

# Tema 6: Propiedades de los Lenguajes Independientes del Contexto

Serafín Moral

Universidad de Granada

Enero, 2013

- Lema de bombeo
- Operaciones con lenguajes independientes del contexto
  - Cerradas: Unión, Concatenación, Clausura
  - No cerradas: Intersección, Complementario
- Algoritmos para gramáticas independientes del contexto
  - Algoritmos de pertenencia
    - Algoritmo de Cocke, Younger, Kasami
    - Algoritmo de Early
- Problemas indecidibles

## Lema

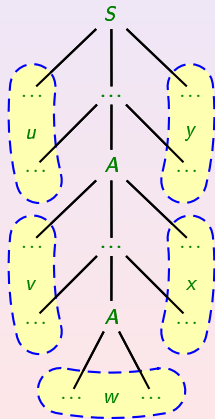
Sea  $L$  un lenguaje independiente del contexto. Entonces, existe una constante  $n$ , que depende solo de  $L$ , tal que si  $z \in L$  y  $|z| \geq n$ ,  $z$  se puede escribir de la forma  $z = uvwxy$  verificando:

- 1  $|vx| \geq 1$
- 2  $|vwx| \leq n$
- 3  $\forall i \geq 0, uv^iwx^iy \in L$

# Demostración (idea)

Supongamos una gramática sin producciones nulas ni unitarias que genera el lenguaje. Si no permitimos que en las sustituciones de una variable  $A$  vuelva a repetirse alguna vez la misma variable  $A$  (excepto si esta variable aparece como raíz), entonces el número de palabras obtenidas es finito. Sea  $L'$  el conjunto de esas palabras y  $n$  igual a la longitud máxima de las palabras de  $L'$  más uno.

Si  $|u| \geq n$ , entonces necesariamente se da una repesión en el árbol de  $u$  de una variable  $A$  como descendiente de ella misma.



Esta es la partición que hace o se verifique el teorema.

## Aplicación

$L = \{a^i b^i c^i \mid i \geq 1\}$  no es independiente del contexto.

Sea  $n$  cualquiera.

Consideremos la palabra  $z = a^n b^n c^n \in L$ .

Supongamos que  $z$  se puede descomponer de la forma  $z = uvwxy$ , verificando las dos primeras condiciones del lema de bombeo:

$|vx| \geq 1$  y  $|vwx| \leq n$ .

No es posible para  $vx$  tener símbolos  $a$  y  $c$  al mismo tiempo y tiene alguno de los símbolos  $a, b, c$ .

En  $uv^2wx^2y$  se añade de uno o dos de los símbolos de la palabra, pero no de los tres. Por tanto  $uv^2wx^2y \notin L$ .

**Conclusión:**  $L$  no es independiente del contexto.

$$L = \{a^i b^j c^k d^l \mid (i = 0) \vee (j = k = l)\}$$

No es independiente del contexto y verifica la condición del lema de bombeo.

$$L_1 = L(G_1), L_2 = L(G_2), \quad G_1 = (V_1, T, P_1, S_1), G_2 = (V_2, T, P_2, S_2)$$

Los lenguajes independientes del contexto son cerrados para las operaciones:

- Unión

$G$  para  $L_1 \cup L_2$

$S \rightarrow S_1, \quad S \rightarrow S_2$  y producciones de las dos gramáticas.

- Concatenación

$G$  para  $L_1 L_2$

$S \rightarrow S_1 S_2$  y producciones de las dos gramáticas.

- Clausura

$G$  para  $L_1^*$ :  $S \rightarrow S_1 S, \quad S \rightarrow \epsilon$  y producciones  $G_1$ .

## Intersección de Lenguajes Independientes del Contexto

La clase de los lenguajes independientes del contexto no es cerrada para la intersección.

$L = \{a^i b^j c^i \mid i \geq 1\}$  no es independiente del contexto.

$L_1 = \{a^i b^j c^j \mid i \geq 1 \text{ y } j \geq 1\}$

$L_2 = \{a^i b^j c^j \mid i \geq 1 \text{ y } j \geq 1\}$  si lo son.

El primero de ellos es generado por la gramática:

$S \rightarrow AB, \quad A \rightarrow aAb \mid ab, \quad B \rightarrow cB \mid c$

y el segundo, por la gramática:

$S \rightarrow CD, \quad C \rightarrow aC \mid a, \quad D \rightarrow bDc \mid bc$

Tenemos que  $L = L_1 \cap L_2$



## Complementario de un Lenguaje Independiente del Contexto

La clase de lenguajes independientes del contexto no es cerrada para el complementario.

