Algoritmo de Earley

n: longitud de la palabra

Me dan unas producciones y una palabra. $i = 0, \dots, n$

1. Iniciación: Hago $reg[0] = (0, 0, S, \varepsilon, \alpha)$ para cada $S \to \alpha$ que haya. α puede ser $A, AB, a \dots$

2. Clausura(i): Añado a $reg[i]$, la clausura de toda variable que esté en la 5ª posición (la que esté más a la izqd). P. ej si tengo $(i, \bullet, \cdot, \cdot, AB)$ añado todas las prod q sean $A \to \alpha$. Las dos primeras posiciones son i. Si añado alguna tupla que ponga una variable en la 5ª pos que no estaba antes, hago su clausura también.

3. Avance(i). Hago i++. Por tanto aquí $i = 1, \dots, n$. Veo el símbolo que está en la pos. i de la palabra que busco, y hago $reg[i]$ = las tuplas con ese símbolo en la 5ª posición. Hago i la segunda posición y paso lo que hay en la 5ª posición a la cuarta (lo añado a la 4ª, 1er var que haya en la 6ª pos). P. ej: símbolo i de mi palabra es b y en $reg[i-1]$ solo hay $(0, 0, B, \varepsilon, b)$ q acabe en b. Entonces $reg_{(i=1)}[i] = (0, 1, B, b, \varepsilon)$

4. Terminación(i). Para cada tupla completa (que acabe en ε), busco en el $reg[j]$, donde j es el 1er valor de la tupla, busco una tupla en la que la variable más a la izqd de la 6ª posición sea la variable que hay en la 3ª pos de la tupla completa q estoy terminando. La añado a $reg[i]$ de la misma manera que he añadido en el avance, pasando la var más a la izqd de la 5ª pos a la cuarta y poniendo i en la segunda posición.

5. Si $i = n$, he terminado, sino vuelvo a la clausura.

Si en reg(n) existe el registro $(o, n, S, \alpha, \varepsilon)$ entonces la palabra pertenece al lenguaje, sino no. α es lo q sea.

FN Chomsky

No puede haber producciones nulas ni unitarias

De la forma $A \to BC$ ó $A \to a$

1. En todas las producciones donde a la derecha haya un símbolo terminal (a) (q no esté solo) lo sustituimos por una variable (X_a) que genere ese símbolo

2. En las producciones donde a la derecha haya más de dos variables, sustituimos la segunda variable por una nueva que genere el resto de la palabra.

FN Greibach

De la forma $A \to a\gamma$ donde γ es una sucesión de variables

Condición: Todas las prod tienen forma $A \to a\gamma$ ó $A \to \gamma$ ↳ $|\gamma| \geq 2$

1. Llamo a todas las variables A_i según el orden que vaya saliendo (S es A_1)

2. Para cada $i = 1, \dots, n$ busco un j menor que i (voy de menor a mayor) tal que $A_i \to A_j\gamma$ donde γ es cualquier cjto de variables.

   2.1 Elimino $A_i \to A_j\gamma$ y añado una nueva regla por cada producción que tenga A_j, sustituyéndolo en $A_i \to A_j\gamma$ por lo que corresponda

   2.2 Si encuentro algún $A_i \to A_i\gamma$, añado una variable B_i con las reglas $B_i \to \gamma$, $B_i \to \gamma B_i$. Para cada regla $A_i \to \gamma$, añado una regla nueva $A_i \to \gamma B_i$. Elimino $A_i \to A_i\gamma$ ↓*

* γ no empieza por A_i

3. Para $i = n, \dots, 1$ busco $j > i$, ahora de mayor a menor, tal que $A_i \to A_j \gamma$

3.1 Elimino $A_i \to A_j \gamma$, y para cada producción que tenga A_j creo una nueva regla, sustituyendo lo que produce A_j en $A_i \to A_j \gamma$ (igual que el 2.1)

4. Para $i = 1, \dots, n$. (-at) Para cada producción $B_j \to A_i \gamma$, elimino $B_j \to A_i \gamma$, y para cada producción que tenga A_i, creo una nueva regla, sustituyendo... (igual q 3.1 y 2.1)

Lema bombeo regulares

$\forall n \in \mathbb{N},\ \exists z \in L,\ |z| \geq n,\ z = uvw \Rightarrow \exists i \in \mathbb{N}: uv^i w \notin L$

$|uv| \leq n$ * Solo tengo que probar q se cumple para una palabra.

