Tema 6: Propiedades de los Lenguajes Independientes del Contexto

Serafín Moral

Universidad de Granada

Diciembre, 2020

Contenido

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

LEMA DE BOMBEO

Lema

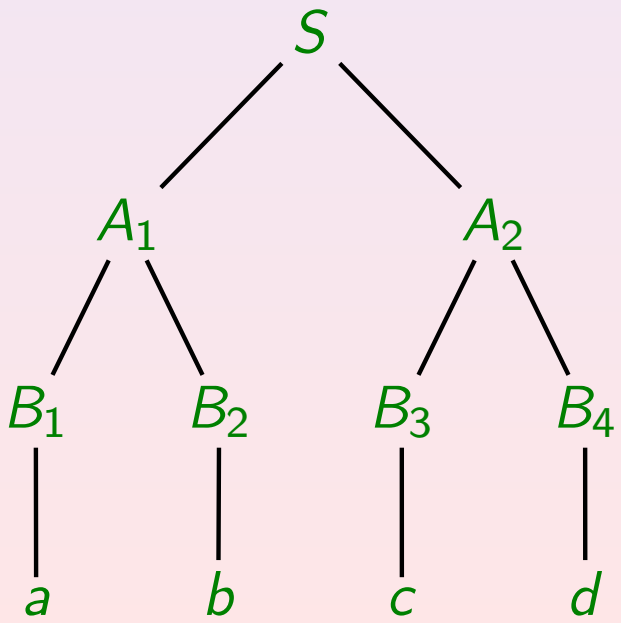
Sea L un lenguaje independiente del contexto. Entonces, existe una constante n, que depende solo de L, tal que si $z \in L$ y $|z| \geq n$, z se puede escribir de la forma $z = uvwxy$ verficando:

1. $|vx| \geq 1$
2. $|vwx| \leq n$
3. $\forall i \geq 0, uv^i wx^i y \in L$

Propiedad Previa

Propiedad

Si en una gramática en forma normal de Chomsky, existe una palabra que es derivada con un árbol de derivación en el que todos los caminos desde la raíz a una hoja son de longitud menor o igual a m, entonces la palabra tiene de longitud menor o igual a 2^{m-1}, además en un camino de la raíz a la hoja de longitud k aparecen k variables.

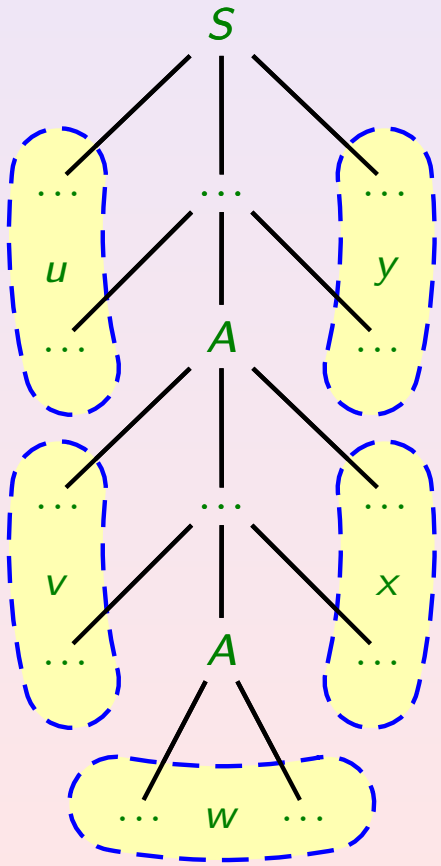


Demostración (idea)

Supongamos una gramática en forma normal de Chomsky que genera el lenguaje y p su número de variables. Sea $n = 2^p$. Si $|u| \geq n$, sea el camino más largo de la raíz a las hojas, entonces necesariamente se da una repetición de la variable en el árbol de u de una variable A como descendiente de ella misma, ya que este camino tiene longitud, al menos, $p+1$ y hay p variables.
Suponemos que A es la primera variable que se repite si empezamos el camino desde la hoja a la raíz.

$|vx| \geq 1$ ya que no hay producciones nulas ni unitarias

$|vwx| \leq n$ ya que la longitud del camino más largo que parte de la A más cercana a la raíz es menor o igual a $p+1$



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

$uv^i wx^i y \in L$, ya que un subárbol con una A se puede sustituir por cualquier otro árbol etiquetado con A

Ejemplo

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.

