Ciencias de la Computación I

Propiedades de Clausura de Lenguajes Libres del Contexto

Propiedades de Clausura de Lenguajes Libres del Contexto

Los lenguajes libres del contexto (LLC) son cerrados bajo las siguientes operaciones:

- ✓ **Unión**
- ✓ **Clausura**
- ✓ **Reversa**
- ✓ **Concatenación**

Los lenguajes libres del contexto (LLC) no son cerrados bajo las siguientes operaciones:

- ✓ **Intersección**
- ✓ **Complemento**

Ejemplo: Unión de Lenguajes Libres del Contexto

$L_1 = \{ a^n b^n / n \geq 0 \}$

$L(G_1)=L_1$

$G_1=<\{A\},\{a,b\},P_1,S_1>$

$P_1=\{S_1 \rightarrow \varepsilon,$
$S_1 \rightarrow A,$
$A \rightarrow aAb,$
$A \rightarrow ab\}$

$L_2 = \{ b^n a^n / n \geq 0 \}$

$L(G_2)=L_2$

$G_2=<\{B\},\{a,b\},P_2,S_2>$

$P_2=\{S_2 \rightarrow \varepsilon,$
$S_2 \rightarrow B,$
$B \rightarrow bBa,$
$B \rightarrow ba\}$

A partir de G_1 y G_2, se puede construir una gramática G como:

$G=<\{A, B, S_1, S_2\}, \{a, b\}, P, S>$

Si los no terminales en G_1 y G_2 tienen el mismo nombre deben renombrarse

$P=P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\} - \{S_1 \rightarrow \varepsilon, S_2 \rightarrow \varepsilon\}$

Considerar el caso especial

Si en $P_1 \cup P_2$ estaba $S_1 \rightarrow \varepsilon$ ó $S_2 \rightarrow \varepsilon$ agregar en P: $S \rightarrow \varepsilon$

$P=\{S \rightarrow \varepsilon, S \rightarrow S_1, S \rightarrow S_2, S_1 \rightarrow A, A \rightarrow aAb, A \rightarrow ab, S_2 \rightarrow B, B \rightarrow bBa, B \rightarrow ba\}$

$L(G) = L(G_1) \cup L(G_2) = L_1 \cup L_2$ G es tipo 2 luego $L_1 \cup L_2$ es LLC

Unión de Lenguajes Libres del Contexto

Teorema:

Dados L_1 y L_2 LLC, $L_1 \cup L_2$ es un lenguaje libre del contexto.

Demostración:

Como L_1 es LLC existe una GLC $G_1 = <N_1, T_1, P_1, S_1>$ tal que $L_1 = L(G_1)$

Como L_2 es LLC existe una GLC $G_2 = <N_2, T_2, P_2, S_2>$ tal que $L_2 = L(G_2)$

A partir de G_1 y G_2 es posible definir una gramática libre del contexto

$G =<N, T, P, S>$ tal que $L(G) = L(G_1) \cup L(G_2) = L_1 \cup L_2$

La gramática G se define como sigue

$G = <N_1 \cup N_2 \cup \{S_1, S_2\}, T_1 \cup T_2, P, S>$ $N_1 \cap N_2 = \varnothing$

$P = P_1 \cup P_2 \cup \{ S \rightarrow S_1, S \rightarrow S_2 \} - \{S_1 \rightarrow \varepsilon, S_2 \rightarrow \varepsilon\}$ y además

Si en P_1 está la regla $S_1 \rightarrow \varepsilon$, ó en P_2 $S_2 \rightarrow \varepsilon$ se agrega a P $S \rightarrow \varepsilon$

Como las reglas de P respetan el formato de las GLC, G es una GLC y entonces $L(G) = L(G_1) \cup L(G_2) = L_1 \cup L_2$ es un lenguaje libre del contexto

Ejemplo: Concatenación de Lenguajes Libres del Contexto

$L_1 = \{ a^n b^n / n \geq 0 \}$

$L(G_1)=L_1$

$G_1=<\{A\},\{a,b\},P_1,S_1>$

$P_1=\{S_1 \rightarrow \varepsilon,$
$S_1 \rightarrow A,$
$A \rightarrow aAb,$
$A \rightarrow ab\}$

