

# AUTÓMATAS Y LENGUAJES

LENGUAJES LIBRES DEL CONTEXTO - GRAMÁTICAS LIBRES DEL CONTEXTO - AÑO 2024

LIC. EN CS. DE LA COMPUTACIÓN - 4to. AÑO

# Objetivos de la Unidad

*Lenguajes Libres del Contexto (LLC).*

- *Estudiar los dispositivos descriptores de los LLC:*
  - *Gramáticas Libres del Contexto (GLC): árboles de derivación, BNF, BNFE, ambigüedad.*
  - *Autómatas Push-Down (APD): no determinísticos y determinísticos.*
  - *Análisis Sintáctico: técnicas top down y bottom-up.*
- *Comprender la teoría de gramáticas para conocer su relación con los Lenguajes de programación.*

# Intuitivamente



$$L_1 = \{a^n b^n / n \geq 0\} \quad L_2 = \{w \in \{0,1\}^* / ww^R\}$$

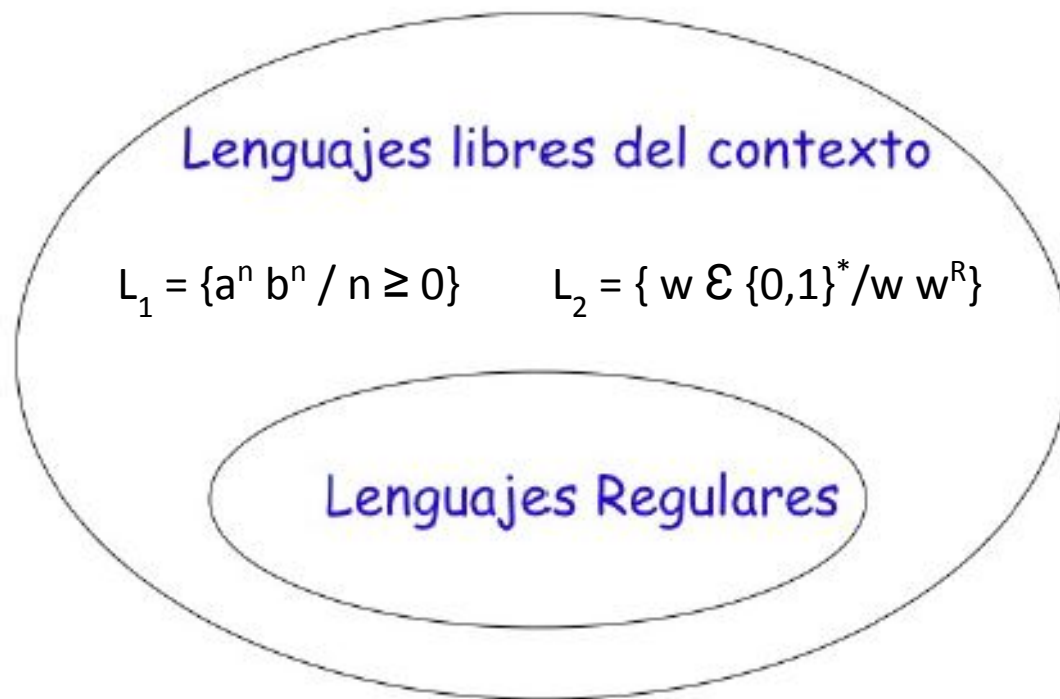
*¿Qué podemos decir  
sobre estos lenguajes?*

Lenguajes Regulares

$a^*b^*$

$(a+b)^*$

# Intuitivamente



# Lenguajes Libres del Contexto (LLC)

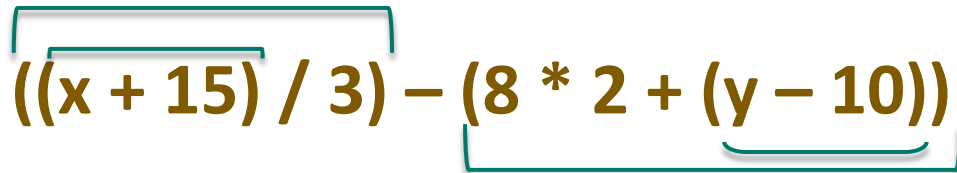
## Características:

- ▶ Estos lenguajes pueden ser generados por **gramáticas libres del contexto**, la cual es una *notación recursiva natural* para este tipo de lenguajes.
- ▶ La principal característica de este tipo de lenguajes es que las cadenas contienen símbolos en modalidad espejo, aunque también se engloban las características propias de los lenguajes Tipo 3 (recordar  $L_3 \subset L_2$ ).