Contraejemplo

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

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

Operaciones con Lenguajes Ind. del Cont.

$L_1 = L(G_1), L_2 = L(G_2), \quad G_1 = (V_1, T, P_1, S_1), G_2 = (V_2, T, P_2, S_2)$
Los lenguajes independientes del contexto son cerrados para las operaciones:

- Unión
  G para $L_1 \cup L_2$
  $S \to S_1, \quad S \to S_2$ y producciones de las dos gramáticas.
- Concatenación
  G para $L_1 L_2$
  $S \to S_1 S_2$ y producciones de las dos gramáticas.
- Clausura
  G para L_1^*: $\quad S \to S_1 S, \quad S \to \varepsilon$ y producciones G_1.

Intersección de Lenguajes Independientes del Contexto

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

$L = \{a^i b^i c^i | i \geq 1\}$ no es independiente del contexto.
$L_1 = \{a^i b^i c^j | i \geq 1 \text{ y } j \geq 1\}$
$L_2 = \{a^i b^j c^j | i \geq 1 \text{ y } j \geq 1\}$ si lo son.

El primero de ellos es generado por la gramática:
$S \to AB, \quad A \to aAb|ab, \quad B \to cB|c$
y el segundo, por la gramática:
$S \to CD, \quad C \to aC|a, \quad D \to bDc|bc$

Tenemos que $L = L_1 \cap L_2$

Complementario

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.

Complementario de un Lenguaje IC Determinista

Si L es un lenguaje independiente del contexto determinista sobre el alfabeto A, entonces su complementario $\overline{L} = A^* \setminus L$ es independiente del contexto determinista

La demostración no es simple por la presencia de transiciones nulas. Se basa en considerar un APD que acepte L por estados finales completo (se puedan leer las palabras completas).
El problema fundamental es que después de leer la palabra de entrada completa, el autómata tiene que pasar por todas las transiciones nulas posibles y si nunca ha pasado por un estado final, entonces acepta.
Para ello hace tres copias de cada estado q: $(q,0),(q,1),(q,2)$ con el siguiente significado:

- $(q,0)$: está en q, no ha pasado por ningún estado final desde el último símbolo leido y con el contenido actual de la pila, pueden quedar transiciones nulas por hacer.
- $(q,1)$: está en q y si ha pasado por ningún estado final desde el último símbolo leido.
- $(q,2)$: está en q, no ha pasado por ningún estado final desde el último símbolo leido y con el contenido actual de la pila, no quedan transiciones nulas por hacer.

Los finales van a ser $(q,2)$ donde $q \in Q$. El estado inicial es $(q_0,0)$ si $q_0 \notin F$ y $(q_0,1)$ en caso contrario.

Complementario L. Independiente del Contexto Determ. (Cont.)

Las transiciones son como sigue:

- Si $(p,\alpha) \in \delta(q,\varepsilon,X)$ entondes añadimos al autómata nuevo $((p,i),\alpha) \in \delta'((q,j),\varepsilon,X)$ donde $j = 0,1$ e $i = 1$ si $j = 1$ ó $p \in F$ e $i = 0$ en caso contrario.
- Si $(p,\alpha) \in \delta(q,a,X)$ donde $a \in A$, añadimos al autómata $((q,2),X) \in \delta((q,0),\varepsilon,X)$ para indicar que si la palabra termina aquí aceptamos (no hemos pasado por un estado final desde el último símbolo leido).

  También se añade $((p,i),\alpha) \in \delta'((q,j),a,X)$ donde $j = 1,2$ e $i = 0$ si $p \notin F$ e $i = 1$ en caso contrario, para poder seguir leyendo símbolos si la palabra no ha terminado.

El autómata así construido acepta el lenguaje complementario y es determinista.

Nota: Este resultado no implica que la intersección de lenguajes ic deterministas sea determinista, ya que la unión de deterministas no es siempre determinista.

Intersección de un lenguaje independiente del contexto y un lenguaje regular

Resultado

Si L es un lenguaje independiente del contexto y R es un lenguaje regular, entonces $L \cap R$ es independiente del contexto.

