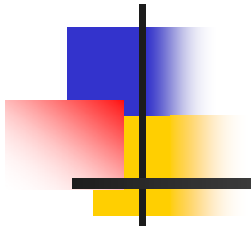


7. Propiedades de los LIC

7.1. Formas normales para las GIC

Fernando Rosa Velardo

Traducción y adaptación de transparencias de Ananth Kalyanaraman
(http://www.eecs.wsu.edu/~ananth/)

“Simplificación” de GICs

Formas de simplificar/limpiar una GIC

(limpiar)

1. Eliminar *símbolos inútiles*

(simplificar)

2. Eliminar producciones ε $A \not\rightarrow \varepsilon$

3. Eliminar producciones *unitarias* $A \not\rightarrow B$

Eliminación de *símbolos inútiles*

Un símbolo X es *alcanzable* si existe:

- $S \Rightarrow^* \alpha X \beta$

Un símbolo es *generador si existe*:

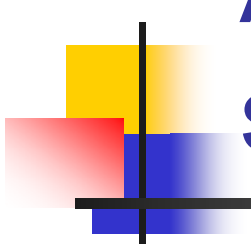
- $X \Rightarrow^* w$,
  - para $w \in T^*$

Para que un símbolo X sea "útil", ha de ser tanto alcanzable como generador

- $S \Rightarrow^* \alpha X \beta \Rightarrow^* w'$, para $w' \in T^*$

alcanzable generador

Algoritmo de eliminación de símbolos inútiles



1. Primero eliminamos los símbolos no generadores

2. Después eliminamos los símbolos no alcanzables

¡El orden importa!

Ejemplo: símbolos inútiles

- S → AB | a
- A → b

1. *A, S* son generadores
2. *B* no es generador (y por lo tanto, es inútil)
3. ==> Eliminamos las producciones en las que aparezca B
   1. S → a
   2. A → b
4. Ahora, A no es alcanzable, y por lo tanto es inútil
5. Simplificada:
   1. S → a

¿Y si comprobamos antes la alcanzabilidad?

No eliminamos:
A → b

$X \Rightarrow^* w$

Algoritmo para encontrar los generadores

- Dada: G=(V,T,P,S)
- Base:
  - Todo símbolo terminal es generador.
- Inducción:
  - Si existe una producción $A \rightarrow \alpha$, y todos los símbolos de α son generadores
  - Entonces A también es generador

$S \Rightarrow^* \alpha X \beta$

Algoritmo para encontrar los alcanzables

- Dada: G=(V,T,P,S)
- Base:
  - S es alcanzable
- Inducción:
  - Si existe una producción $A \rightarrow x_1 x_2 \ldots x_k$, donde A es alcanzable
  - Entonces los símbolos $x_1, x_2, \ldots x_k$ son alcanzables

¿Para qué eliminar transiciones ε?

$A \rightarrow \varepsilon$

Eliminación de producciones ε

Ojo: no es posible eliminar producciones ε en lenguajes que contienen ε

Hablaremos del *resto del lenguaje*

Teorema: Si G=(V,T,P,S) es una GIC para L entonces L-{ε} tiene una GIC sin producciones ε

Definición: A es "anulable" si A=> ε*

- Si A es anulable entonces una producción de la forma B → CAD se puede reemplazar por:
  - B → CD | CAD
    - Lo que nos permite eliminar las transiciones ε para A

Algoritmo para detectar símbolos anulables

- Base:
  - Si existe la producción $A \rightarrow \varepsilon$ entonces A es anulable (aunque existan otras producciones para A)
- Inducción:
  - Si existe $B \rightarrow C_1C_2\ldots C_k$, y *todos* los C_i son anulables entonces B es anulable

Eliminación de producciones ε

Dada: G=(V,T,P,S)

Algoritmo:

1. Detectar todas las variables anulables de G
2. Construir $G_1=(V,T,P_1,S)$ de la siguiente manera:
   i. Para cada producción $A \rightarrow X_1X_2\ldots X_k$, donde k≥1, si ***m*** de los ***k*** X_i's son anulables
   ii. Entonces G_1 tiene 2^m versiones de esta producción
      i. Todas las combinaciones para las que cada X_i anulable aparece o no
   iii. Eliminar todas las producciones $A \rightarrow \varepsilon$

Ejemplo: eliminación de producciones ε

- Sea L el lenguaje dado por la GIC:
  i. $S \rightarrow AB$
  ii. $A \rightarrow aAA \mid \varepsilon$
  iii. $B \rightarrow bBB \mid \varepsilon$

Objetivo: Construir G1, gramática sin producciones ε, para L-{ε}

- Símbolos anulables: {S, A, B}
- G_1:
  - $B \rightarrow b \mid bB \mid bB \mid bBB$
    - ==> $B \rightarrow b \mid bB \mid bBB$
  - análogamente, $A \rightarrow a \mid aA \mid aAA$
  - análogamente, $S \rightarrow A \mid B \mid AB$
- Obs: $L(G) = L(G_1) \cup \{\varepsilon\}$

G_1:
- $S \rightarrow A \mid B \mid AB$
- $A \rightarrow a \mid aA \mid aAA$
- $B \rightarrow b \mid bB \mid bBB$

Eliminación de producciones unitarias

$A \not\rightarrow B$ ← B variable

¿Para qué eliminar producciones unitarias?