# Lenguajes Libres del Contexto (LLC)

*En los siguientes ejemplos, propios de los lenguajes de programación y de la matemática, podemos apreciar esta característica de anidamientos, sincronización o modalidad espejo:*

- Uso de paréntesis en expresiones aritméticas:

$$((x + 15) / 3) - (8 * 2 + (y - 10))$$


- Uso de paréntesis y corchetes en expresiones tipo C:

$h(f[i] * (arr[i][j], c[g(x)]), d[i])$



# Lenguajes Libres del Contexto

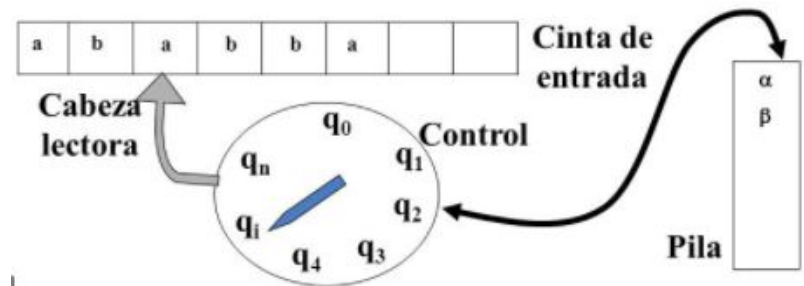
(LLC)

Gramáticas Libres del Contexto (GLC)

Autómatas Push-Down (APD)

$$G = (N, \Sigma, P, S)$$

Conjunto de símbolos  
No Terminales  
Conjunto de símbolos  
Terminales o alfabeto  
Conjunto de producciones  
Símbolo de comienzo  
o distinguido



# Historia de las gramáticas



Ejemplo: el lenguaje inglés

$\langle sentence \rangle \rightarrow \langle noun\_phrase \rangle \langle predicate \rangle$

$\langle noun\_phrase \rangle \rightarrow \langle article \rangle \langle noun \rangle$

$\langle predicate \rangle \rightarrow \langle verb \rangle$

$\langle article \rangle \rightarrow a$

$\langle article \rangle \rightarrow the$

$\langle noun \rangle \rightarrow cat$

$\langle noun \rangle \rightarrow dog$

$\langle verb \rangle \rightarrow runs$

$\langle verb \rangle \rightarrow walks$

$\langle sentence \rangle \Rightarrow \langle noun\_phrase \rangle \langle predicate \rangle$

$\Rightarrow \langle noun\_phrase \rangle \langle verb \rangle$

$\Rightarrow \langle article \rangle \langle noun \rangle \langle verb \rangle$

$\Rightarrow the \langle noun \rangle \langle verb \rangle$

$\Rightarrow the \text{ dog } \langle verb \rangle$

$\Rightarrow the \text{ dog } walks$



“the dog walk”



# Gramáticas Libres del Contexto (GLC)

**Definición:** Una gramática  $G = (N, \Sigma, P, S)$  es libre de contexto (GLC), si sus producciones en  $P$  tienen la siguiente forma:  $A \rightarrow \alpha$ , con  $A \in N$  y  $\alpha \in (N \cup \Sigma)^*$ .

El nombre “*libre de contexto*” se debe a que cada una de las producciones pueden ser aplicadas independientemente del contexto en donde aparezca un no terminal en una *forma sentencial*.

Su importancia radica en que permiten describir los **aspectos sintácticos** de los lenguajes de programación. Por lo tanto tienen un rol central en el contexto de compiladores.

***Las GLC también son llamadas Gramáticas Tipo 2.***

# Gramáticas Libres del Contexto (GLC)

- ★ Se utilizan las letras mayúsculas para expresar los símbolos no terminales.
- ★ Para los símbolos del alfabeto  $\Sigma$  se usan las letras minúsculas, números, símbolos en general que pueden ser delimitadores, símbolos de puntuación, etc.
- ★ El símbolo distinguido forma parte del conjunto de no terminales.

# Gramáticas Libres del Contexto (GLC)

Gramática:  $S \rightarrow aSb$    $L = \{a^n b^n / n \geq 0\}$   
 $S \rightarrow \lambda$

Derivación de la sentencia:  $ab$

$$\begin{array}{ccc} & S \Rightarrow aSb \Rightarrow ab & \\ \nearrow & & \nwarrow \\ S \rightarrow aSb & & S \rightarrow \lambda \end{array}$$

**Recordar:** el símbolo  $\rightarrow$  se utiliza para expresar las reglas, en forma práctica y se lee produce.

**Recordar:** El símbolo  $\Rightarrow$  se lee deriva. Representa la relación deriva, que permite derivar cadenas a partir del símbolo distinguido y aplicando las reglas de la GLC.

