Ciencias de la Computación I

Gramáticas Libres del Contexto y Lenguajes Libres del Contexto

Gramáticas Formales

Una gramática formal es una cuadrupla G = <N, T, P, S>

N = conjunto finito de símbolos no terminales

T = conjunto finito de símbolos terminales

$N \cap T = \varnothing$

S = símbolo distinguido o axioma $S \notin (N \cup T)$

P = conjunto finito de reglas de producción (permiten generar cadenas a partir de S)

$\alpha \rightarrow \beta$

$\alpha = \varphi A \rho$

$\beta = \varphi \omega \rho$

$A \in N \cup \{S\}$

$\varphi, \omega, \rho \in (N \cup T)^*$

De acuerdo a formato de reglas se pueden definir 4 tipos de gramáticas y sus correspondientes lenguajes

Gramáticas Libres del Contexto (GLC) (Tipo 2)

- **Generan los lenguajes libres del contexto (reconocidos por Autómatas de Pila)**
- **Son importantes para definir la sintaxis de lenguajes de programación**
- **Se definen como una cuadrupla G = <N, T, P, S>**

  N = conjunto finito de símbolos no terminales

  T = conjunto finito de símbolos terminales

  N ∩ T = ∅

  S = símbolo distinguido o axioma S ∉ (N ∪ T)

  P = conjunto finito de reglas de producción (con formato de tipo 2)

$A \rightarrow w$	$A \in N \cup \{S\}$
$S \rightarrow \varepsilon$	$w \in (N \cup T)^* - \{\varepsilon\}$

Del lado izquierdo de la regla un no terminal o el símbolo distinguido S
Del lado derecho de la regla una combinación de terminales y/o no terminales excepto ε.
Si $\varepsilon \in L$ se agrega $S \rightarrow \varepsilon$

Gramáticas Libres del Contexto (Tipo 2)

Ejemplo:

Sea G = <{A}, {a, b}, P, S> donde P = { $S \rightarrow A$,
$A \rightarrow aAb$,
$A \rightarrow ab$ }

Derivaciones de cadenas

$S \Rightarrow A \Rightarrow ab$ — $ab \in L(G)$

$S \Rightarrow A \Rightarrow aAb \Rightarrow aabb$ — $aabb \in L(G)$

$S \Rightarrow A \Rightarrow aAb \Rightarrow aaAbb \Rightarrow aaabbb$ — $aaabbb \in L(G)$

Se pueden generar infinitas cadenas …..

G es libre del contexto y genera :

$L = \{a^n b^n / n > 0\}$

Gramáticas Libres del Contexto (Tipo 2)

• **Las reglas de producción indican cómo reemplazar un no terminal sin considerar el contexto en el que se encuentra**

Ejemplo: G gramática libre del contexto

Sea G = <{A}, {a, b}, P, S> donde P = { S → A, A → aAb, A → ab }

- Del lado izquierdo siempre un no terminal o el símbolo distinguido
- Del lado derecho, cualquier combinación de terminales y no terminales, excepto ε. Sólo si ε ∈ L, se agrega S → ε

S ⇒ A ⇒ a A b ⇒ aabb

S ⇒ A ⇒ a A b ⇒ aa A bb ⇒ aaabbb

No importa en qué "contexto" está el no terminal A

Se lo reemplaza por la cadena en el lado derecho de la regla de producción

Gramáticas Libres del Contexto (Tipo 2)

Derivación inmediata ⇒ : La cadena ω_2 se obtiene de la cadena ω_1 en un paso usando reglas de P de una gramática G =< N, T, P, S>

$\omega_1 \Rightarrow \omega_2$ sí y sólo sí

Caso 1) $\alpha A \beta \Rightarrow \alpha\, w\, \beta$

$A \rightarrow w$ es regla de P
$\alpha, \beta \in (N \cup T)^*$
$A \in N \cup \{S\}$
$w \in (N \cup T)^* - \{ \varepsilon \}$

Caso 2) $S \Rightarrow \varepsilon$

si $w_2 = \varepsilon$ $S \rightarrow \varepsilon$ es regla de P

Ejemplo G = <{A}, {a, b}, {S → A, A → aAb, A → ab }, S>

S ⇒ A ⇒ aAb ⇒ aaAbb ⇒ aaabbb

(aAb: αAβ; aaAbb: α w β)

Gramáticas Libres del Contexto (Tipo 2)

$\overset{*}{\Rightarrow}$ (clausura reflexiva y transitiva de la derivación inmediata $\Rightarrow$)

La cadena α_n se obtiene de la cadena α_1 en cero o más pasos usando las reglas de P