$L_2 = \{ b^n a^n / n \geq 0 \}$

$L(G_2)=L_2$

$G_2=<\{B\},\{a,b\},P_2,S_2>$

$P_2=\{S_2 \rightarrow \varepsilon,$
$S_2 \rightarrow B,$
$B \rightarrow bBa,$
$B \rightarrow ba\}$

A partir de G_1 y G_2, se puede construir una gramática $G =<\{A,B,S_1,S_2\}, \{a,b\}, P, S>$

$P=P_1 \cup P_2 \cup \{S \rightarrow S_1S_2\} - \{S_1 \rightarrow \varepsilon, S_2 \rightarrow \varepsilon\}$

Considerar casos especiales

Si en $P_1 \cup P_2$ estaban	$S_1 \rightarrow \varepsilon$ y $S_2 \rightarrow \varepsilon$	agregar en P : $S \rightarrow \varepsilon$
Si en P_1 estaba	$S_1 \rightarrow \varepsilon$	agregar en P : $S \rightarrow S_2$
Si en P_2 estaba	$S_2 \rightarrow \varepsilon$	agregar en P : $S \rightarrow S_1$

$P=\{S \rightarrow S_1S_2, S \rightarrow \varepsilon, S \rightarrow S_1, S \rightarrow S_2, S_1 \rightarrow A, A \rightarrow aAb, A \rightarrow ab, S_2 \rightarrow B, B \rightarrow bBa, B \rightarrow ba\}$

$L(G) = L(G_1) \,.\, L(G_2) = L_1 \,.\, L_2$ G es tipo 2 luego $L_1.L_2$ es LLC

Concatenación de Lenguajes Libres del Contexto

Teorema: Dados L_1 y L_2 LLC, $L_1 \,.\, L_2$ es un lenguaje libre del contexto.

Demostración:

Como L_1 es LLC existe una GLC $G_1 = <N_1, T_1, P_1, S_1>$ tal que $L_1 = L(G_1)$

Como L_2 es LLC existe una GLC $G_2 = <N_2, T_2, P_2, S_2>$ tal que $L_2 = L(G_2)$

A partir de G_1 y G_2 es posible definir una gramática libre del contexto

$G =<N, T, P, S>$ tal que $L(G) = L(G_1) \,.\, L(G_2) = L_1 \,.\, L_2$

La gramática G se define como sigue

$G = <N_1 \cup N_2 \cup \{S_1, S_2\}, T_1 \cup T_2, P, S>$ $\quad N_1 \cap N_2 = \varnothing$

$P = (P_1 \cup P_2 \cup \{ S \rightarrow S_1S_2 \}) - \{S_1 \rightarrow \varepsilon, S_2 \rightarrow \varepsilon\}$ y además

- **Si en P_1 está la regla $S_1 \rightarrow \varepsilon$, se agrega a P la regla $S \rightarrow S_2$**
- **Si en P_2 está la regla $S_2 \rightarrow \varepsilon$, se agrega a P la regla $S \rightarrow S_1$**
- **Si en P_1 está la regla $S_1 \rightarrow \varepsilon$, y en P_2 $S_2 \rightarrow \varepsilon$ se agrega a P $S \rightarrow \varepsilon$**

Como las reglas de P respetan el formato de las GLC, G es una GLC y entonces $L(G) = L(G_1) \,.\, L(G_2) = L_1 \,.\, L_2$ es un lenguaje libre del contexto

Ejemplo: Clausura de un Lenguaje Libre del Contexto

$L_1 = \{ a^n b^n / n \geq 0 \}$

$L(G_1)=L_1$

$G_1=<\{A\},\{a,b\},P_1,S_1>$

$P_1=\{S_1 \rightarrow \varepsilon,$
$S_1 \rightarrow A,$
$A \rightarrow aAb,$
$A \rightarrow ab\}$

A partir de G_1 se puede construir una gramática G como:

$G=<\{A, X, S_1\}, \{a, b\}, P, S>$

$P=\{S \rightarrow \varepsilon, S \rightarrow X, X \rightarrow S_1, X \rightarrow S_1X, S_1 \rightarrow A, A \rightarrow aAb, A \rightarrow ab\}$

$L(G) = (L(G_1))^* = L_1^*$ G es tipo 2 luego L_1^* es LLC

Clausura de un Lenguaje Libre del Contexto

Teorema: Dado L_1 LLC, L_1^* es un lenguaje libre del contexto.