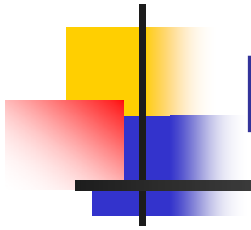
Menos pasos de derivación

Ej.

antes	después
A → B | …	A → xxx | yyy | zzz | …
B → C | …	B → xxx | yyy | zzz | …
C → D | …	C → xxx | yyy | zzz | …
D → xxx | yyy | zzz	D → xxx | yyy | zzz

Eliminación $A \rightarrow B$ de producciones unitarias

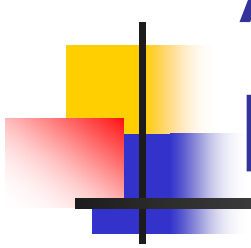
- Una producción unitaria es de la forma A → B, donde B es una variable
- Por ejemplo,
  1. E → T | E+T
  2. T → F | T*F
  3. F → I | (E)
  4. I → a | b | Ia | Ib | I0 | I1
- ¿Cómo eliminamos producciones unitarias?
  - Reemplazamos E → T por E → F | T*F
  - Y continuamos recursivamente mientras haya producciones unitarias:
    - E → F | T*F | E+T
    - E → I | (E) | T*F| E+T
    - E → a | b | Ia | Ib | I0 | I1 | (E) | T*F | E+T
    - Ahora E no tiene producciones unitarias
  - Igual para el resto de las producciones unitarias



Pares unitarios: algoritmo

- (A,B) es un ***"par unitario"*** si $A =>^{*} B$
- Algoritmo para detectar pares unitarios:
  - Base: Todos los pares (A,A) son unitarios. Si $A \rightarrow B$ es una producción, (A,B) es un par unitario.
  - Inducción: Si (A,B) es un par unitario y $A \rightarrow C$ es una producción entonces (A,C) es un par unitario.

Algoritmo para eliminar producciones unitarias



Entrada: G=(V,T,P,S)

Salida: G_1=(V,T,P_1,S) equivalente, sin producciones unitarias

Algoritmo:

1. Encuentra todos los pares unitarios
2. Para cada (A,B) unitario:

   Añadimos la producción $A \rightarrow \alpha$, para cada producción no unitaria $B \rightarrow \alpha$

Ejemplo: eliminación de producciones unitarias

G:

1. E → T | E+T
2. T → F | T*F
3. F → I | (E)
4. I → a | b | Ia | Ib | I0 | I1

Pares unitarios	*Producciones añadidas*
(E,E)	E → E+T
(E,T)	E → T*F
(E,F)	E → (E)
(E,I)	E → a|b|Ia | Ib | I0 | I1
(T,T)	T → T*F
(T,F)	T → (E)
(T,I)	T → a|b| Ia | Ib | I0 | I1
(F,F)	F → (E)
(F,I)	F → a| b| Ia | Ib | I0 | I1
(I,I)	I → a| b | Ia | Ib | I0 | I1

G_1:

1. E → E+T | T*F | (E) | a| b | Ia | Ib | I0 | I1
2. T → T*F | (E) | a| b | Ia | Ib | I0 | I1
3. F → (E) | a| b | Ia | Ib | I0 | I1
4. I → a | b | Ia | Ib | I0 | I1

Todo junto…

- Teorema: Si G es una GIC que genera un lenguaje L (no vacío ni {ε}), existe otra GIC G_1 tal que $L(G_1)=L(G) - \{\varepsilon\}$, *y* G_1:
  - no tiene producciones ε
  - no tiene producciones unitarias
  - no tiene símbolos inútiles

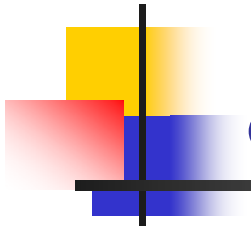
- Algoritmo:

  1) eliminar producciones ε
  2) eliminar producciones unitarias
  3) eliminar símbolos inútiles

De nuevo, ¡el orden importa!

¿Por qué?

Formas normales



¿Para qué formas normales?

- Si se sabe que las producciones de la gramática tienen cierta forma, entonces:
  a. Es más sencillo diseñar algoritmos para gramáticas
  b. Es más sencillo probar cosas

Forma Normal de Chomsky (FNC)

Sea G una GIC.