$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$ decimos que $\alpha_1 \overset{*}{\Rightarrow} \alpha_n$ $\quad \alpha_i \in (N \cup T)^*$

Ejemplo G = <{A}, {a, b}, {S → A, A → aAb, A → ab }, S>

$S \Rightarrow A \Rightarrow aAb \Rightarrow aaAbb \Rightarrow aaabbb$ → $S \overset{*}{\Rightarrow} aaabbb$

En varios pasos

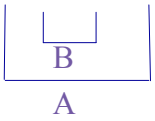
Lenguaje generado por una gramática libre del contexto G = <N, T, P, S>:

$$L(G) = \{ x / S \overset{*}{\Rightarrow} x \ \ y \ \ x \in T^* \}$$

Una cadena $x \in L(G)$, si se puede generar a partir de S y sólo está formada por símbolos terminales

Construcción de Gramáticas Libres del Contexto

$L=\{ c^j a^n b^{2n} d^j \ / \ j, n >= 0 \}$ Caso Anidamiento



$S \rightarrow \varepsilon$

$S \rightarrow A$

$A \rightarrow c\ A\ d$

$A \rightarrow cd$

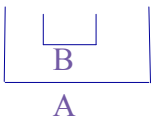
$A \rightarrow B$ G=<{A, B},{a,b,c,d},P, S>

$B \rightarrow aBbb$

$B \rightarrow abb$

Construcción de Gramáticas Libres del Contexto

$L=\{c^j a^n b^{2n} d^j / j, n>=0\}$



$S \rightarrow \varepsilon$
$S \rightarrow A$
$A \rightarrow c\ A\ d$
$A \rightarrow cd$
$A \rightarrow B$
$B \rightarrow a\ B\ bb$
$B \rightarrow a\ bb$

$G=<\{A, B\},\{a,b,c,d\},P, S>$

Controlar si generamos todas y sólo las cadenas del lenguaje

Casos	Cadenas	Ejemplos de derivaciones
si j, n=0	ε	$S \Rightarrow \varepsilon$
si j=0 y n>0	$a^n b^{2n}$	$S \Rightarrow A \Rightarrow B \Rightarrow aBbb \Rightarrow aabbbb$
si n=0 y j>0	$c^j d^j$	$S \Rightarrow A \Rightarrow cAd \Rightarrow ccdd$
si j, n>0	$c^j a^n b^{2n} d^j$	$S \Rightarrow A \Rightarrow cAd \Rightarrow cBd \Rightarrow cabbd$

Construcción de Gramáticas Libres del Contexto

$L=\{h^m e^m g^p d^{2p+1} / m, p>=0\}$ Caso Concatenación

$S \rightarrow AB$
$S \rightarrow B$
$A \rightarrow h\ A\ e$
$A \rightarrow he$
$B \rightarrow g\ B\ dd$
$B \rightarrow d$

$G=<\{A, B\},\{h, e, g, d\},P, S>$

Construcción de Gramáticas Libres del Contexto

$L=\{h^m e^m g^p d^{2p+1} / m, p>=0\}$

A B

$S \rightarrow AB$
$S \rightarrow B$
$A \rightarrow h\ A\ e$
$A \rightarrow he$
$B \rightarrow g\ B\ dd$
$B \rightarrow d$

$G=<\{A, B\},\{h, e, g, d\},P, S>$

Controlar si generamos todas y sólo las cadenas del lenguaje

Casos	Cadenas	Ejemplos de derivaciones
si m, p=0	d	$S \Rightarrow B \Rightarrow d$
si m=0 y p>0	$g^p\ d^{2p+1}$	$S \Rightarrow B \Rightarrow gBdd \Rightarrow gddd$
si p=0 y m>0	$h^m\ e^m d$	$S \Rightarrow AB \Rightarrow heB \Rightarrow hed$
si p, m>0	$h^m\ e^m g^p\ d^{2p+1}$	$S \Rightarrow AB \Rightarrow heB \Rightarrow hegBdd \Rightarrow hegddd$

Backus-Naur Form (BNF)

• Notación utilizada frecuentemente para escribir gramáticas de tipo 2 o libres del contexto.