Demostración:

Como L_1 es LLC existe una GLC $G_1 = <N_1, T_1, P_1, S_1>$ tal que $L_1 = L(G_1)$

A partir de G_1 es posible definir una gramática libre del contexto

$G =<N, T, P, S>$ tal que $L(G) = L^*(G_1)$

La gramática G se define como sigue $G = <N, T, P, S>$

$N = N_1 \cup \{S_1, X\}$ $X \notin N_1$

$T = T_1$

$P = P_1 \cup \{ S \rightarrow \varepsilon, S \rightarrow X, X \rightarrow S_1X, X \rightarrow S_1 \} - \{S_1 \rightarrow \varepsilon\}$

Como las reglas de P respetan el formato de las GLC, G es una GLC y entonces $L(G) = (L (G_1))^* = L_1^*$ es un lenguaje libre del contexto

Ejemplo Reversa de un Lenguaje Libre del Contexto

$L_1 = \{ a^n b^n / n \geq 0 \}$

$L(G_1)=L_1$

$G_1=<\{A\}, \{a, b\}, P_1, S_1>$

$P_1=\{S_1 \rightarrow \varepsilon,$
$S_1 \rightarrow A,$
$A \rightarrow aAb,$
$A \rightarrow ab\}$

A partir de G_1 se puede construir G como:

$G=<\{A\}, \{a, b\}, P, S_1>$

$P = \{S_1 \rightarrow \varepsilon,\ S_1 \rightarrow A,\ A \rightarrow bAa,\ A \rightarrow ba\}$

$L(G) = L(G_1)^R = L_1^R$

G es tipo 2 luego
L_1^R es LLC

Reversa de un Lenguaje Libre del Contexto

Teorema: Dado L_1 LLC, L_1^R es un lenguaje libre del contexto.

Demostración:

Como L_1 es LLC existe una GLC $G_1 = <N_1, T_1, P_1, S_1>$ tal que $L_1 = L(G_1)$

A partir de G_1 es posible definir una gramática libre del contexto
$G =<N, T, P, S>$ tal que $L(G) = (L(G_1))^R$

La gramática G se define como sigue $G = <N_1, T_1, P, S_1>$

Donde $P = P_1$ con cada regla de producción de P_1 de la forma

$A \rightarrow \omega$ reemplazada por $A \rightarrow \omega^R$ para $A \in N \cup \{S\}$ y $\omega \in \{N \cup T\}^* - \{\varepsilon\}$

Como las reglas de P respetan el formato de las GLC, G es una GLC y entonces $L(G) = (L(G_1))^R = L_1^R$ es un lenguaje libre del contexto

Intersección de Lenguajes Libres del Contexto

Es una operación cerrada?
Contraejemplo

$L_1 = \{ a^n b^n c^k / n, k \geq 0 \}$ Libre del contexto

$L_2 = \{ a^k b^n c^n / n, k \geq 0 \}$ Libre del contexto

$L_1 \cap L_2 = \{ a^n b^n c^n / n \geq 0 \}$ No es Libre del contexto
es Sensible al contexto

La intersección NO es una operación CERRADA para Lenguajes Libres del Contexto

Ciencias de la Computación I

BNF

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 $\rightarrow$ 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

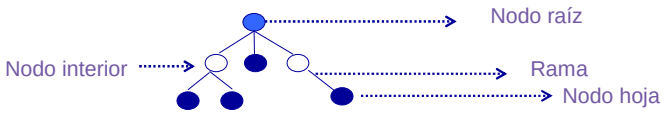
for i := 1 **to** 9 **do** sum:= i mal definida según BNF anterior

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.

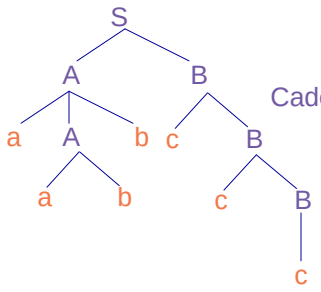
Árbol de derivación

Ejemplo:

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

P = { S → AB, A → aAb, A → ab, B → cB, B → c }

- La cadena aabbccc ∈ L(G)?