$|v| \geq 1$ ~~Para palabras grande usar $0^n 1^n$ por ej~~

Ejemplo

$L = 0^i 1^i$; $z = 0^n 1^n$; $u = 0^k$, $v = 0^l$, $w = 0^{n-k-l} 1^n$

$k + l \leq n$, $l \geq 1$.

para $i=2$ $uv^2w = 0^{k+2l+n-k-l} 1^n = 0^{n+l} 1^n$, $n+l \neq n \Rightarrow z \notin L$

Lema bombeo indep contexto

$z = uvwxy$; $|vx| \geq 1$, $|vwx| \leq n$; $uv^i w x^i y$

Ejemplo

$L = \{a^i b^i c^i \mid i \geq 1\}$, $z = a^n b^n c^n \in L$.

Como $|vwx| \leq n$, no puedo tener las 3 letras ahí, como mucho tendré dos sucesiones de dos letras, que sumadas dan n.

$i=2$, $uv^2 w x^2 y$

Si en $|vwx|$ tengo:

$\{a\} \Rightarrow a^{n+l} b^n c^n$

$\{b\} \Rightarrow a^n b^{n+l} c^n$

$\{c\} \Rightarrow a^n b^n c^{n+l}$

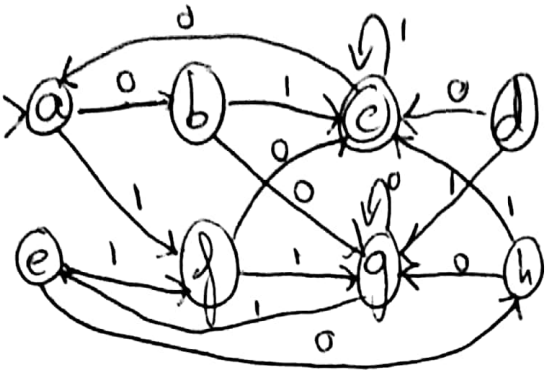
$\{a,b\} \Rightarrow a^{n+l} b^{n+j} c^n$

$\{b,c\} \Rightarrow a^n b^{n+l} c^{n+j}$

Autómata Mínimal

1. Elimino estados inaccesibles
2. Veros estados equivalentes

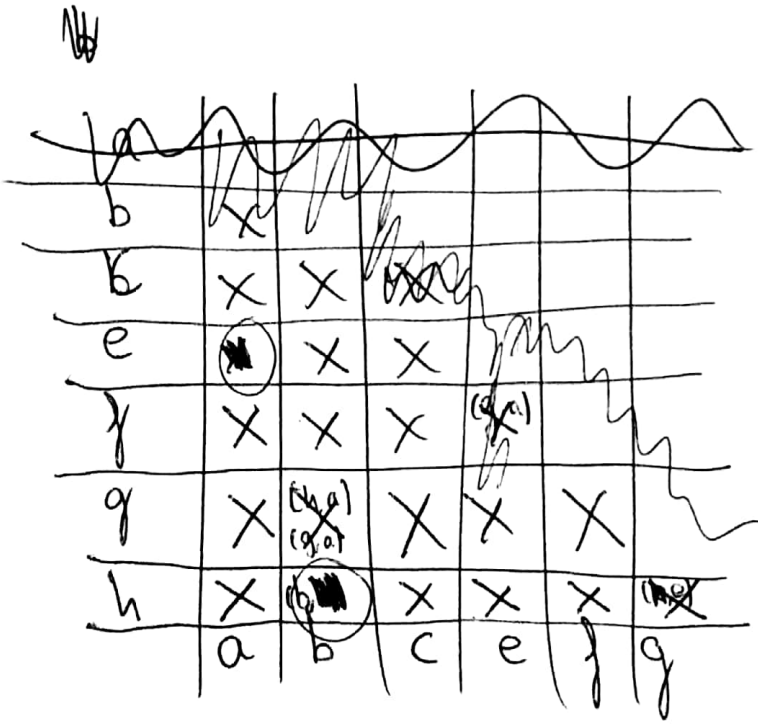
Ej:



