^n 1^{2m} \mid n \neq m\}$

Ejemplos de cadenas que pertenecen al lenguaje:

Ejercicio más complicado

Ejercicio

Dar una GLC para $L = \{0^n 1^{2m} \mid n \neq m\}$

Ejemplos de cadenas que pertenecen al lenguaje:

- $0, 00, 0^+, (11)^+, 0011, 00011, 0001111 \in L$
- $\lambda, 011, 0111, 0011111 \notin L$

Ejercicio más complicado

Ejercicio

Dar una GLC para $L = \{0^n 1^{2m} \mid n \neq m\}$

Ejemplos de cadenas que pertenecen al lenguaje:

- $0, 00, 0^+, (11)^+, 0011, 00011, 0001111 \in L$
- $\lambda, 011, 0111, 0011111 \notin L$

Obs: $\{0^n 1^{2m} \mid n \neq m\} = \{0^n (11)^m \mid n \neq m\}$

Ejercicio más complicado

Ejercicio

Dar una GLC para $L = \{0^n 1^{2m} \mid n \neq m\}$

Ejemplos de cadenas que pertenecen al lenguaje:

- $0, 00, 0^+, (11)^+, 0011, 00011, 0001111 \in L$
- $\lambda, 011, 0111, 0011111 \notin L$

Obs: $\{0^n 1^{2m} \mid n \neq m\} = \{0^n (11)^m \mid n \neq m\}$
Pista: Es parecido a $L_2 = \{a^n b^m \mid n > m\}$

Ejercicio más complicado

Ejercicio

Dar una GLC para $L = \{0^n 1^{2m} \mid n \neq m\}$

Ejemplos de cadenas que pertenecen al lenguaje:

- $0, 00, 0^+, (11)^+, 0011, 00011, 0001111 \in L$
- $\lambda, 011, 0111, 0011111 \notin L$

Obs: $\{0^n 1^{2m} \mid n \neq m\} = \{0^n (11)^m \mid n \neq m\}$
Pista: Es parecido a $L_2 = \{a^n b^m \mid n > m\}$

$$\begin{aligned} G &= \langle \{S, B, Z, U\}, \{0,1\}, P, S\rangle \\ P &: \; S \to ZB \mid BU \\ &\quad\; B \to 0B11 \mid \lambda \\ &\quad\; Z \to 0Z \mid 0 \\ &\quad\; U \to 11U \mid 11 \end{aligned}$$

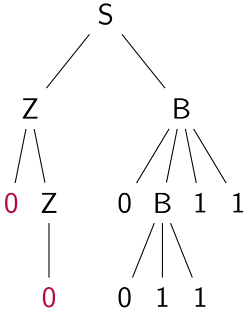
$$\begin{aligned} G' &= \langle \{S, Z, U\}, \{0,1\}, P', S\rangle \\ P' &: \; S \to 0S11 \mid Z \mid U \\ &\quad\; Z \to 0Z \mid 0 \\ &\quad\; U \to 11U \mid 11 \end{aligned}$$

2c2018 1p - Ej 4

2c2018 1p - Ej 4

Dar una GLC para $L = \{0^n 1^{2m} \mid n \neq m\}$

Veamos por ejemplo la derivación de 00001111



$$
\begin{aligned}
G &= \langle \{S, B, Z, U\}, \{0,1\}, P, S \rangle \\
P &: \; S \rightarrow ZB \mid BU \\
&\quad\; B \rightarrow 0B11 \mid \lambda \\
&\quad\; Z \rightarrow 0Z \mid 0 \\
&\quad\; U \rightarrow 11U \mid 11
\end{aligned}
$$

Solo como curiosidades. Sean G_1, G_2 gramáticas libres de contexto y L_1 y L_2 los lenguajes que generan.

- $w \stackrel{?}{\in} L_1$ es computable (*parsing*, 2da mitad de la materia).

Computabilidad

Solo como curiosidades. Sean G_1, G_2 gramáticas libres de contexto y L_1 y L_2 los lenguajes que generan.

- $w \stackrel{?}{\in} L_1$ es computable (*parsing*, 2da mitad de la materia).
- $L_1 \stackrel{?}{=} \emptyset$ es computable.

Solo como curiosidades. Sean G_1, G_2 gramáticas libres de contexto y L_1 y L_2 los lenguajes que generan.

- $w \overset{?}{\in} L_1$ es computable (*parsing*, 2da mitad de la materia).
- $L_1 \overset{?}{=} \emptyset$ es computable.
- ¿Es G_1 ambigua? es **indecidible**.
  Más adelante vamos a ver una condición suficiente (pero no necesaria) para que no lo sea que *sí* es computable.

Computabilidad

Solo como curiosidades. Sean G_1, G_2 gramáticas libres de contexto y L_1 y L_2 los lenguajes que generan.

- $w \overset{?}{\in} L_1$ es computable (*parsing*, 2da mitad de la materia).
- $L_1 \overset{?}{=} \emptyset$ es computable.
- ¿Es G_1 ambigua? es **indecidible**.
  Más adelante vamos a ver una condición suficiente (pero no necesaria) para que no lo sea que *sí* es computable.
- ¿Es L_1 intrínsecamente ambiguo?