Supongamos un autómata con pila $M = (Q, A, B, \delta, q_0, Z_0, F)$ que acepta L por el criterio de estados finales y un autómata finito determinista $M' = (Q', A, \delta', q_0', F')$. Construimos el autómata con pila $M'' = (Q'', A, B, \delta'', q_0'', Z_0, F'')$ de la siguiente forma:

- $Q'' = Q \times Q'$
- $q_0'' = (q_0, q_0')$
- $F'' = F \times F'$
- $\delta''((p,q),a,X) = \{((r,s),\alpha) | (r,\alpha) \in \delta(p,a,X), s = \delta'(q,a)\}$, donde $a \in A$
  $\delta''((p,q),\varepsilon,X) = \{((r,q),\alpha) | (r,\alpha) \in \delta(p,\varepsilon,X)\}$

Este autómata acepta $L \cap R$ y por tanto es independiente del contexto.

Lenguaje no Determinista

Lenguaje $L = \{a^i b^j c^k \mid i \neq j \vee j \neq k\}$ es independiente del contexto pero no es determinista

Si fuese independiente del contexto determinista, entonces $\overline{L}$ sería independiente del contexto determinista.
Sea L_1 el lenguaje asociado a la expresión regular $\mathbf{a^* b^* c^*}$. Entonces $L_2 = \overline{L} \cap L_1$ sería independiente del contexto.
Pero $L_2 = \overline{L} \cap L_1 = \{a^i b^i c^i \mid i \geq 0\}$ que sabemos que no es independiente del contexto.
Luego si $L = \{a^i b^j c^k \mid i \neq j \vee j \neq k\}$ fuese independiente del contexto determinista, llegaríamos a una contradicción.

Algoritmos

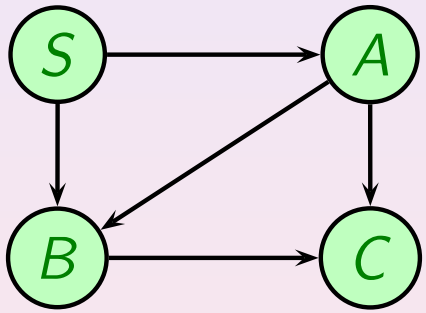
- Ya vimos un algoritmo para determinar si el lenguaje generado por una gramática es vacío: En el algoritmo de eliminar símbolos y producciones inútiles, si en la primera parte S resulta inútil el lenguaje es vacío, y es distinto del vaci en caso contrario.
- Un algoritmo para determinar si el lenguaje 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 $A \to \alpha B \beta$. El lenguaje es infinito si y solo si este grafo tiene ciclos.

Ejemplo: finito-infinito

Consideremos la gramática con producciones,

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

El grafo asociado es:



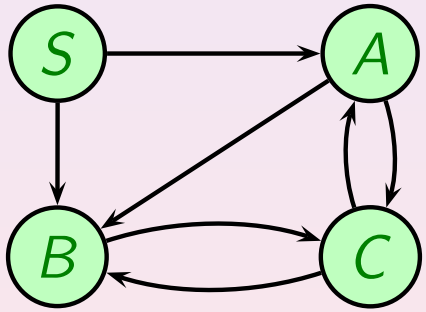
El grafo no tiene ciclos y el lenguaje generado es finito. De hecho sólo se pueden generar las siguientes palabras:
$\{ab, aaa, bab, baaa, aaab, aaaaa\}$

Ejemplo: finito-infinito

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

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

El grafo asociado es:



El grafo tiene ciclos y el lenguaje generado es infinito.

Algoritmos de Pertenencia

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 $u_{i,j}$ subcadena de u que comienza en la posición i y tiene logitud j ($j = 1, \ldots, n, \quad i = 1, \ldots, n-j+1$).
Se trata de calcular V_{ij}: variables que generan $u_{i,j}$.

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

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

Algoritmo

1. Para $i = 1$ hasta n
   2. Calcular $V_{i1} = \{A \mid A \to a$ es una produccion y el simbolo i-esimo de u 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 $j - 1$

         $V_{ij} = V_{ij} \cup \{A \mid A \to BC$ es una produccion, $B \in V_{ik}$ y $C \in V_{i+k,j-k}\}$