1. Elimino d pq es inaccesible

	0	1
h	g	c
a	b	f
b	g	c
e	h	f
f	c	g
g	g	e

2.1 Elimino el final(es) con cualquiera
2.2 Para cada pareja de estados veo donde llega con la tabla
2.3 En los estados a los q llege, apunto de donde vengo.
2.4 Si hay un estado final y otro no, los tacho.
2.5 Si me lleva a 2 estados distinguibles, los tacho



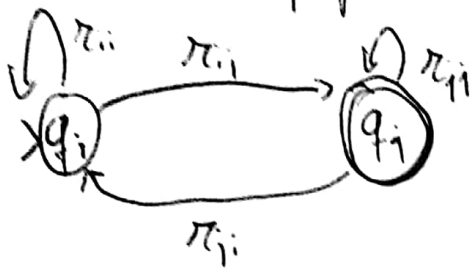
Creo un estado q englobe a la pareja

(e,a), b, h) ⇒ e≡a, b≡h

{e,a} {b,h}

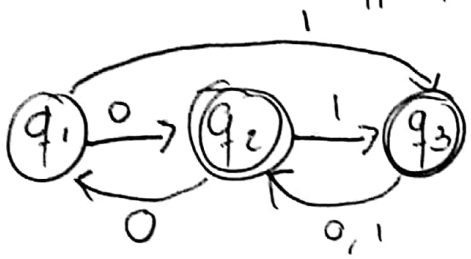
Pasar autómata a ~~gramática~~ expr reg (eliminacion estados intermedios)

La idea es simplificar el autómata hasta que quede algo como:

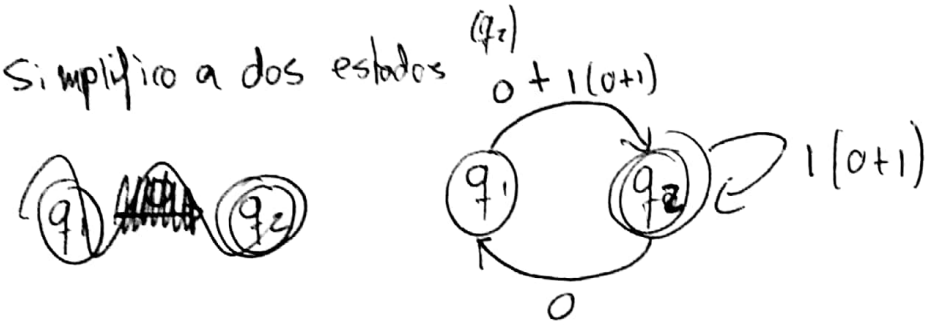


r_{ab} → expr reg q me lleva del estado a al b.

$L(A) = (r_{ii} + r_{ij} r^*_{jj} r_{ji})^* r_{ij} r^*_{jj}$ * Una vez por cada estado final



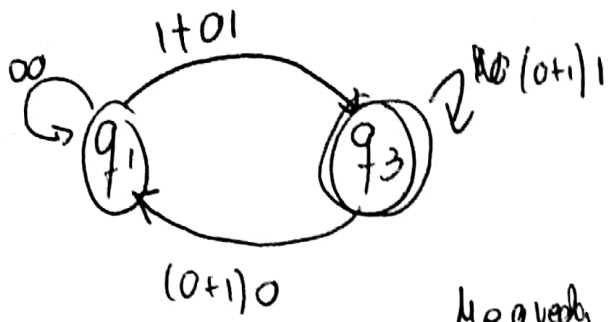
Simplifico a dos estados (q_2)