Computabilidad

Solo como curiosidades. Sean G_1, G_2 gramáticas libres de contexto y L_1 y L_2 los lenguajes que generan.

- $w \stackrel{?}{\in} L_1$ es computable (*parsing*, 2da mitad de la materia).
- $L_1 \stackrel{?}{=} \emptyset$ es computable.
- ¿Es G_1 ambigua? es **indecidible**.
  Más adelante vamos a ver una condición suficiente (pero no necesaria) para que no lo sea que *sí* es computable.
- ¿Es L_1 intrínsecamente ambiguo? es **indecidible**.
- $L_1 \cap L_2 \stackrel{?}{=} \emptyset$ es **indecidible**.

Computabilidad

Solo como curiosidades. Sean G_1, G_2 gramáticas libres de contexto y L_1 y L_2 los lenguajes que generan.

- $w \overset{?}{\in} L_1$ es computable (*parsing*, 2da mitad de la materia).
- $L_1 \overset{?}{=} \emptyset$ es computable.
- ¿Es G_1 ambigua? es **indecidible**.
  Más adelante vamos a ver una condición suficiente (pero no necesaria) para que no lo sea que *sí* es computable.
- ¿Es L_1 intrínsecamente ambiguo? es **indecidible**.
- $L_1 \cap L_2 \overset{?}{=} \emptyset$ es **indecidible**.
- $L_1 \overset{?}{=} L_2$ es **indecidible**.

Solo como curiosidades. Sean G_1, G_2 gramáticas libres de contexto y L_1 y L_2 los lenguajes que generan.

- $w \stackrel{?}{\in} L_1$ es computable (*parsing*, 2da mitad de la materia).
- $L_1 \stackrel{?}{=} \emptyset$ es computable.
- ¿Es G_1 ambigua? es **indecidible**.
  Más adelante vamos a ver una condición suficiente (pero no necesaria) para que no lo sea que *sí* es computable.
- ¿Es L_1 intrínsecamente ambiguo? es **indecidible**.
- $L_1 \cap L_2 \stackrel{?}{=} \emptyset$ es **indecidible**.
- $L_1 \stackrel{?}{=} L_2$ es **indecidible**.
- $L_1 \stackrel{?}{=} \Sigma^*$ es **indecidible**.

Conclusiones

- Repasamos qué eran las gramáticas, hicimos algunos ejercicios sencillos y luego otros no tan sencillos.

Conclusiones

- Repasamos qué eran las gramáticas, hicimos algunos ejercicios sencillos y luego otros no tan sencillos.
- Vimos que suele ser útil ver **ejemplos** para entender la pinta del lenguaje, y luego usarlos como **tests** para validar que no generemos cadenas de más y que no quede ninguna afuera. Es una técnica súmamente útil sobre todo para los lenguajes más complicados.

Conclusiones

- Repasamos qué eran las gramáticas, hicimos algunos ejercicios sencillos y luego otros no tan sencillos.
- Vimos que suele ser útil ver **ejemplos** para entender la pinta del lenguaje, y luego usarlos como **tests** para validar que no generemos cadenas de más y que no quede ninguna afuera. Es una técnica súmamente útil sobre todo para los lenguajes más complicados.
- Además, vimos una alternativa para convencerse de que el lenguaje generado es correcto en algunas gramáticas, pensando en el **lenguaje generado por cada no terminal**.

Conclusiones

- Repasamos qué eran las gramáticas, hicimos algunos ejercicios sencillos y luego otros no tan sencillos.
- Vimos que suele ser útil ver **ejemplos** para entender la pinta del lenguaje, y luego usarlos como **tests** para validar que no generemos cadenas de más y que no quede ninguna afuera. Es una técnica súmamente útil sobre todo para los lenguajes más complicados.
- Además, vimos una alternativa para convencerse de que el lenguaje generado es correcto en algunas gramáticas, pensando en el **lenguaje generado por cada no terminal**.
- En la misma línea, ayuda pensar que cada símbolo no terminal tiene un **rol** (según el lenguaje que genera), y ponerle un nombre acorde.

Conclusiones

- Repasamos qué eran las gramáticas, hicimos algunos ejercicios sencillos y luego otros no tan sencillos.
- Vimos que suele ser útil ver **ejemplos** para entender la pinta del lenguaje, y luego usarlos como **tests** para validar que no generemos cadenas de más y que no quede ninguna afuera. Es una técnica súmamente útil sobre todo para los lenguajes más complicados.
- Además, vimos una alternativa para convencerse de que el lenguaje generado es correcto en algunas gramáticas, pensando en el **lenguaje generado por cada no terminal**.
- En la misma línea, ayuda pensar que cada símbolo no terminal tiene un **rol** (según el lenguaje que genera), y ponerle un nombre acorde.
- ¡Eso es todo! Ya pueden hacer la guía 7 entera.