Definición:

*G está en **FNC** si todas sus producciones son de la forma:*

i. $A \rightarrow BC$ *donde A,B,C son variables, o*

ii. $A \rightarrow a$ *donde a es un terminal*

- *Además, G no tiene símbolos inútiles*

Por i y ii:

=> no puede contener producciones unitarias ni producciones ε)

¿Está la siguiente GIC en FNC?

G_1:

1. $E \rightarrow E+T \mid T*F \mid (E) \mid Ia \mid Ib \mid I0 \mid I1$
2. $T \rightarrow T*F \mid (E) \mid Ia \mid Ib \mid I0 \mid I1$
3. $F \rightarrow (E) \mid Ia \mid Ib \mid I0 \mid I1$
4. $I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

- G no tiene producciones ε
- G no tiene producciones unitarias
- G no tiene símbolos inútiles
- Pero…
  - las producciones no están en FNC

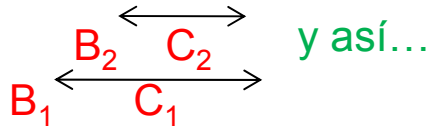
Conversión de G en FNC

- Hipótesis: G no tiene producciones ε ni unitarias, ni símbolos inútiles

1) Para cada terminal ***a*** que aparezca en el cuerpo de una producción junto con otros símbolos:
   i. creamos una única variables X_a, y una producción $X_a \rightarrow a$, y
   ii. Reemplazamos las demás apariciones de a (acompañada de más símbolos) por X_a

2) Ahora todas las producciones tienen la forma:
   - $A \rightarrow B_1B_2\ldots B_k$ $(k\geq 3)$ o $A \rightarrow a$

3) Reemplazamos cada producción $A \rightarrow B_1B_2B_3\ldots B_k$ por:



   - $A \rightarrow B_1C_1$ $C_1 \rightarrow B_2C_2$ … $C_{k-3} \rightarrow B_{k-2}C_{k-2}$ $C_{k-2} \rightarrow B_{k-1}B_k$

Ejemplo

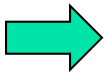
G:

$S \rightarrow AS \mid BABC$

$A \rightarrow A1 \mid 0A1 \mid 01$

$B \rightarrow 0B \mid 0$

$C \rightarrow 1C \mid 1$



G en FNC:

$X_0 \rightarrow 0$

$X_1 \rightarrow 1$

$S \rightarrow AS \mid BY_1$

$Y_1 \rightarrow AY_2$

$Y_2 \rightarrow BC$

$A \rightarrow AX_1 \mid X_0Y_3 \mid X_0X_1$

$Y_3 \rightarrow AX_1$

$B \rightarrow X_0B \mid 0$

$C \rightarrow X_1C \mid 1$

Todas las producciones tiene la forma correcta

Ejemplo

G:

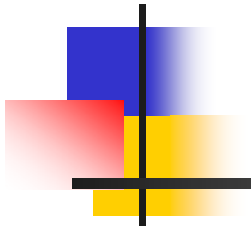
1. $E \rightarrow E+T \mid T*F \mid (E) \mid Ia \mid Ib \mid I0 \mid I1$
2. $T \rightarrow T*F \mid (E) \mid Ia \mid Ib \mid I0 \mid I1$
3. $F \rightarrow (E) \mid Ia \mid Ib \mid I0 \mid I1$
4. $I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

Paso (1)

1. $E \rightarrow EX_{+}T \mid TX_{*}F \mid X_{(}EX_{)} \mid IX_a \mid IX_b \mid IX_0 \mid IX_1$
2. $T \rightarrow TX_{*}F \mid X_{(}EX_{)} \mid IX_a \mid IX_b \mid IX_0 \mid IX_1$
3. $F \rightarrow X_{(}EX_{)} \mid IX_a \mid IX_b \mid IX_0 \mid IX_1$
4. $I \rightarrow X_a \mid X_b \mid IX_a \mid IX_b \mid IX_0 \mid IX_1$
5. $X_{+} \rightarrow +$
6. $X_{*} \rightarrow *$
7. $X_{+} \rightarrow +$
8. $X_{(} \rightarrow ($
9. .....

Paso (2)

1. $E \rightarrow EC_1 \mid TC_2 \mid X_{(}C_3 \mid IX_a \mid IX_b \mid IX_0 \mid IX_1$
2. $C_1 \rightarrow X_{+}T$
3. $C_2 \rightarrow X_{*}F$
4. $C_3 \rightarrow EX_{)}$
5. $T \rightarrow$.....
6. ....