Si fuese para el complementario, como ya es cerrada para la unión sería también para la intersección.

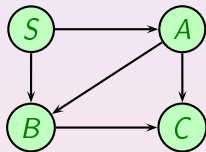
- Ya vimos un algoritmo para determinar si el lenguaje generado por una gramática es **vacío**: En el algoritmo de eliminar símbolos y producciones inútiles, si en la primera parte  $S$  resulta inútil el lenguaje es vacío, y es distinto del vacío en caso contrario.
- Un algoritmo para determinar si el lenguaje generado por una gramática independiente del contexto es **infinito** se puede basar en lo siguiente: se eliminan símbolos y producciones inútiles y producciones nulas y unitarias; entonces se construye un grafo dirigido en el que los nodos son las variables y hay un arco de  $A$  a  $B$  si y solo si tenemos una producción  $A \rightarrow \alpha B \beta$ . El lenguaje es infinito si y solo si este grafo tiene ciclos.

# Ejemplo: finito-infinito

Consideremos la gramática con producciones,

$$S \rightarrow AB, \quad A \rightarrow BC|a, \quad B \rightarrow CC|b, \quad C \rightarrow a$$

El grafo asociado es:



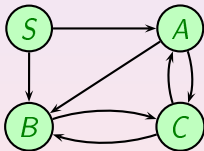
El grafo no tiene ciclos y el lenguaje generado es finito. De hecho sólo se pueden generar las siguientes palabras:  
 $\{ab, aaa, bab, baaa, aaab, aaaaa\}$

# Ejemplo: finito-infinito

Si al ejemplo anterior le añadimos la producción:  $C \rightarrow AB$ ,  
obtendríamos la gramática:

$$S \rightarrow AB, \quad A \rightarrow BC|a, \quad B \rightarrow CC|b, \quad C \rightarrow a, \quad C \rightarrow AB$$

El grafo asociado es:



El grafo tiene ciclos y el lenguaje generado es infinito.

Dada una gramática de tipo 2,  $G = (V, T, P, S)$  y una palabra  $u \in T^*$ , determinar si la palabra puede ser generada por la gramática.

Ya hemos visto algoritmos anteriormente para gramáticas sin producciones nulas ni unitarias o para gramáticas en forma normal de Greibach.

Eran algoritmos de búsqueda en el espacio de todas las posibles derivaciones.

Aquí vamos a ver algoritmos más sofisticados.

- El algoritmo de Cocke-Younger-Kasami para gramáticas en forma normal de Chomsky.
- El algoritmo de Early para gramáticas cualesquiera.

# El Algoritmo de Cocke-Younger-Kasami

- Complejidad  $O(n^3)$ , donde  $n$  es la longitud de la palabra  $u$ .
- Para gramáticas en forma normal de Chomsky.

## Estrategia

Sea  $u_{ij}$  subcadena de  $u$  que comienza en la posición  $i$  y tiene longitud  $j$  ( $j = 1, \dots, n$ ,  $i = 1, \dots, n - j + 1$ ).

Se trata de calcular  $V_{ij}$ : variables que generan  $u_{ij}$ .

- **Condición básica:** Si el  $i$ -ésimo símbolo de  $u$  es  $a$  y tenemos la producción  $A \rightarrow a$ , entonces  $A \in V_{i1}$ .
- **Condición recursiva** ( $j > 1$ ): Si  $A \rightarrow BC$  es una producción y  $B \in V_{ik}$ ,  $C \in V_{i+k, j-k}$ , entonces  $A \in V_{ij}$ .

$$A \Rightarrow BC \Rightarrow u_{ik} u_{i+k, j-k} = u_{ij}$$

1. Para  $i = 1$  hasta  $n$
2. Calcular  $V_{i1} = \{A \mid A \rightarrow a \text{ es una produccion y el simbolo } i\text{-esimo de } u \text{ es } a\}$
3. Para  $j = 2$  hasta  $n$
4. Para  $i = 1$  hasta  $n - j + 1$
5.  $V_{ij} = \emptyset$
6. Para  $k = 1$  hasta  $j - 1$   
 $V_{ij} = V_{ij} \cup \{A \mid A \rightarrow BC \text{ es una produccion, } B \in V_{ik} \text{ y } C \in V_{i+k, j-k}\}$