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

Ejemplo

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

Comprobar la pertenencia de las palabras $baaba, aaaaa$

Representación Gráfica

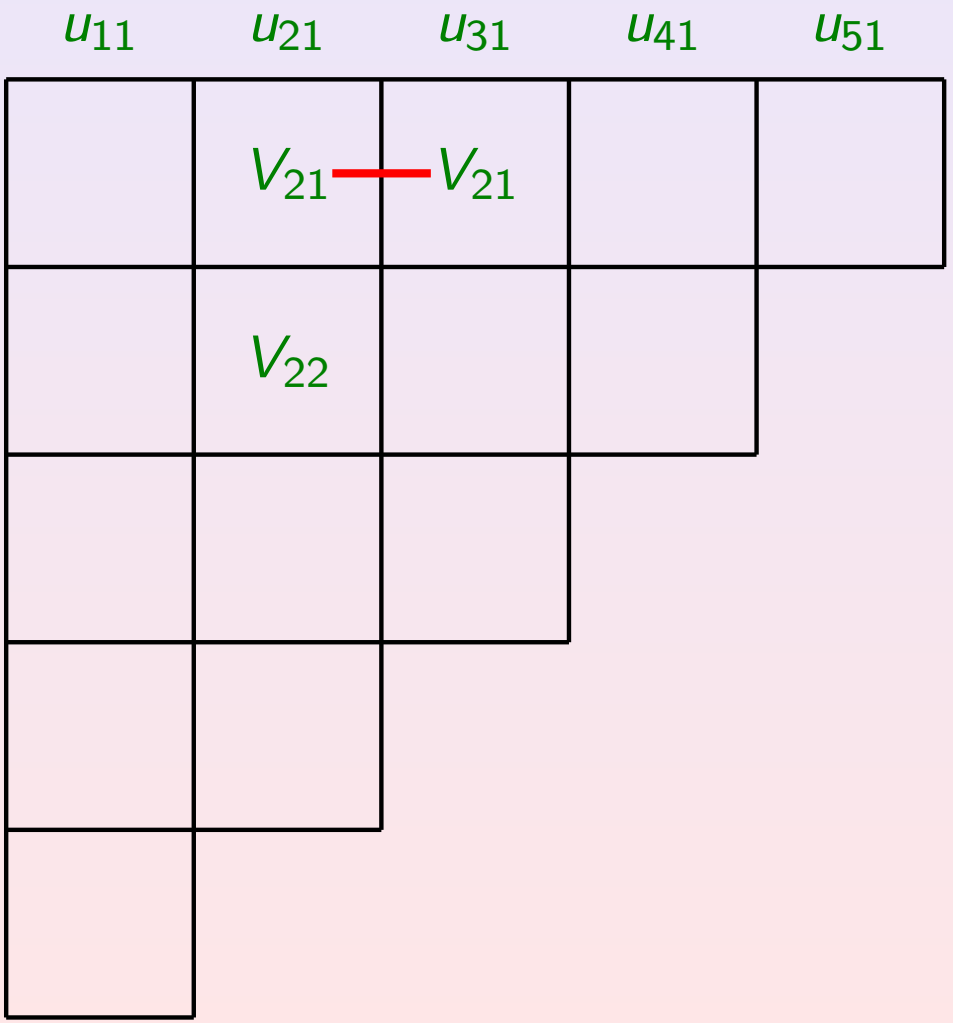
u_{11}	u_{21}	u_{31}	u_{41}	u_{51}
V_{11}	V_{21}	V_{31}	V_{41}	V_{51}
V_{12}	V_{22}	V_{32}	V_{42}	
V_{13}	V_{23}	V_{33}		
V_{14}	V_{24}			
V_{15}				