Cadena derivada: aabbccc

Árbol de derivación

Ejemplo: BNF para expresiones aritméticas "simplificadas"

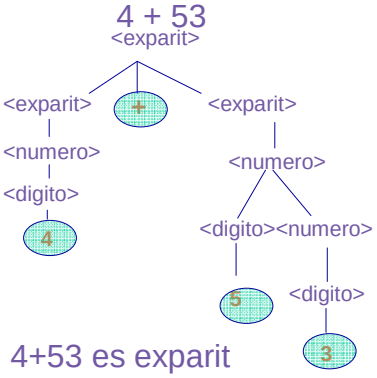
<exparit> :: = <exparit> **+** <exparit> | <exparit> **-** <exparit> | <exparit> ***** <exparit> |

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

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

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

4 + 53



4+53 es exparit

4+5*

no se llega
a un terminal

4+5* no es exparit

Árbol de derivación

Ejemplo: BNF para expresiones aritméticas "simplificadas"
<exparit> :: = <exparit> **+** <exparit> | <exparit> **-** <exparit> | <exparit> ***** <exparit> |

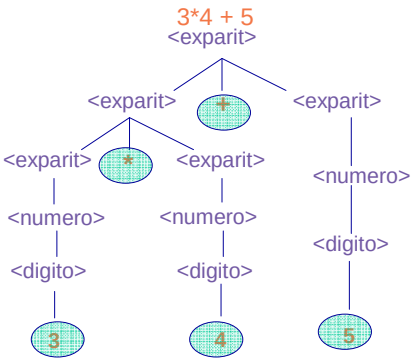
<exparit> **/** <exparit> | **(**<exparit>**)** | <numero>

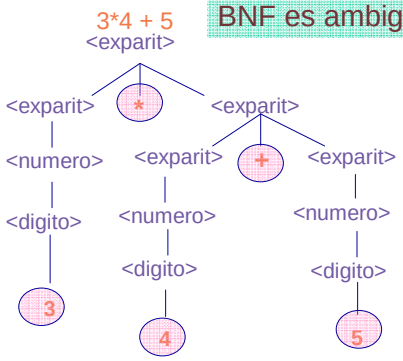
<numero> :: =<digito> |<digito><numero>

<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 (o dos derivaciones más a la izquierda diferentes)**.

• Una **GLC G** es **no ambigua** si **toda cadena** de L(G) **tiene un único árbol de derivación (o una única derivación más a la izquierda).**

• 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.

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.

BNF no ambiguo

Ejemplo: BNF no ambiguo para expresiones aritméticas "simplificadas"

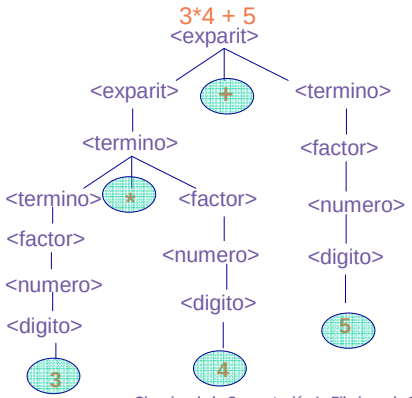
<exparit> :: = <exparit> **+** <termino> | <exparit> **-** <termino> |<termino>

<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

BNF Extendido

- Usa nuevas reglas y símbolos.

Sintaxis	Significado
::=	se define como
t	el símbolo terminal t
<nt>	el símbolo no terminal nt
(...)	usado para agrupar
*	cero o más repeticiones del elemento anterior
+	una o más repeticiones del elemento anterior
[...]	elemento opcional
|	alternativa de varias formas sintácticas válidas

Nota: Para distinguir los metasímbolos de los terminales con el mismo caracter se usa ´
(por ej., si el lenguaje que describe el BNF incluye los paréntesis deben estar precedidos por ´)

BNF Extendido

Ejemplos:

L_1 = { x / x ∈ { begin, end }* y x es una cadena de begin end balanceados }

BNF para L_1

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

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

BNF Extendido para L_1

<lista> :: = <anidados> <anidados>* ó <lista> :: = <anidados>+

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