## Relación deriva (Repaso)

- ▶ Sean  $\alpha, \beta, \delta, \gamma \in (\Sigma \cup N)^*$ , definimos la relación deriva ( $\Rightarrow$ ) sobre  $(\Sigma \cup N)^*$  como sigue:  
 $\delta\alpha\gamma \Rightarrow \delta\beta\gamma$  *sii*  $\exists \alpha \rightarrow \beta \in P$   
Esto significa que  $\delta\beta\gamma$  es obtenida a partir de  $\delta\alpha\gamma$  por la aplicación de la regla o producción  $\alpha \rightarrow \beta \in P$
- ▶ Si  $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow \alpha_n$  con  $n \geq 0$  y  $\alpha_i \in (N \cup \Sigma)^*$  para  $i \in \{0, 1, \dots, n\}$ , luego decimos que  $\alpha_0 \xRightarrow{*} \alpha_n$ , es decir la clausura reflexo- transitiva o que  $\alpha_0$  deriva en 0 o más
- ▶ Si  $\alpha_n \in (\Sigma \cup N)^*$ , luego  $\alpha_n$  es denominado una *forma sentencial*.
- ▶ Si  $\alpha_n \in \Sigma^*$ , es una *sentencia* del lenguaje.

## Lenguaje generado por una GLC

Sea  $G = (N, \Sigma, P, S)$  una GLC, entonces cualquier cadena  $\alpha \in (N \cup \Sigma)^*$  tal que  $S \xRightarrow{*} \alpha$  es una forma sentencial. Notar que el lenguaje  $L(G)$  está formado por las formas sentenciales que están en  $\Sigma^*$ , es decir que consisten solamente de símbolos terminales.



### Lenguaje generado por una GLC

El lenguaje generado por una gramática  $G = (N, \Sigma, P, S)$  (denotado  $L(G)$ ) es:  $L(G) = \{w \in \Sigma^* / S \xRightarrow{*} w\}$ .



## Ejemplo

Construir una GLC que genere el lenguaje

$L = \{ww^R / w \in \{a, b\}^*\}$ , observemos que las cadenas que pertenecen a este lenguaje tienen la forma:

$\underbrace{abb}_w \underbrace{bba}_{w^R}, \underbrace{babb}_w \underbrace{bbab}_{w^R}, \underbrace{baa}_w \underbrace{aab}_{w^R}, \underbrace{baabbb}_w \underbrace{bbbaab}_{w^R}, \text{ etc.}$

Para construir una gramática que genere este lenguaje, debemos observar como se aparean los símbolos, si miramos la primer cadena vemos que la primera  $a$  se corresponde con la última  $a$ , la  $b$  que sigue lo hace con la  $b$  anterior a la última  $a$ , y así siguiendo. De manera similar ocurre en cualquier cadena que pertenece a este lenguaje, corroborarlo.

## Ejemplo (Cont.)

Observemos la sincronización que se da en las cadenas de este lenguaje:



Entonces una gramática que genere este lenguaje puede ser la siguiente:

$$G = \langle \{S\}, \{a, b\}, P, S \rangle$$
$$P = \{S \rightarrow aSa \mid bSb \mid \lambda\}$$

*Determinar, usando la relación deriva, si la gramática construida genera las cadenas miembros del lenguaje.*

## Ejemplo

Construir una GLC que genere el lenguaje

$L = \{w \in \{a, b\}^* / |w|_a = |w|_b\}$ , observemos que las cadenas que pertenecen a este lenguaje tienen la forma:  
*aaabbababb, aabb, bababababa, bbaaaabb, etc.*

Para pensar como construir una GLC que genere este lenguaje tenemos que observar que, cada vez que se genere una *a* debe generarse una *b* y viceversa, el número de veces que querramos, por lo tanto la gramática podría ser:



## Ejemplo (Cont.)

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

*Determinar, usando la relación deriva, si la gramática construida, genera las cadenas miembros del lenguaje.*

***Genera todas las cadenas posibles que pertenecen al lenguaje?***

Por ejemplo, permite generar cadenas como: **abba**, **aabbbbbaa**, **baab**, etc. ? No, no se pueden generar cadenas como estas, por lo tanto es necesario revisar la gramática y obtener las reglas que permitan generar todas las cadenas que pertenecen al lenguaje.