• Esta notación sigue las siguientes convenciones:

- no terminales se escriben entre **< >**
- terminales son cadenas de caracteres sin < >
- en lugar de → se utiliza **:: =** que se lee “se define como”
- varias reglas del tipo

<A> :: = <B_1>
<A> :: = <B_2>
...
<A> :: = <B_n>

Se pueden escribir como

<A> :: = <B_1> | <B_2> | ... | <B_n>

Backus-Naur Form (BNF)

Ejemplos:

L = { x / x ∈ { (,) }* y x es una cadena de paréntesis balanceados }

BNF para L

<cadena_par> :: = <parentesis> | <parentesis> <cadena_par>

<parentesis> :: = (<cadena_par >) | ( )

Ejemplos

((( ))) ∈ L

(( ))( ) ∈ L

(( ) ∉ L

Backus-Naur Form (BNF)

Ejemplos:

L = { x / x ∈ { begin, end }* y x es una cadena de begin end balanceados }

BNF para L

<lista> :: = <anidados> | <anidados> <lista>

<anidados> :: = begin <lista > end | begin end

Ejemplos

begin begin end end begin end ∈ L

begin begin end ∉ L

Backus-Naur Form (BNF)

BNF para sentencia **for** de Pascal

<sentencia_for> :: = **for** <variable> **:=** <lista_for> **do** <sentencia>

<lista_for> :: = <exparit> **to** <exparit> | <exparit> **downto** <exparit>

<variable> :: = **a** | **…** | **z** /*en este ejemplo simplificamos*/

<sentencia> :: = <variable> **:=** <variable> /*en este ejemplo simplificamos*/

<exparit> :: = **0** | **…** | **9** /*en este ejemplo simplificamos*/

Ejemplos

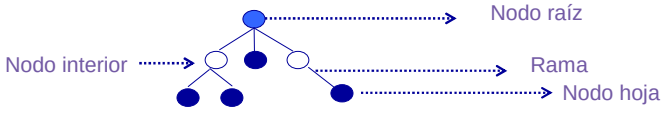
for i := 1 **to** 3 **do** i := j — bien definida según BNF anterior

for i := 1 **to** 9 **do** sum:= i — mal definida según BNF anterior

for i = 2 **to** 9 **do** k:=i — mal definida según BNF anterior

Árbol

- El concepto de árbol es muy usado en computación.
- Un árbol es un conjunto de puntos, llamados **nodos**, unidos por líneas, llamadas **arcos.** Un arco conecta dos nodos distintos.



- Propiedades del árbol:

- hay un único nodo distinguido, llamado raíz.

- Un nodo padre puede tener conectados uno o mas nodos hijos. El padre de un nodo se dibuja por encima de los nodos hijos. El nodo raíz no tiene padre. Los nodos hojas no tienen hijos.

- cada nodo es alcanzable desde el nodo raíz mediante un único camino

Árbol de derivación

Permite mostrar gráficamente la derivación de cualquier cadena de un lenguaje a partir del símbolo distinguido de una gramática que genera el lenguaje.

Un árbol es un árbol de derivación para una gramática G = <N, T, P, S> si:

- La raíz del árbol se rotula con S, el símbolo distinguido de G

- Cada nodo interior está rotulado con un símbolo de N

- Cada hoja está rotulada con un símbolo de $N \cup T$

- Si un nodo interior tiene rótulo A y sus hijos tienen rótulos $x_1, x_2, ..., x_n$ entonces la regla $A \rightarrow x_1 x_2 ... x_n \in P$

Para cada cadena del lenguaje generado por una gramática es posible construir al menos un árbol de derivación en el cual cada hoja tiene como rótulo un símbolo terminal y si se leen las hojas de izquierda a derecha se obtiene la cadena.

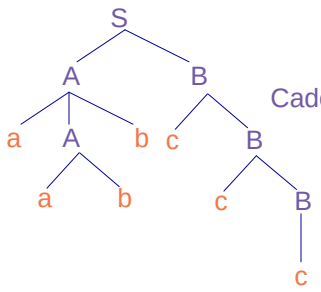
Árbol de derivación

Ejemplo:

Sea G = <{A, B}, {a, b, c}, P, S> donde

P = { $S \rightarrow AB$, $A \rightarrow aAb$, $A \rightarrow ab$, $B \rightarrow cB$, $B \rightarrow c$ }

- La cadena aabbccc $\in$ L(G)?