Ejemplo. Cálculo segunda fila

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

Comprobar la pertenencia de las palabras $baaba, aaaaa$

Representación Gráfica

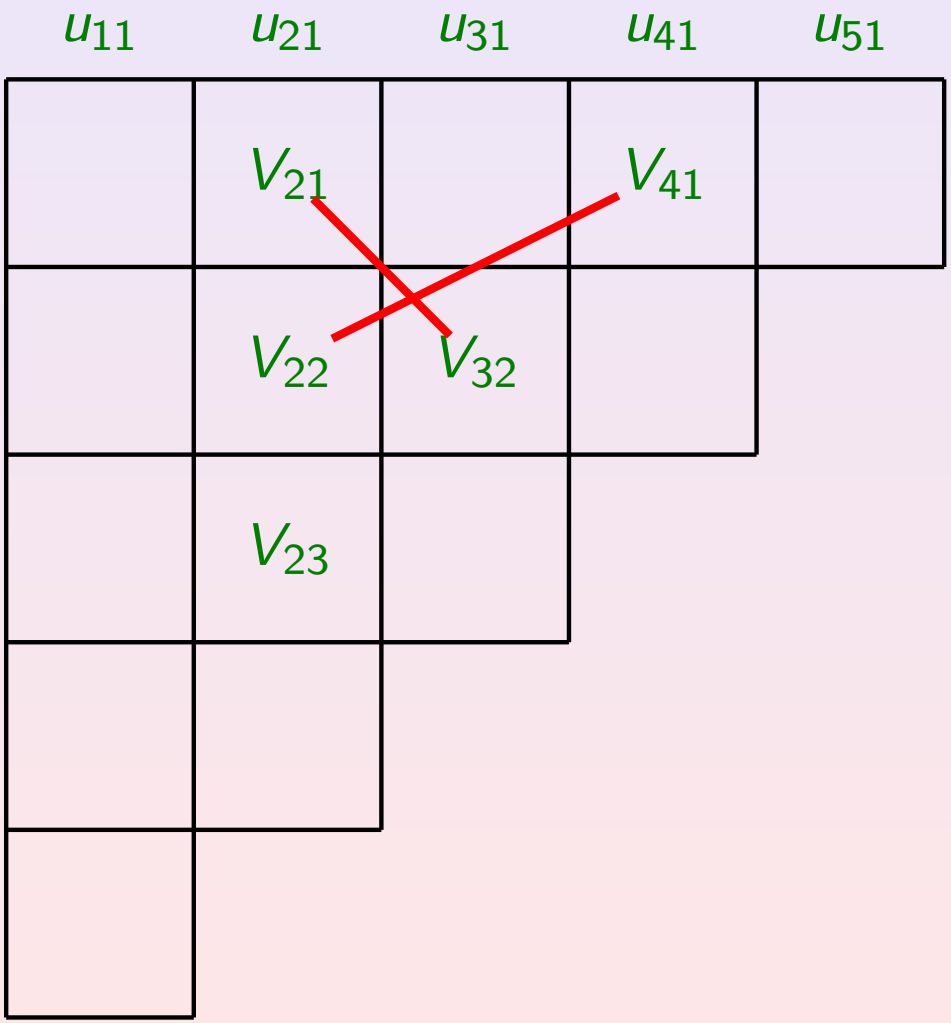


Ejemplo. Cálculo tercera fila

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

Comprobar la pertenencia de las palabras $baaba, aaaaa$

Representación Gráfica

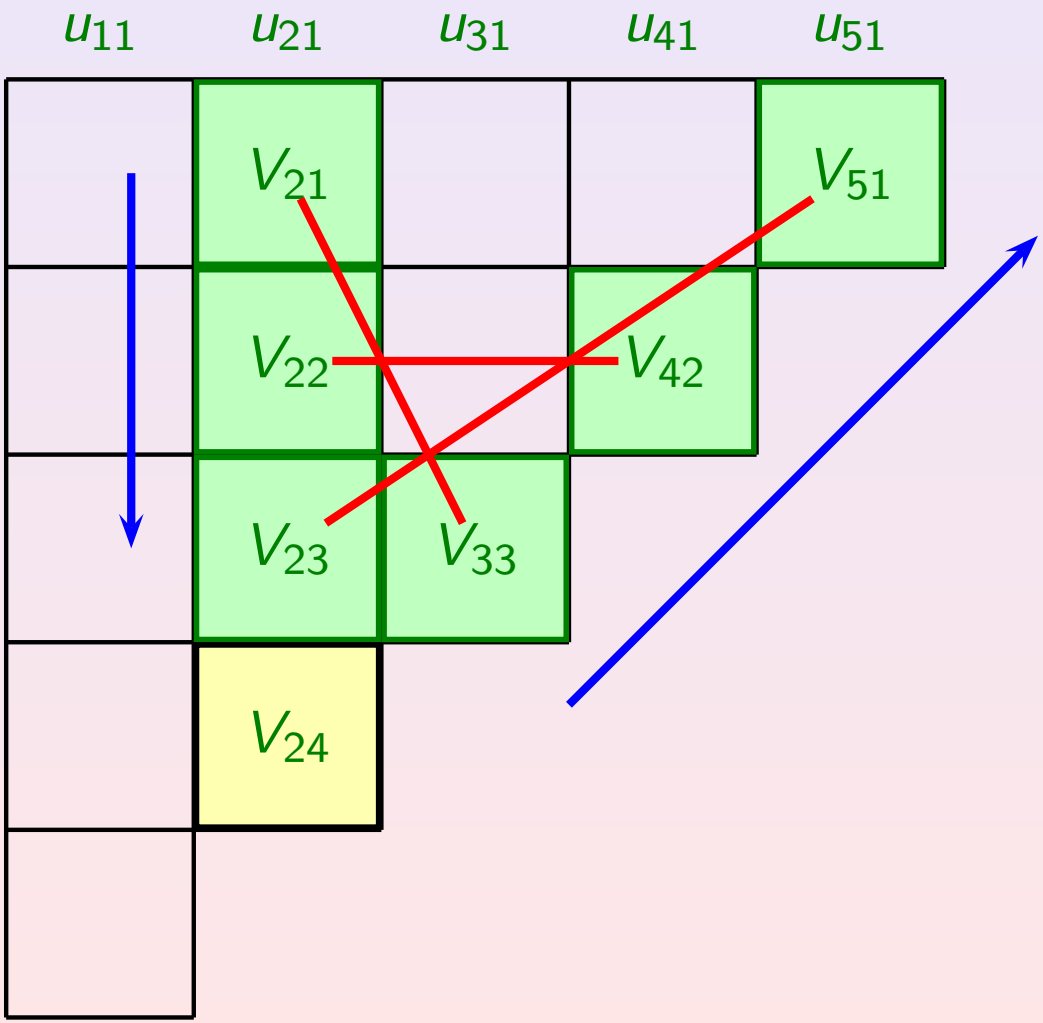


Ejemplo. Fila Cuarta

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

Comprobar la pertenencia de las palabras $baaba, aaaaa$

Representación Gráfica



Ejemplo

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

b	a	a	b	a
B	A C	A C	B	A C
A S	B	S C	A S	
$\emptyset$	B	B		
$\emptyset$	S C A			
S A C				

La palabra *baaba* es generada

Ejemplo

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

a	a	a	a	a
A C	A C	A C	A C	A C
B	B	B	B	
S C A	S C A	S C 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, α, β), 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 α y el objetivo es encontrar todos los k takes que β 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, α, β).

Algoritmo

P1 *Inicialización.-* Sea