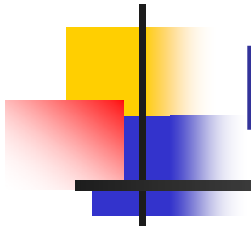
7. Propiedades de los LIC

7.2. Lema de bombeo para los LIC

Fernando Rosa Velardo

Traducción y adaptación de transparencias de Ananth Kalyanaraman
(http://www.eecs.wsu.edu/~ananth/)

El regreso del lema de bombeo…



- Resultado útil para probar que algunos lenguajes NO son incontextuales
  - (como en el caso de los lenguajes regulares)

- Antes de enunciar el lema de bombeo,
  - Una propiedad acerca de los árboles de derivación de GIC en FNC

Profundidad de los árboles de derivación

Obs: los árboles de derivación de una GIC en FNC son árboles binarios

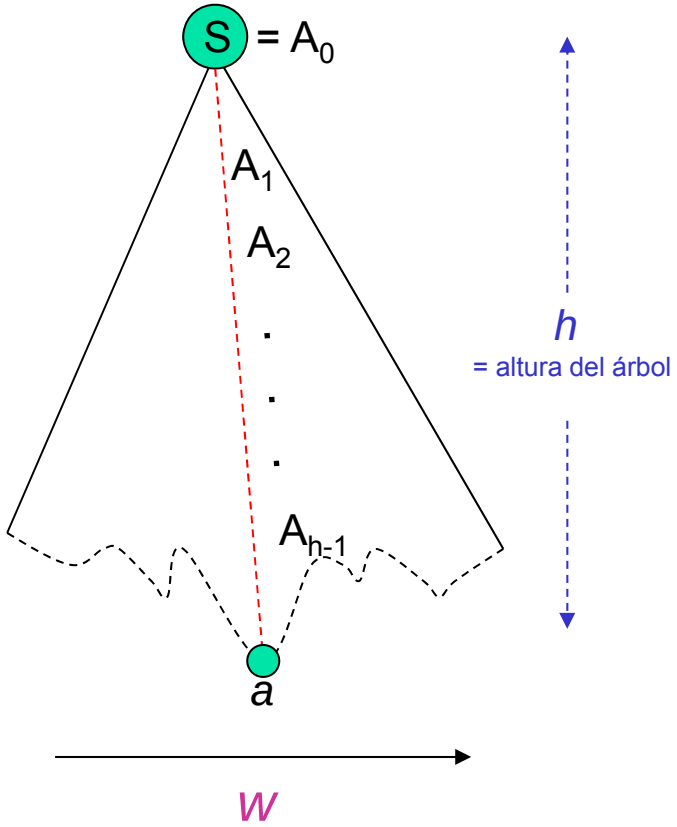
Dados:

- Un árbol de derivación para ***w***, en una GIC G=(V,T,P,S) en FNC
- *h*, la altura del árbol de derivación

Entonces:

- $|w| \leq 2^{h-1}$

Árbol para w



Demostración

Por inducción sobre h

Base: h = 1

- La derivación es "S → a"
- $|w| = 1 = 2^{1-1}$.

Hipótesis de inducción: h = k-1

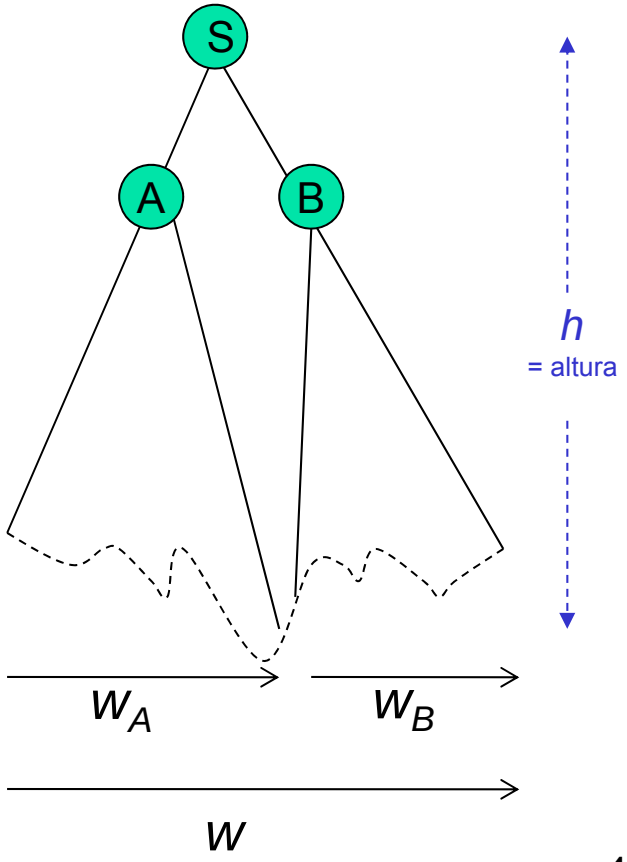
- $|w| \leq 2^{k-2}$

Paso inductivo: h = k

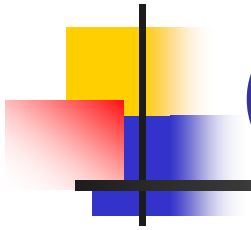
S tiene exactamente dos hijos:
S → AB

- Alturas de los subárboles de A y B a lo sumo h-1
- $w = w_A w_B$, con $|w_A| \leq 2^{k-2}$ y $|w_B| \leq 2^{k-2}$
- $|w| \leq 2^{k-1}$