- De q_2 a q_0 solo puedo llegar con un 0.
- De q_1 a q_2 puedo llegar con un 0 ó pasando por q_3
- De q_1 no puedo hacer bucle (pasando por 1 estado max)
- De q_2 puedo hacer bucle pasando por q_3

$$\left((0+1(0+1))\,(1(0+1))^*\,0\right)^*\,(0+1(0+1))\,(1(0+1))^*$$

Ahora hago lo mismo con q_3



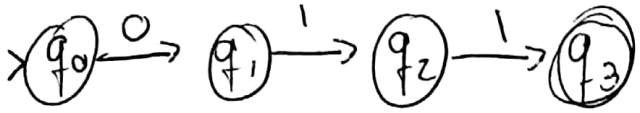
Me queda

$$(00 + (1+01)((0+1)1)^*(0+1)0)^*(1+01)((0+1)1)^*$$

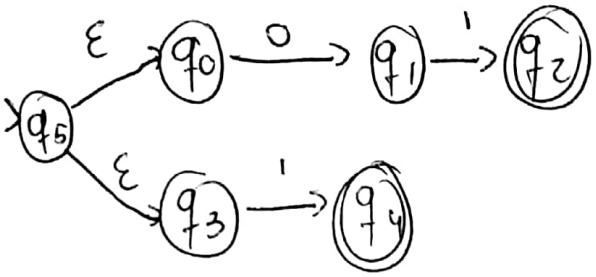
Y tendría que juntarla con la que obtuve antes.

Pasar exp reg → autómata

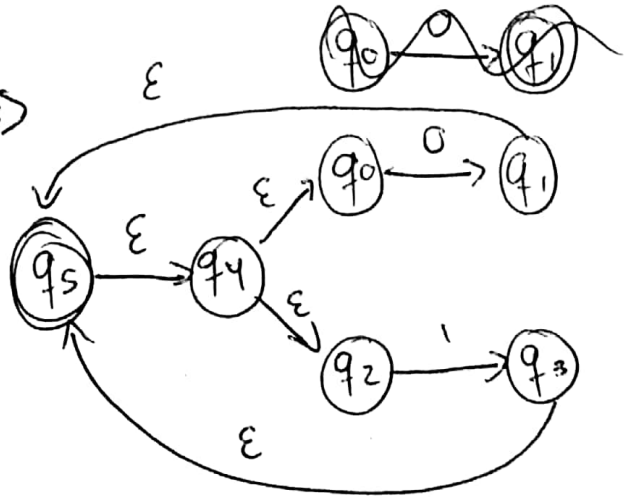
011 ⇒



$(01+1)$ ⇒



$(0+1)^*$ ⇒



Con esto ya podría construir cualquier autómata

Gramática → Autómata pila

Dada una gramática que no produzca la palabra vacía, puedo obtener un autómata que genere su lenguaje. Solo 1 estado

1. Para cada símbolo terminal a, añado $\delta(q, a, a) = \{(q, \varepsilon)\}$
2. Para cada variable $A \to \gamma$ (γ lo que sea), añado $\delta(q, \varepsilon, A) = (q, \gamma)$. Si A tiene más de una prod, entonces llevará a un conjto. $\delta(q, \varepsilon, A) = \{(q, \gamma), (q, \beta)\}$

Autómata pila → Gramática

Sea q_0 estado inicial y R símbolo inicial de la pila.

1. Para cada estado q_i del autómata, añado la regla $S \to [q_0, R, q_i]$
2. Para cada $\delta(\ ,\ ,\ ) = (\ ,\ )$

2.1 Si es de la forma $\delta(q_i, a, X) = (q_1, \varepsilon)$ (puede ser igual a)

Añado $[q_i, X, q_1] \to a$

2.2 Si es de la forma $\delta(q_i, a, X) = (q_1, YX)$

Añado $[q_i, X, q_k] \to a\,[q_1, Y, q_l]\,[q_l, X, q_k]$

En q_k y q_l pruebo combinaciones de todos los estados

2.3 $\delta(q_i, a, X) = (q_1, YYX)$

⋮

Añado $[q_i, X, q_3] \to a\,[q_1, Y, q_1]\,[q_1, Y, q_2]\,[q_2, X, q_3]$

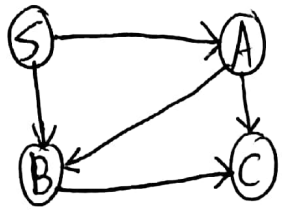
(Igual q antes pero 1 más)

$$n^\circ \text{ combinaciones} = (n^\circ \text{estados})^{n^\circ \text{simb pila}}$$