*Se calcula para todo  $i, j (j \in \{1, \dots, n\}, i \leq n - j + 1)$ , el conjunto de variables  $V_{ij}$  que generan  $u_{ij}$ .*

# Ejemplo

$S \rightarrow AB|BC$ ,  $A \rightarrow BA|a$ ,  $B \rightarrow CC|b$ ,  $C \rightarrow AB|a$

Comprobar la pertenencia de las palabras  $baaba, aaaaa$

## Representación Gráfica

$u_{11}$	$u_{21}$	$u_{31}$	$u_{41}$	$u_{51}$
$V_{11}$	$V_{21}$	$V_{31}$	$V_{41}$	$V_{51}$
$V_{12}$	$V_{22}$	$V_{32}$	$V_{42}$	
$V_{13}$	$V_{23}$	$V_{33}$		
$V_{14}$	$V_{24}$			
$V_{15}$				



## Ejemplo. Cálculo segunda fila

$S \rightarrow AB|BC$ ,  $A \rightarrow BA|a$ ,  $B \rightarrow CC|b$ ,  $C \rightarrow AB|a$

Comprobar la pertenencia de las palabras  $baaba, aaaaa$

### Representación Gráfica

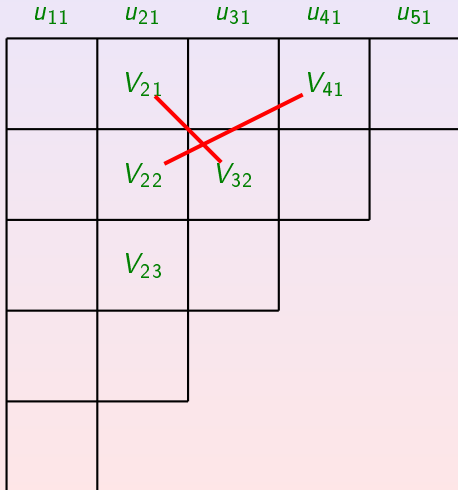
$u_{11}$	$u_{21}$	$u_{31}$	$u_{41}$	$u_{51}$
	$V_{21}$	$V_{21}$		
	$V_{22}$			

## Ejemplo. Cálculo tercera fila

$S \rightarrow AB|BC$ ,  $A \rightarrow BA|a$ ,  $B \rightarrow CC|b$ ,  $C \rightarrow AB|a$

Comprobar la pertenencia de las palabras  $baaba, aaaaa$

### Representación Gráfica

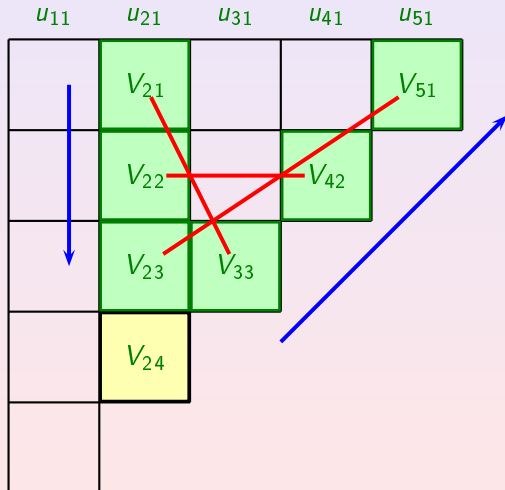


## Ejemplo. Fila Cuarta

$S \rightarrow AB|BC$ ,  $A \rightarrow BA|a$ ,  $B \rightarrow CC|b$ ,  $C \rightarrow AB|a$

Comprobar la pertenencia de las palabras  $baaba, aaaaa$

### Representación Gráfica



# Ejemplo

$S \rightarrow AB, S \rightarrow BC, A \rightarrow BA, A \rightarrow a,$   
 $B \rightarrow CC, B \rightarrow b, C \rightarrow AB, C \rightarrow a$

	b	a	a	b	a
	B	A C	A C	B	A C
	A S	B	S C	A S	
0		B	B		
0		S C A			
S C	A C				

La palabra *baaba* es generada

# Ejemplo

$S \rightarrow AB, S \rightarrow BC, A \rightarrow BA, A \rightarrow a,$   
 $B \rightarrow CC, B \rightarrow b, C \rightarrow AB, C \rightarrow a$

a		a		a		a		a	
A	C	A	C	A	C	A	C	A	C
B		B		B		B			
S	C	S	C	S	C				
A		A		A					
B		B							
S	C								
A									

La palabra **aaaaa** es generada

# El algoritmo de Early

## Propiedades

Es un algoritmo para el problema de la pertenencia que se puede aplicar a toda gramática sin producciones nula ni unitarias. Está basado en programación dinámica. Es  $O(n^3)$  en general, pero  $O(n^2)$  para gramáticas no ambiguas y lineal en muchas gramáticas de uso común.