Árbol de derivación para w



Consecuencias del teorema (para GICs en FNC)

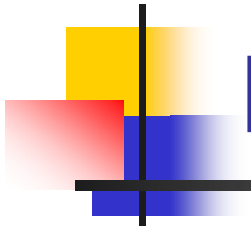


Resultado:

- Si la altura de un árbol de derivación es k,
  - ==> $|w| \leq 2^{k-1}$
- Dicho de otra manera:
  - Un árbol de derivación de altura k no puede generar palabras de longitud mayor que 2^{k-1}

Consecuencia:

- Si $|w| \geq 2^k$ sus árboles de derivación tienen una altura de al menos k+1



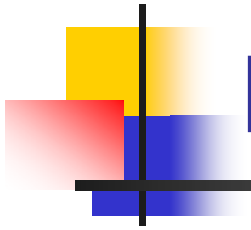
Lema de bombeo para LICs

Sea L un LIC.

Entonces existe una constante N, tal que

- si $z \in L$ con $|z| \geq N$, podemos descomponer $z=uvwxy$, tal que:
  1. $|vwx| \leq N$
  2. $vx \neq \varepsilon$
  3. Para todo $k \geq 0$: $uv^k wx^k y \in L$

Obs: bombeamos en dos sitios



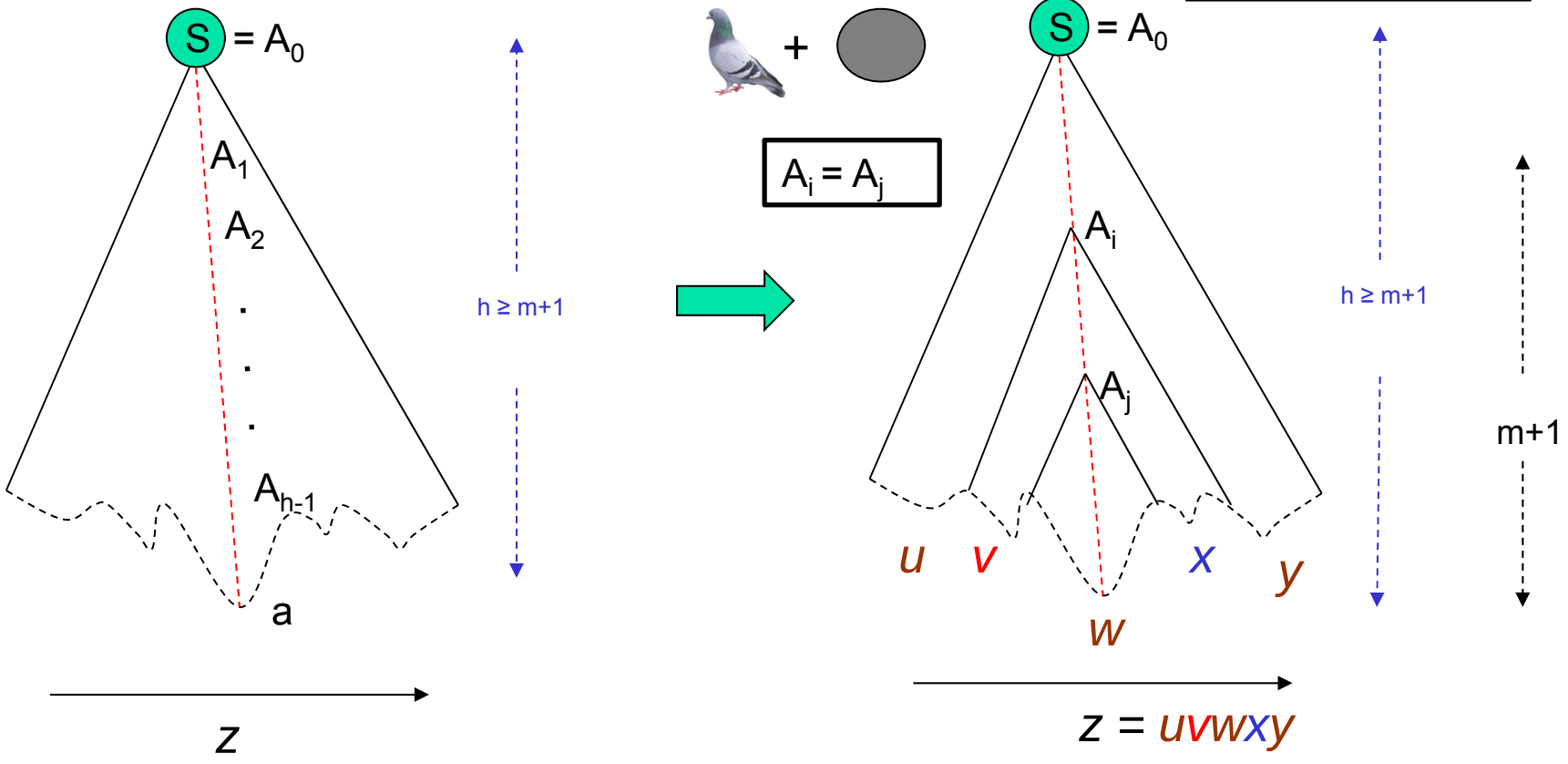
Demostración (sketch)