## Ejemplo (Cont.)

Posibles soluciones:

1)

$$S \rightarrow aSbS \mid bSaS \mid \lambda$$

2)

$$S \rightarrow aSb \mid bSa \mid SS \mid \lambda$$

Corroborarlo!

# Ejercicios

Construir una GLC para cada uno de los siguientes lenguajes:

- ❖  $L_1 = \{a^n b^m c^{n+m} / n, m \geq 0\} = \{\lambda, abbccc, ac, bc, aabccc, \dots\}$
- ❖  $L_2 = \{a^i b^k c b^k a^i / i, k > 0\}$
- ❖  $L_3 = \{0^i 1^{i+k} 2^k 3^{n+1} / i, k, n \geq 0\}$

**Pregunta:** ¿por qué decimos que las GLC permiten describir los aspectos sintácticos de los lenguajes de programación?



# Derivaciones de más a la izquierda y más a la derecha

La forma de aplicar las producciones en una secuencia de derivaciones puede ser arbitraria, en el sentido que cualquier símbolo no terminal de una forma sentencial puede ser escogido para reemplazarlo por su parte derecha.

Sin embargo, es posible establecer algún criterio de selección del no terminal a expandir, con la idea de restringir el número de elecciones que se tienen para derivar una cadena. Tales criterios son:

## Derivaciones en una GLC

- ▶ Elegir siempre el no terminal de más a la izquierda. Tal derivación se denomina *derivación de más a la izquierda*, y se usa la notación  $\Rightarrow_{lm} \xRightarrow{*}_{lm}$ .
- ▶ Elegir siempre el no terminal de más a la derecha. Tal derivación se denomina *derivación de más a la derecha*, y se usa la notación  $\Rightarrow_{rm} \xRightarrow{*}_{rm}$ .

**$lm$  = left more = derivación de más a la izquierda**

**$rm$  = right more = derivación de más a la derecha**

# Ejemplo derivaciones de más a la izquierda y más a la derecha

Dada la gramática:

$$\begin{aligned} E &\rightarrow I \mid E + E \mid E * E \mid (E) \\ I &\rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{aligned}$$

$E \Rightarrow_{lm} E * E \Rightarrow_{lm} I * E \Rightarrow_{lm} b * E \Rightarrow_{lm} b * (E) \Rightarrow_{lm}$   
 $b * (E + E) \Rightarrow_{lm} b * (I + E) \Rightarrow_{lm} b * (b + E) \Rightarrow_{lm}$   
 $b * (b + I) \Rightarrow_{lm} b * (b + I0) \Rightarrow_{lm} b * (b + I00) \Rightarrow_{lm} b * (b + a00)$   
Generar la misma cadena utilizando derivaciones de más a la derecha y la relación deriva. Queda como ejercicio.

- El análisis sintáctico está estrechamente relacionado con el modo de realizar las derivaciones (más a la izquierda o más a la derecha).

# Árbol de derivación o sintáctico

Es un recurso alternativo para representar las derivaciones de las gramáticas Tipo 2. Este es un concepto muy importante en el contexto de análisis sintáctico (etapa importante en el proceso de compilación) de lenguajes de programación.

**Definición:** Sea  $G = (N, \Sigma, P, S)$  una GLC, luego un árbol es un *Árbol de Derivación* para  $G$  si:



# Árbol de derivación

1. Cada nodo interior está rotulado por un símbolo no terminal.
2. Cada hoja es rotulada por un símbolo de  $(N \cup \Sigma) \cup \{\lambda\}$ . No obstante si el nodo  $n$  tiene rótulo  $\lambda$ , luego  $n$  es una hoja y es el único descendiente de su padre. (Para las producciones del tipo  $A \rightarrow \lambda$ )
3. Si un nodo interior  $n$  tiene rótulo  $A$  y los vértices  $n_1, n_2, \dots, n_k$  son los descendientes de  $n$ , de izquierda a derecha, con rótulos  $X_1, X_2, \dots, X_k$  respectivamente, luego  $A \rightarrow X_1 X_2 \dots X_k \in P$