Supondremos que  $u[i..j]$  es la subcadena de  $u$  que va de la posición  $i$  a la posición  $j$ .

El algoritmo producirá registros de la forma  $(i, j, A, \alpha, \beta)$ , donde  $i$  y  $j$  son enteros y  $A \rightarrow \alpha\beta$  es una producción de la gramática.

Un registro indicará un hecho y un objetivo. El hecho es que  $u[i+1..j]$  es derivable a partir de  $\alpha$  y el objetivo es encontrar todos los  $k$  tales que  $\beta$  deriva a  $u[j+1..k]$ . Si encontramos uno de estos  $k$  sabemos que  $A$  deriva  $u[i+1..k]$ .

Para cada  $j$ ,  $REGISTROS[j]$  contendrá todos los registros existentes de la forma  $(i, j, A, \alpha, \beta)$ .

**P1** *Inicialización.*- Sea

- $REGISTROS[0] = \{(0, 0, S, \epsilon, \beta) : S \rightarrow \beta \text{ es una producción}\}$
- $REGISTROS[j] = \emptyset$  para  $j = 1, \dots, n$ .
- $j = 0$

**P2** *Clausura.*- Para cada registro  $(i, j, A, \alpha, B\gamma)$  en  $REGISTROS[j]$  y cada producción  $B \rightarrow \delta$ , crear el registro  $(j, j, B, \epsilon, \delta)$  e insertarlo en  $REGISTROS[j]$ . Repetir la operación recursivamente para los nuevos registros insertados.

**P3** *Avance.*- Para cada registro  $(i, j, A, \alpha, c\gamma)$  en  $REGISTROS[j]$ , donde  $c$  es un símbolo terminal que aparece en la posición  $j+1$  de  $u$ , crear  $(i, j+1, A, \alpha c, \gamma)$  e insertarlo en  $REGISTROS[j+1]$ .

Hacer  $j = j + 1$ .

# Algoritmo de Early (II)

- P4** *Terminación.*- Para cada par de registros de la forma  $(h, i, B, \gamma, A\delta)$  en  $REGISTROS[i]$  y  $(i, j, A, \alpha, \epsilon)$  en  $REGISTROS[j]$ , crear el nuevo registro  $(h, j, B, \gamma A, \delta)$  e insertarlo en  $REGISTROS[j]$ .
- P5** Si  $j < n$  ir a **P2**.
- P6** Si en  $REGISTROS[n]$  hay un registro de la forma  $(0, n, S, \alpha, \epsilon)$ , entonces  $u$  es generada. En caso contrario no es generada.



# Ejemplo

$S \rightarrow AB, S \rightarrow BC, A \rightarrow BA, A \rightarrow a, B \rightarrow CC, B \rightarrow b, C \rightarrow AB, C \rightarrow a$

Palabra *baa*.

*REGISTROS*[0] : (0,0,S, $\epsilon$ ,AB), (0,0,S, $\epsilon$ ,BC), (0,0,A, $\epsilon$ ,BA),  
(0,0,A, $\epsilon$ ,a), (0,0,B, $\epsilon$ ,CC), (0,0,B, $\epsilon$ ,b), (0,0,C, $\epsilon$ ,AB), (0,0,C, $\epsilon$ ,a),

*REGISTROS*[1] : (0,1,B,b, $\epsilon$ ), (0,1,S,B,C), (0,1,A,B,A),  
(1,1,C, $\epsilon$ ,AB),

(1,1,C, $\epsilon$ ,a), (1,1,A, $\epsilon$ ,BA), (1,1,A, $\epsilon$ ,a), (1,1,B, $\epsilon$ ,CC), (1,1,B, $\epsilon$ ,b)