- Sea L un LIC
  - Sea G una GIC para L en FNC
  - Sea m = número de variables en G
  - Tomamos $N=2^m$.
  - Sea $z \in L$ tal que $|z| \geq N$

    ➔ árbol de derivación de z tiene una altura ≥ m+1
    (resultado anterior)

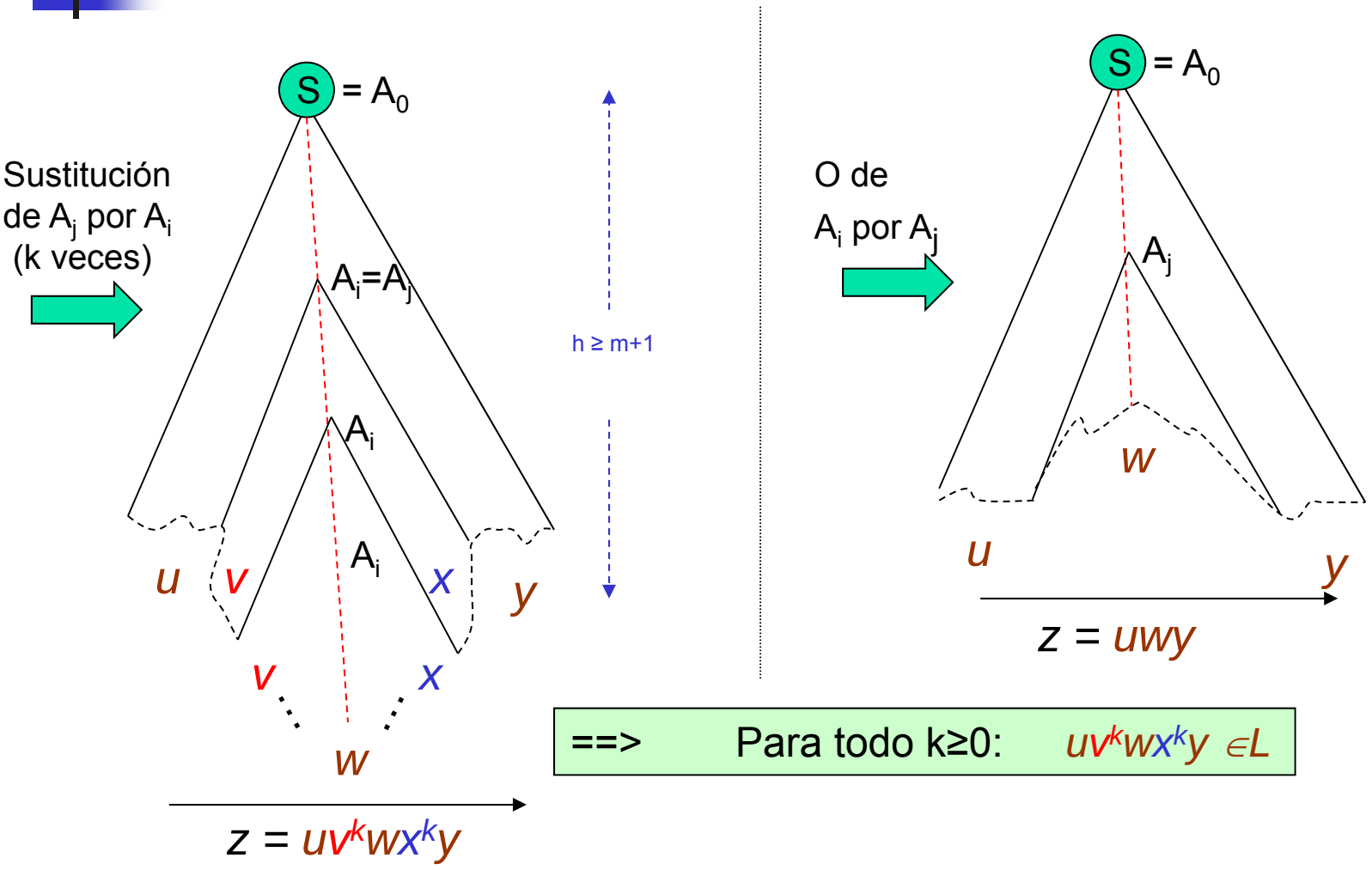
Árbol de derivación de z

Se repite alguna de las últimas m+1 variables



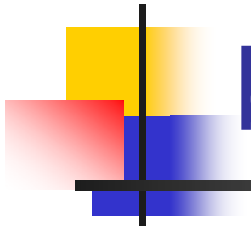
- Por lo tanto, $vx \neq \varepsilon$

Extensiones del árbol de derivación…



==> Para todo k≥0: $uv^kwx^ky \in L$

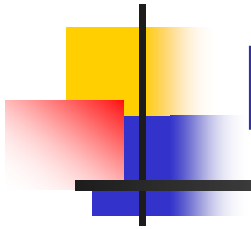
Aplicación del lema de bombeo para LICs



Ejemplo 1: $L = \{a^m b^m c^m \mid m>0 \}$

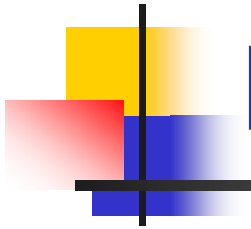
Hipótesis: L no es un LIC

- Sea N = constante del lema de bombeo
- Tomamos $z = a^N b^N c^N$
- $z = uvwxy$
- Como $|vwx| \leq N$ *y* $vx \neq \varepsilon$
  - ==> v, x no pueden contener los tres símbolos (a,b,c)
  - ==> podemos bombear para construir otra palabra que no está en L



Ejemplo #2

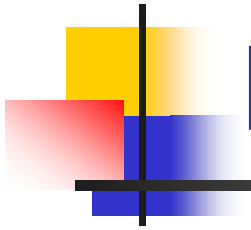
- L = { ww | w en {0,1}*} no es un LIC
  - z = 0^N0^N
    - ¿qué ocurre?
  - z = $0^N1^N0^N1^N$
    - ¿qué ocurre?



Ejemplo 3

- L = { 0^{k^2} | k cualquier entero)

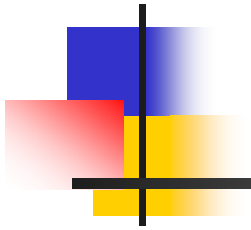
- L no es un LIC



Ejemplo 4

- $L = \{a^i b^j c^k \mid i<j<k \}$