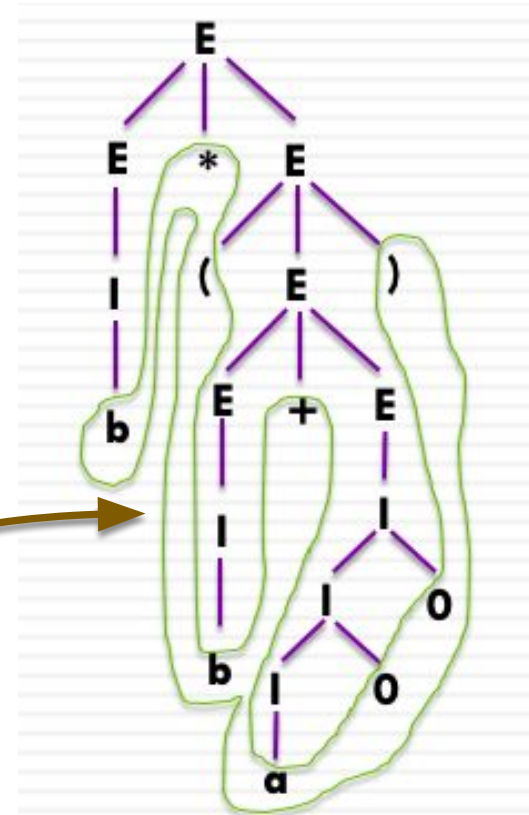
Los árboles de derivación también se denominan árboles sintácticos o árboles de parser.



## Ejemplo Árbol de derivación

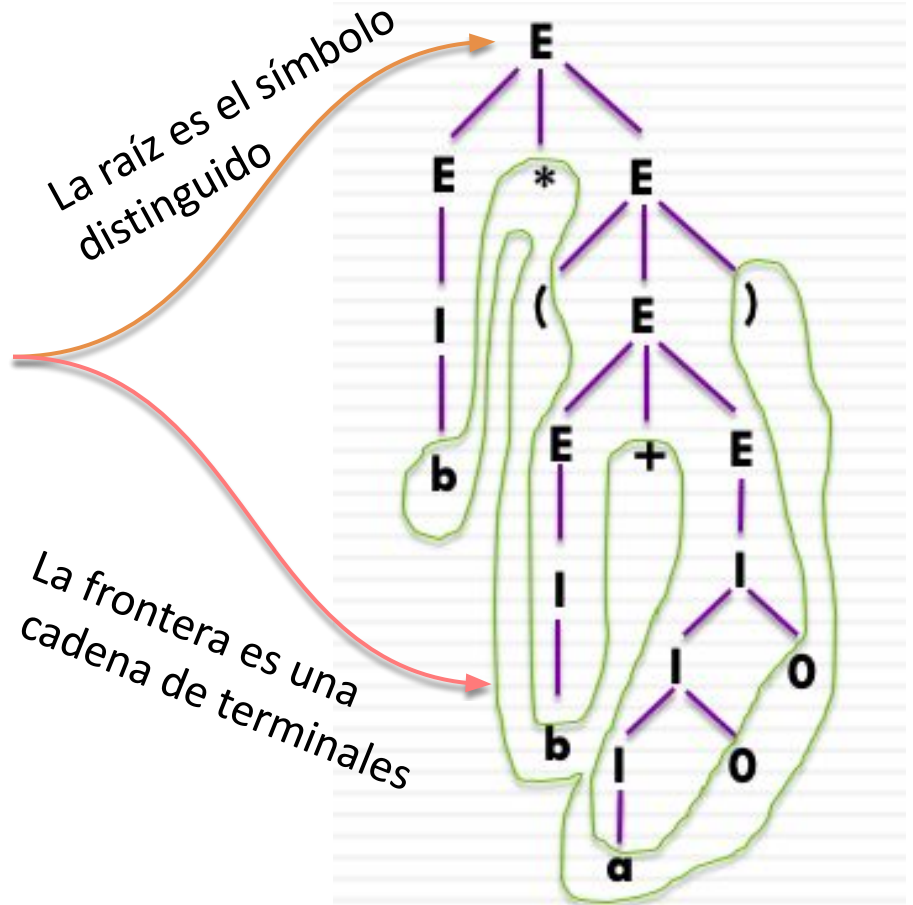
$$E \rightarrow I \mid E + E \mid E * E \mid (E)$$
$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

Frontera del árbol



# Árbol de derivación

Los árboles de derivación que nos importan son aquellos en los que:



# Ejemplo: construir el árbol de derivación para $w = abbccc$

Dada la siguiente GLC:

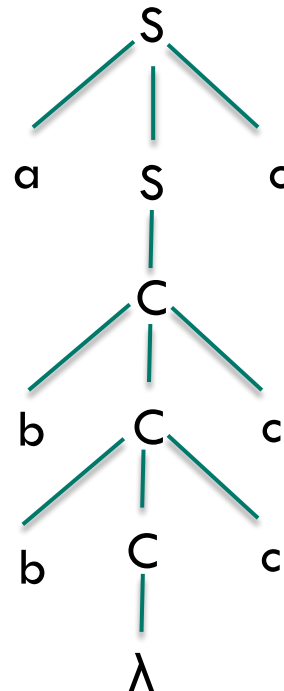
$$S \rightarrow aSc / C$$

$$C \rightarrow bCc / \lambda$$

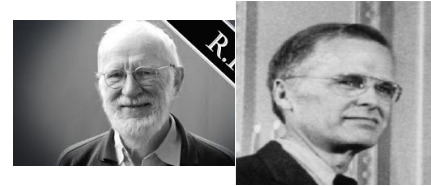
**genera**



$$L_1 = \{a^n b^m c^{n+m} / n, m \geq 0\}$$



# Backus Naur Form (BNF)



La BNF fue desarrollada por John Backus y Peter Naur en la década de 1960 como una herramienta para definir la sintaxis del lenguaje de programación Algol 60.

Desde entonces, se ha convertido en una notación estándar para describir la sintaxis de los lenguajes de programación y otros sistemas formales.

En esta notación se utilizan comúnmente las siguientes convenciones:

- ▶ Los no terminales se escriben entre paréntesis angulares  $\langle \rangle$ .
- ▶ Los terminales se representan con cadenas de caracteres sin corchetes angulares.
- ▶ El lado izquierdo de cada regla debe tener únicamente un no terminal (ya que es una gramática libre del contexto).
- ▶ El símbolo  $::=$ , que se lee se define como o se reescribe como, se utiliza en lugar de  $\rightarrow$ .

# Ejemplos

BNF para describir el lenguaje cuyas cadenas son números:

```
<number> ::= <digit> | <number> <digit>
```

```
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

BNF para el lenguaje cuyas cadenas consisten en paréntesis anidados:

```
<cadena> ::= <cadena> <parentesis> / <parentesis>
```

```
<parentesis> ::= (<cadena>)/()
```

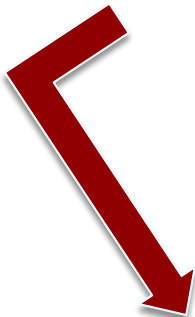
**Pregunta:** *¿qué rol juega la sintaxis en este contexto? o ¿las BNF sirven para describir sintaxis?*

## Ejemplos (Cont.)

$\langle \text{expresion} \rangle ::= \langle \text{expresion} \rangle + \langle \text{term} \rangle \mid \langle \text{term} \rangle$   
 $\langle \text{term} \rangle ::= \langle \text{term} \rangle * \langle \text{factor} \rangle \mid \langle \text{factor} \rangle$   
 $\langle \text{factor} \rangle ::= (\langle \text{expresion} \rangle) \mid \langle \text{iden} \rangle \mid \langle \text{entero} \rangle$   
 $\langle \text{iden} \rangle ::= \langle \text{letra} \rangle \mid \langle \text{iden} \rangle \langle \text{letra} \rangle \mid \langle \text{iden} \rangle \langle \text{digito} \rangle$   
 $\langle \text{entero} \rangle ::= \langle \text{digito} \rangle \mid \langle \text{entero} \rangle \langle \text{digito} \rangle$   
 $\langle \text{letra} \rangle ::= A|B|\dots|Z$   
 $\langle \text{digito} \rangle ::= 0|1|2|3|4|5|6|7|8|9$

**Aclaración:** Considerar que se hace abuso de notación, lo correcto es escribir todas las letras, separados por la barra.

Se usa solo a los fines de simplificar la escritura en esta presentación



¿Qué cadenas permite describir esta BNF?

Usamos árboles de derivación o la relación deriva para determinar qué cadenas se pueden generar.

## BNF Extendida (BNFE)

La BNFE surge a partir de algunas extensiones realizadas a la BNF, las cuales no afectan la potencia descriptiva, sino que incrementan su legibilidad y facilidad de escritura. Las extensiones son las siguientes:

- ▶ Si un elemento es opcional se encierra entre [ ].

Ejemplo:

$\langle \text{selec} \rangle ::= \text{if}(\langle \text{exp} \rangle) \langle \text{sent} \rangle [\text{else} \langle \text{sent} \rangle]$

- ▶ Para la elección de alternativas se usa | y opcionalmente también se pueden usar ( ).

Ejemplo:

$\langle \text{for} \rangle ::= \text{for} \langle \text{var} \rangle := \langle \text{expresion} \rangle (\text{to} | \text{downto})$   
 $\langle \text{expresion} \rangle \text{ do } \langle \text{sentencia} \rangle$

- ▶ Una secuencia arbitraria se encierra entre {}.

Ejemplo:

$\langle \text{lista} - \text{ident} \rangle ::= \langle \text{identificador} \rangle \{, \langle \text{identificador} \rangle\}$



# Ejemplo

## BNF

$\langle \text{entero} - \text{con} - \text{signo} \rangle ::= \langle \text{entero} \rangle \mid \langle \text{signo} \rangle \langle \text{entero} \rangle$

$\langle \text{entero} \rangle ::= \langle \text{digito} \rangle \mid \langle \text{digito} \rangle \langle \text{entero} \rangle$

$\langle \text{signo} \rangle ::= + \mid -$

$\langle \text{digito} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

## BNF extendida

$\langle \text{entero} - \text{con} - \text{signo} \rangle ::= [+|-] \langle \text{digito} \rangle \{ \langle \text{digito} \rangle \}$

**Pregunta:** ¿estará completa esta BNFE?



# BNFE para el lenguaje de programación Lisp

$\langle \text{expressions} \rangle ::= \langle \text{atomic} - \text{symbol} \rangle \mid$   
 $(\langle \text{expressions} \rangle . \langle \text{expressions} \rangle) \mid \langle \text{list} \rangle$   
 $\langle \text{list} \rangle ::= \{(\langle \text{expressions} \rangle)\}$   
 $\langle \text{atomic} - \text{symbol} \rangle ::= \langle \text{letter} \rangle \langle \text{atom} - \text{part} \rangle$   
 $\langle \text{atom} - \text{part} \rangle ::= \langle \text{letter} \rangle \langle \text{atom} - \text{part} \rangle \mid$   
 $\langle \text{digit} \rangle \langle \text{atom} - \text{part} \rangle$   
 $\langle \text{letter} \rangle ::= a|b|\dots|z$   
 $\langle \text{digit} \rangle ::= 0|1|\dots|9$

# Ambigüedad

**Definición:** una gramática  $G$ , libre de contexto es ambigua si existe al menos una cadena  $w$  en  $L(G)$  para la cual existe más de un Árbol de Derivación con frontera  $w$ .

**Teorema 2:** para cada gramática  $G = (N, \Sigma, P, S)$  y la cadena  $w \in \Sigma^*$ ,  $w$  tiene dos árboles de parser distintos sí y sólo sí tiene dos derivaciones de más a la izquierda distintas, partiendo de  $S$ .

Lo mismo se establece para derivaciones de más a la derecha.

# Ejemplos

Dada la siguiente gramática:

$$G = \langle \{S, A\}, \{a\}, P, S \rangle$$

$$P: \begin{aligned} S &\rightarrow AA \\ A &\rightarrow aSa \mid a \end{aligned}$$

La cual genera el lenguaje  $L = \{a^{2+3i} / i \geq 0\}$ .

Analicemos si dicha gramática es ambigua, tomemos la cadena  $aaaaa$  y veamos si es posible hallar dos derivaciones de más a la izquierda distintas:

$$\begin{aligned} S &\Rightarrow_{lm} AA \Rightarrow_{lm} aA \Rightarrow_{lm} aaSa \Rightarrow_{lm} aaAAa \Rightarrow_{lm} aaaAa \Rightarrow aaaaa \\ S &\Rightarrow_{lm} AA \Rightarrow_{lm} aSaA \Rightarrow_{lm} aAAaA \Rightarrow_{lm} aaAaA \Rightarrow_{lm} aaaaA \Rightarrow aaaaa \end{aligned}$$

*Por lo tanto la gramática es ambigua.*

## Removiendo ambigüedad

Así como es posible transformar un AFND a un AFD, sería útil poder eliminar ambigüedad en una gramática de manera algorítmica. Sin embargo, el hecho es que **no hay un algoritmo** para determinar si una gramática es ambigua o no, el problema es indecidible.

Dada una gramática ambigua, en algunos casos, se puede hallar una gramática no ambigua equivalente.

Sin embargo, existen lenguajes cuya ambigüedad es inevitable (inherentemente ambiguos).



## Ejemplo

Para la gramática que analizamos previamente:

$$G = \langle \{S, A\}, \{a\}, P, S \rangle$$

$$P: \begin{aligned} S &\rightarrow AA \\ A &\rightarrow aSa \mid a \end{aligned}$$

Analicemos si es posible eliminar la ambigüedad. Lo que podemos observar en este caso, es que la producción  $S \rightarrow AA$ , es la que la provoca entonces lo que tenemos que hacer es reemplazar esa regla. Una forma es generar dos  $a$ 's y luego generar grupos de 3  $a$ 's:

$$G = \langle \{S, T\}, \{a\}, P, S \rangle$$

*Possible solución* 

$$P: \begin{aligned} S &\rightarrow aa \mid aaT \\ T &\rightarrow aaaT \mid aaa \end{aligned}$$

## Ejemplo

Consideremos la siguiente BNF que describe la sintaxis (simplificada) de una sentencia de asignación en Pascal:

```
< sent - asig > ::= < var > := < exp >  
< exp > ::= < exp > + < exp > | < exp > - < exp > | (< exp >)|  
           < exp > * < exp > | < exp > div < exp > | < var > | < num >  
< var > ::= A|B|C|D|...|Z  
< num > ::= 0|1|2|3|4|5|6|7|8|9
```

## Ejemplo (Cont.)

$\langle \text{sent} - \text{asig} \rangle ::= \langle \text{var} \rangle := \langle \text{exp} \rangle$   
 $\langle \text{exp} \rangle ::= \langle \text{exp} \rangle + \langle \text{exp} \rangle \mid \langle \text{exp} \rangle - \langle \text{exp} \rangle \mid (\langle \text{exp} \rangle) \mid$   
 $\quad \langle \text{exp} \rangle * \langle \text{exp} \rangle \mid \langle \text{exp} \rangle \text{div} \langle \text{exp} \rangle \mid \langle \text{var} \rangle \mid \langle \text{num} \rangle$   
 $\langle \text{var} \rangle ::= A \mid B \mid C \mid D \mid \dots \mid Z$   
 $\langle \text{num} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Determinar si la BNF es ambigua, utilizando derivaciones de más a la izquierda y considerando la cadena **J := 1 + 2 \* 3**

$\langle \text{sent-asig} \rangle \Rightarrow_{\text{lm}} \langle \text{var} \rangle := \langle \text{exp} \rangle \Rightarrow_{\text{lm}} J := \langle \text{exp} \rangle \Rightarrow_{\text{lm}} J := \langle \text{exp} \rangle + \langle \text{exp} \rangle$   
 $\Rightarrow_{\text{lm}} J := \langle \text{num} \rangle + \langle \text{exp} \rangle \Rightarrow_{\text{lm}} J := 1 + \langle \text{exp} \rangle \Rightarrow_{\text{lm}} J := 1 + \langle \text{exp} \rangle * \langle \text{exp} \rangle \Rightarrow_{\text{lm}}$   
 $J := 1 + \langle \text{num} \rangle * \langle \text{exp} \rangle \Rightarrow_{\text{lm}} J := 1 + 2 * \langle \text{exp} \rangle \Rightarrow_{\text{lm}} J := 1 + 2 * \langle \text{num} \rangle \Rightarrow_{\text{lm}}$   
 **$J := 1 + 2 * 3$**

$\langle \text{sent-asig} \rangle \Rightarrow_{\text{lm}} \langle \text{var} \rangle := \langle \text{exp} \rangle \Rightarrow_{\text{lm}} J := \langle \text{exp} \rangle \Rightarrow_{\text{lm}} J := \langle \text{exp} \rangle * \langle \text{exp} \rangle$   
 $\Rightarrow_{\text{lm}} J := \langle \text{exp} \rangle + \langle \text{exp} \rangle * \langle \text{exp} \rangle \Rightarrow_{\text{lm}} J := \langle \text{num} \rangle + \langle \text{exp} \rangle * \langle \text{exp} \rangle \Rightarrow_{\text{lm}}$   
 $J := 1 + \langle \text{exp} \rangle * \langle \text{exp} \rangle \Rightarrow_{\text{lm}} J := 1 + \langle \text{num} \rangle * \langle \text{exp} \rangle \Rightarrow_{\text{lm}} J := 1 + 2 * \langle \text{exp} \rangle$   
 $\Rightarrow_{\text{lm}} J := 1 + 2 * \langle \text{num} \rangle \Rightarrow_{\text{lm}}$   **$J := 1 + 2 * 3$**

## Ejemplo (Cont.)

- Al encontrar dos derivaciones de más a la izquierda distintas que generan la misma cadena, se concluye que la BNF es ambigua.
- Se deja como ejercicio determinar si es posible corregir la ambigüedad en la BNF dada y de ser factible corregirla.



¿DUDAS?