Cadena derivada: aabbccc

Árbol de derivación

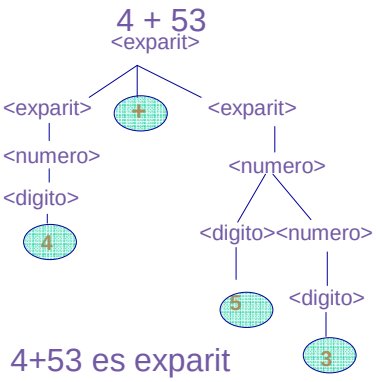
Ejemplo: BNF para expresiones aritméticas "simplificadas"

<exparit> :: = <exparit> **+** <exparit> | <exparit> **-** <exparit> | <exparit> ***** <exparit> |

<exparit> **/** <exparit> | **(**<exparit>**)** | <numero>

<numero> :: =<digito> |<digito><numero>

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



4+53 es exparit

4+5* no es exparit

Árbol de derivación

Ejemplo: BNF para expresiones aritméticas "simplificadas"

<exparit> :: = <exparit> **+** <exparit> | <exparit> **-** <exparit> | <exparit> ***** <exparit> |

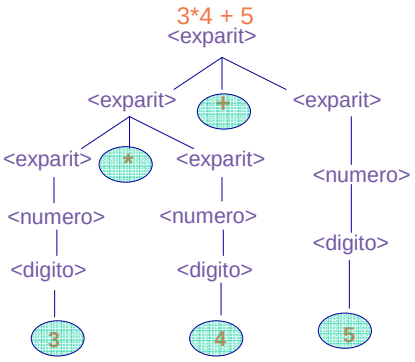
<exparit> **/** <exparit> | **(**<exparit>**)** | <numero>

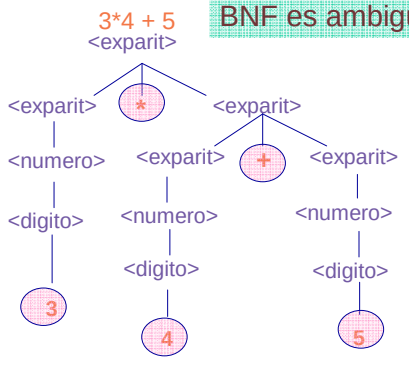
<numero> :: =<digito> |<digito><numero>

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

Si para la misma cadena 3*4+5 hay dos árboles diferentes: BNF es ambiguo

Se quiere derivar la expresión aritmética 3*4 + 5





Gramáticas ambiguas

• En el ejemplo anterior, existen dos árboles de derivación para la misma cadena

→ **gramática ambigua**

• Una **GLC G** es **ambigua** si existe **al menos una cadena** de L(G) para la cual hay **dos o más árboles de derivación distintos**.

• Una **GLC G** es **no ambigua** si **toda cadena** de L(G) **tiene un único árbol de derivación.**

• La ambigüedad puede ser un problema para los lenguajes en los que el significado depende, en parte, de la estructura (lenguaje natural, lenguajes de programación). En el ejemplo anterior 3*4+5 puede ser igual a 17 o 27 según el árbol de derivación que se construya.

• Una cadena que tiene más de un árbol de derivación suele tener más de un significado entonces en aplicaciones de compilación es necesario diseñar, cuando sea posible, gramáticas no ambiguas.

Gramáticas ambiguas

En algunos casos, dada una gramática ambigua, se puede encontrar otra no ambigua que genera el mismo lenguaje.

Por ejemplo, es posible definir una GLC no ambigua para generar las expresiones aritméticas del ejemplo anterior.

Si todas las GLC para un lenguaje L son ambiguas se dice que L es inherentemente ambiguo.

Por ejemplo, $L = \{a^i b^j c^k / i, j, k \geq 0 \text{ y } (i = j \text{ ó } j = k)\}$ es inherentemente ambiguo

Intuitivamente, cualquier GLC para L debe tener reglas generar cadenas en las que i = j y también reglas para generar cadenas en las que j = k. Si una cadena tiene i = j = k claramente tendrá dos derivaciones posibles.

Teorema:

No existe un algoritmo para determinar si una GLC es ambigua o no.

BNF no ambiguo

Ejemplo: BNF no ambiguo para expresiones aritméticas "simplificadas"

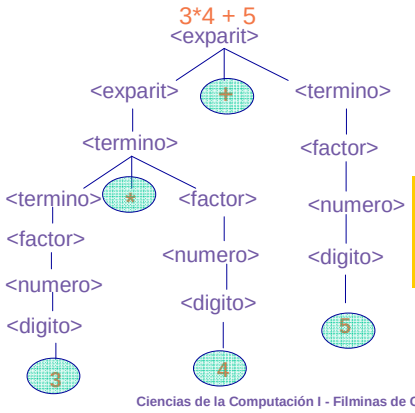
<exparit> :: = <exparit> **+** <termino> | <exparit> **-** <termino> |<termino>

<termino> :: = <termino> ***** <factor>| <termino> **/** <factor> | <factor>

<factor> :: =**(**<exparit>**)** | <numero>

<numero> :: =<digito> |<digito><numero>

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



Para 3*4+5
un único árbol de derivación

En general,
toda cadena ↔ un árbol de derivación
entonces este BNF es no ambiguo