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

P2 *Clausura.-* Para cada registro $(i, j, A, \alpha, B\gamma)$ en $REGISTROS[j]$ y cada producción $B \to \delta$, crear el registro $(j, j, B, \varepsilon, \delta)$ 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 registro $(i, j, A, \alpha, \varepsilon)$ (registro completo) en $REGISTROS[j]$, buscar los registros de la forma $(h, i, B, \gamma, A\delta)$ en $REGISTROS[i]$
y para cada uno de ellos crear el nuevo registro $(h, j, B, \gamma A, \delta)$ e insertarlo en $REGISTROS[j]$.

$$\left.\begin{array}{l}(i, j, A, \alpha, \varepsilon) \in REGISTROS[j] \\ (h, i, B, \gamma, A\delta) \in REGISTROS[i]\end{array}\right\} \Rightarrow (h, j, B, \gamma A, \delta) \in REGISTROS[j]$$

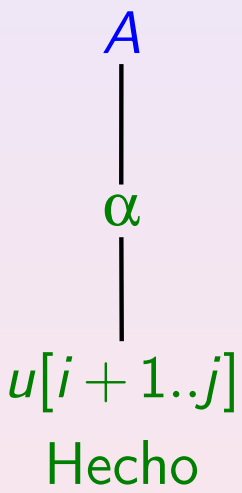
P5 Si $j < n$ ir a P2.

P6 Si en $REGISTROS[n]$ hay un registro de la forma $(0, n, S, \alpha, \varepsilon)$, entonces u es generada. En caso contrario no es generada.

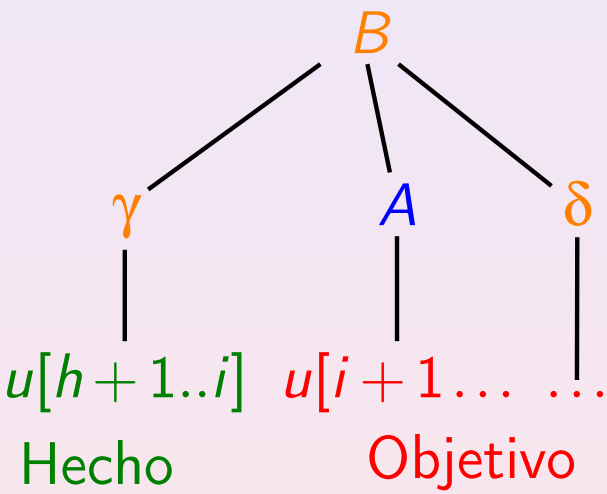
Terminación

$$\left.\begin{array}{l}(i,j,A,\alpha,\varepsilon) \in REGISTROS[j] \\ (h,i,B,\gamma,A\delta) \in REGISTROS[i]\end{array}\right\} \Rightarrow (h,j,B,\gamma A,\delta) \in REGISTROS[j]$$