*REGISTROS*[2] : (1,2,C,a, $\epsilon$ ), (1,2,A,a, $\epsilon$ ), (0,2,S,BC, $\epsilon$ ), (0,2,A,BA, $\epsilon$ ),  
(1,2,C,A,B), (1,2,B,C,C), (0,2,S,A,B), (0,2,C,A,B), (2,2,B, $\epsilon$ ,CC),  
(2,2,B, $\epsilon$ ,b), (2,2,C, $\epsilon$ ,AB), (2,2,C, $\epsilon$ ,a), (2,2,A, $\epsilon$ ,BA), (2,2,A, $\epsilon$ ,a)

*REGISTROS*[3] : (2,3,C,a, $\epsilon$ ), (2,3,A,a, $\epsilon$ ), (1,3,B,CC, $\epsilon$ ), (2,3,B,C,C)  
(2,3,C,A,B), (1,3,A,B,A)

Como (0,3,S, $\alpha$ , $\epsilon$ ) no está en *REGISTROS*[3], la palabra *baa* no es generada

# Ejemplo

$S \rightarrow T, \quad S \rightarrow S + T, \quad T \rightarrow F, \quad T \rightarrow T * F, \quad F \rightarrow a, \quad F \rightarrow b, \quad F \rightarrow (S)$

Palabra:  $(a + b) * a$

**REGISTROS**[0] :  $(0, 0, S, \epsilon, T), (0, 0, S, \epsilon, S + T), (0, 0, T, \epsilon, F), (0, 0, T, \epsilon, T * F),$   
 $(0, 0, F, \epsilon, a), (0, 0, F, \epsilon, b), (0, 0, F, \epsilon, (S))$

**REGISTROS**[1] :  $(0, 1, F, (, S)), (1, 1, S, \epsilon, T), (1, 1, S, \epsilon, S + T), (1, 1, T, \epsilon, F),$   
 $(1, 1, T, \epsilon, T * F), (1, 1, F, \epsilon, a), (1, 1, F, \epsilon, b), (1, 1, F, \epsilon, (S)),$

**REGISTROS**[2] :  $(1, 2, F, a, \epsilon),$   
 $(1, 2, T, F, \epsilon), (1, 2, S, T, \epsilon), (1, 2, T, T, *F), (0, 2, F, (S, )), (1, 2, S, S, +T)$

**REGISTROS**[3] :  $(1, 3, S, S +, T),$   
 $(3, 3, T, \epsilon, F), (3, 3, T, \epsilon, T * F), (3, 3, F, \epsilon, a), (3, 3, F, \epsilon, b), (3, 3, F, \epsilon, (S))$

**REGISTROS**[4] :  $(3, 4, F, b, \epsilon), (3, 4, T, F, \epsilon), (1, 4, S, S + T, \epsilon),$   
 $(3, 4, T, T, *F), (0, 4, F, (S, )), (1, 4, S, S, +T)$

**REGISTROS**[5] :  $(0, 5, F, (S, \epsilon)), (0, 5, T, F, \epsilon), (0, 5, S, T, \epsilon), (0, 5, T, T, *F),$   
 $(0, 5, S, S, +T),$

**REGISTROS**[6] :  $(0, 6, T, T *, F), (6, 6, F, \epsilon, a), (6, 6, F, \epsilon, b), (6, 6, F, \epsilon, (S)),$

**REGISTROS**[7] :  $(6, 7, F, a, \epsilon), (0, 7, T, T * F, \epsilon), (0, 7, S, T, \epsilon), (0, 7, T, T, *F),$   
 $(0, 7, S, S, +T)$

Como tenemos  $(0, 7, S, T, \epsilon)$ , entonces la palabra  $(a + b) * c$  es generada.

Suponemos que  $G$ ,  $G_1$  y  $G_2$  son gramáticas independientes del contexto dadas y  $R$  es un lenguaje regular.

- Saber si  $L(G_1) \cap L(G_2) = \emptyset$ .
- Determinar si  $L(G) = T^*$ , donde  $T$  es el conjunto de símbolos terminales.
- Comprobar si  $L(G_1) = L(G_2)$ .
- Determinar si  $L(G_1) \subseteq L(G_2)$ .
- Determinar si  $L(G_1) = R$ .
- Comprobar si  $L(G)$  es regular.
- Determinar si  $G$  es ambigua.
- Conocer si  $L(G)$  es inherentemente ambiguo.
- Comprobar si  $L(G)$  es determinista.