Ver si un lenguaje es finito

Dada una gramática, creo un grafo con un nodo por cada variable no terminal. Hago aristas dirigidos según las producciones que tenga. Si hay ciclo entonces es infinito

Ejemplo $S \to AB\mid\lambda$, $A \to BC \mid a$, $B \to CC \mid b$, $C \to a$



Como no hay ciclo, es finito

Eliminar producciones unitarias

1. Añado a un conjunto H todas las producciones de la forma $A \to B$ (A, B tienen un solo elemento). H += (A, B)
2. Si en H tengo dos pares tal que (A, B), (B, C) entonces añado (A, C)
3. Elimino de la gramática las producciones que haya añadido a H
4. Añado a la gramática las producciones que están en H, pero sustituyendo la segunda variable por sus producciones. Es decir, si $A \to *\alpha \mid \gamma$, añadiría $S \to \alpha$, $S \to \gamma$ por tener (S, A)

Eliminación producciones / símbolos inútiles

1. Veo símbolos que generan palabras completas y los añado a V_T, y los elimino con todos sus reglas de prod.
2. Veo símbolos que generen los símbolos que tengo en V_T, los añado a V_T y elimino sus reglas de prod.
3. Mientras V_T cambie, repito paso dos

   Los que sobren serán las producciones inútiles
4. Ahora tengo que ver las variables y símbolos a los q llego desde S. Analizo los producciones de S. Las variables a los que llego las añado a V_S y luego analizo sus producciones. Los símbolos terminales a los que llego los añado a T_S.
5. Como es recursivo, cuando acabe tendré en V_S y T_S a todo lo que llego desde S. Si algo no está, lo puedo eliminar

   * Si la variable inicial es inútil, el lenguaje es vacío

Eliminar producciones nulas

1. Para cada variable $A \to \varepsilon$, añado A a H
2. Para cada variable que produzca elementos de H la añado a H.
   $B \to \gamma$, γ cualquier cantidad de variables, pero todas en H.
3. Para cada producción donde a la derecha aparezca al menos una variable de H, añado reglas eliminando las combinaciones de elementos de H que aparezcan en la producción

* Nota: Si $S \in H$, el lenguaje genera la palabra vacía. Con este algoritmo generamos el mismo lenguaje pero sin ε.

Ejemplo

$S \to ABb$, $S \to ABC$, $C \to abC$, $B \to bB$, ~~$B \to \varepsilon$~~

$A \to aA$, ~~$A \to \varepsilon$~~, $C \to AB$

$H = \{A, B, C, S\}$

$S \to Bb$, $S \to Ab$, $S \to b$, $S \to BC \mid AC \mid AB \mid C \mid$ ~~BA~~ $B \mid A$

$C \to ab \mid B \mid A$, $B \to b$, $A \to a$

Autómata Producto/Intersección

Si tengo dos AFD, con estados q_i, p_i, que acepten L_1 y L_2. Puedo construir un autómata que acepte $L_1 \cap L_2$. Agrupo los estados en pares, partiendo de $\{q_0, p_0\}$, y haciendo finales aquellos donde los dos sean finales.

Autómata Unión

Dos AFD, q_i, p_i, para que un autómata acepte $L_1 \cup L_2$ sigo el mismo procedimiento que antes pero hago finales los estados donde al menos un estado del par es final.

Autómata complementario

Para obtener el autómata que genera el lenguaje compl. cambio los estados finales por no finales y viceversa

Para esto tiene que estar completamente definidos los estados y ser AFD