- L no es un LIC

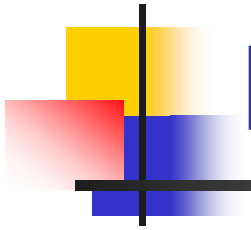


7. Propiedades de los LIC

7.3. Propiedades de clausura de los LIC

Fernando Rosa Velardo

Traducción y adaptación de transparencias de Ananth Kalyanaraman
(http://www.eecs.wsu.edu/~ananth/)

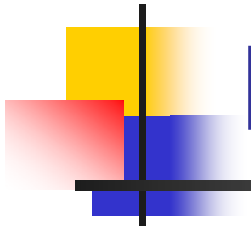


Propiedades de clausura

- LICs cerrados para:
  - Unión
  - Concatenación
  - Clausura de Kleene
  - Reflexión

- LICs NO cerrados para:
  - Intersección
  - Diferencia
  - Complemento

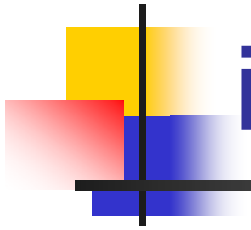
Obs: Los lenguajes regulares sí lo son



LICs cerrados para…

- Sean S_1 y S_2 las variables iniciales de las GICs para L_1 y L_2
  - Unión: $\mathbf{S_{new} \rightarrow S_1 \mid S_2}$
  - Concatenación: $\mathbf{S_{new} \rightarrow S_1\ S_2}$
  - Clausura de Kleene: $\mathbf{S_{new} \rightarrow S_1\ S_{new} \mid \varepsilon}$
  - Reflexión: Sustitiuimos $A \rightarrow \alpha$ por $A \rightarrow \alpha^R$

LICs *no* cerrados para la intersección



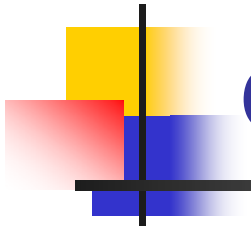
- Demostración constructiva:
  - $L_1 = \{0^n1^n2^i \mid n \geq 1, i \geq 1\}$
  - $L_2 = \{0^i1^n2^n \mid n \geq 1, i \geq 1\}$
- Tanto L_1 como L_2 son LICs
  - ¿Gramáticas?
- Pero $L_1 \cap L_2$ *no* en un LIC
  - $L_1 \cap L_2 = \{0^n1^n2^n \mid n \geq 1\}$