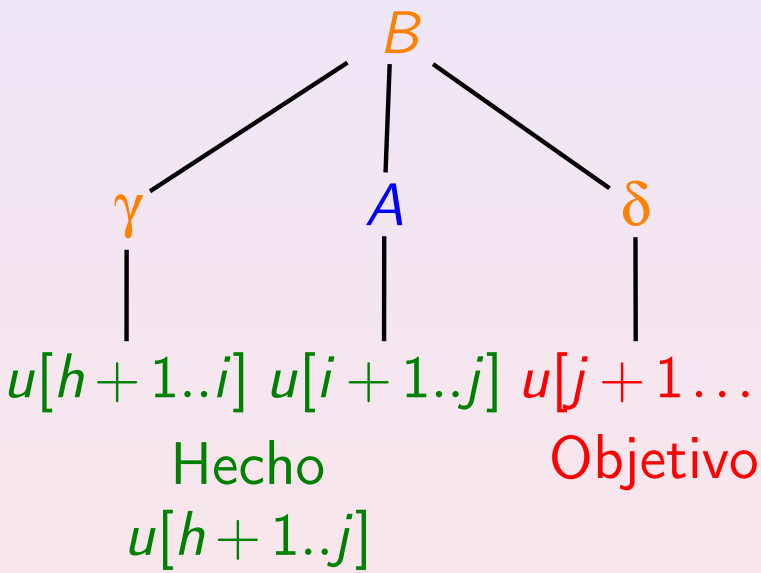
$(i,j,A,\alpha,\varepsilon)$



$(h,i,B,\gamma,A\delta)$



$(h,j,B,\gamma A,\delta)$



Ejemplo

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

Palabra baa.

$REGISTROS[0]: (0,0,S,\varepsilon,AB), (0,0,S,\varepsilon,BC), (0,0,A,\varepsilon,BA),$
$(0,0,A,\varepsilon,a), (0,0,B,\varepsilon,CC), (0,0,B,\varepsilon,b), (0,0,C,\varepsilon,AB), (0,0,C,\varepsilon,a),$
$REGISTROS[1]: (0,1,B,b,\varepsilon), (0,1,S,B,C), (0,1,A,B,A),$
$(1,1,C,\varepsilon,AB),$
$(1,1,C,\varepsilon,a), (1,1,A,\varepsilon,BA), (1,1,A,\varepsilon,a), (1,1,B,\varepsilon,CC), (1,1,B,\varepsilon,b)$
$REGISTROS[2]: (1,2,C,a,\varepsilon), (1,2,A,a,\varepsilon), (0,2,S,BC,\varepsilon), (0,2,A,BA,\varepsilon),$
$(1,2,C,A,B), (1,2,B,C,C), (0,2,S,A,B), (0,2,C,A,B), (2,2,B,\varepsilon,CC),$
$(2,2,B,\varepsilon,b), (2,2,C,\varepsilon,AB), (2,2,C,\varepsilon,a), (2,2,A,\varepsilon,BA), (2,2,A,\varepsilon,a)$
$REGISTROS[3]: (2,3,C,a,\varepsilon), (2,3,A,a,\varepsilon), (1,3,B,CC,\varepsilon), (2,3,B,C,C)$
$(2,3,C,A,B), (1,3,A,B,A)$
Como $(0,3,S,\alpha,\varepsilon)$ no está en $REGISTROS[3]$, la palabra baa no es generada

Ejemplo

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

Palabra: $(a+b)*a$

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

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

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

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

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

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

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

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

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

Para recuperar un árbol de derivación, cada registro contiene una lista de enlaces a otros registros. Esta lista está inicialmente vacía. Cada vez que se produzca una terminación se añade a la lista del registro el registro completo que ha intervenido. En los avances, la lista se hereda.

Problemas Indecidibles

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.