LICs no cerrados para el complemento

- Consecuencia de no ser cerrados para la intersección

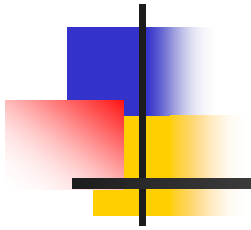
- $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

LICs no cerrados para la diferencia



- Se sigue del hecho de no ser cerrados para el complemento

- Si fuesen cerrados para la diferencia:
  - $\overline{L} = \Sigma^* - L$
  - y L también sería un LIC

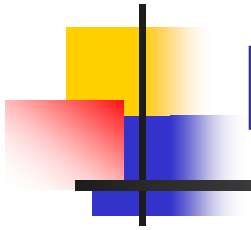


7. Propiedades de los LIC

7.4. Propiedades de decisión de los LIC

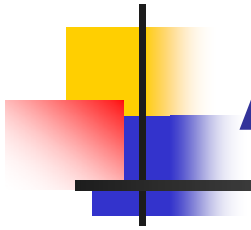
Fernando Rosa Velardo

Traducción y adaptación de transparencias de Ananth Kalyanaraman
(http://www.eecs.wsu.edu/~ananth/)



Propiedades de decisión

- Test de vacío:
  - ¿S generadora?
- Test de pertenencia
  - Ejecución del AP
  - Enumeración de los árboles de derivación de una GIC en FNC
  - Otro algoritmo más eficiente…



Algoritmo CYK

Entrada: G=(V,T,P,S) en FNC, $w=a_1 \ldots a_n$ en T*

Salida: Respuesta a ¿w en L(G)?

Algoritmo:

- X_{ij}=conjunto de variables que generan la palabra $a_i a_{i+1} \ldots a_j$
  - Base: X_{ij}=conjunto de variables A para las que hay una producción $A \rightarrow a_i$
  - Inducción: X_{ij}=conjunto de variables A tal que:
    - Existe una producción $A \rightarrow BC$
    - B pertenece a X_{ik} y C pertenece a $X_{k+1,j}$
- ¿Está S en X_{1n}?

Algoritmo CYK, paso a paso

G:
S → AB | BC
A → BA | a
B → CC | b
C → AB | a

w=baaba

X_{15}				
X_{14}	X_{25}			
X_{13}	X_{24}	X_{35}		
X_{12}	X_{23}	X_{34}	X_{45}	
X_{11}	X_{22}	X_{33}	X_{44}	X_{55}
b	a	a	b	a

Algoritmo CYK, paso a paso

G:
S → AB | BC
A → BA | a
B → CC | b
C → AB | a

w=baaba

b	a	a	b	a

Algoritmo CYK, paso a paso

G:
S → AB | BC
A → BA | a
B → CC | b
C → AB | a

w=baaba

{B}	{A,C}	{A,C}	{B}	{A,C}
b	a	a	b	a

Algoritmo CYK, paso a paso

G:
S → AB | BC
A → BA | a
B → CC | b
C → AB | a

w=baaba

{S,A}	{B}	{S,C}	{S,A}	
{B}	{A,C}	{A,C}	{B}	{A,C}
b	a	a	b	a

Algoritmo CYK, paso a paso

G:
S → AB | BC
A → BA | a
B → CC | b
C → AB | a

w=baaba

-	{B}	{B}		
{S,A}	{B}	{S,C}	{S,A}	
{B}	{A,C}	{A,C}	{B}	{A,C}
b	a	a	b	a

Algoritmo CYK, paso a paso

G:
S → AB | BC
A → BA | a
B → CC | b
C → AB | a

w=baaba

-	{S,A,C}			
-	{B}	{B}		
{S,A}	{B}	{S,C}	{S,A}	
{B}	{A,C}	{A,C}	{B}	{A,C}
b	a	a	b	a

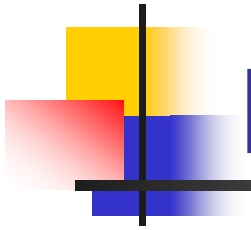
Algoritmo CYK, paso a paso

G:
S → AB | BC
A → BA | a
B → CC | b
C → AB | a

w=baaba

{S,A,C}				
-	{S,A,C}			
-	{B}	{B}		
{S,A}	{B}	{S,C}	{S,A}	
{B}	{A,C}	{A,C}	{B}	{A,C}
b	a	a	b	a

Problemas “indecidibles” para LICs



- ¿Es una GIC dada ambigua?
- ¿Es un LIC dado inherentemente ambiguo?
- ¿Es la intersección de dos LICs vacía?
- ¿Son dos LICs iguales?
- Dada G, ¿L(G)=∑*?