el conjunto cociente y minimizar AFD

- Q/E_0 agrupas los estados en dos, los F y $\bar{F}$ y cada grupo forma una clase de equivalencia y en la tabla de transición pones a qué clase va cada transición. (mirando qué estados tiene cada clase).
- Buscas en la tabla aquellos estados cuyas transiciones vayan a las mismas clase de equivalencia, y los agrupas como una nueva Q/E_n
- repites hasta que dos conjuntos consecutivos sean iguales o hasta Q/E_{n-2} siendo n el nº de estados una vez pasa esto, la última es el Q/E.
- El automata mínimo equivalente está formado por estas clases de eq del Q/E.
- para saber si dos AFD son equivalentes, hacemos el AFD mín de cada uno y comparamos sus clases de equivalencia resultantes.

AFND → AFD

- Calculo T^*, poner todas las transiciones con λ de un estado a sí mismo (q_n, q_n). Hacer la tabla de transiciones y ponemos en la siguiente fila las transiciones con λ y las transiciones intermedias posibles.
- Escribes λ^* en la tabla y tachas λ.
- calculas la tabla usando como símbolo de transición $(\lambda^*$ símbolo $\lambda^*)$ para todo símbolo. ∈ Σ. se empieza por el estado inicial $q_0 \xrightarrow{\lambda^*} O \xrightarrow{sub} O \xrightarrow{\lambda^*}$ R
- Llamamos λ^*símbolo$\lambda^* \equiv$ símbolo, y las transiciones no definidas al ∅ (sumidero) y ponemos también las transiciones del ∅ que van al ∅.
- creas una nueva tabla, en la que empiezas por →O p*lambda* pones sus transiciones, y si alguna va a más de un estado, se combinan, y se hace las transiciones para el estado combinado. y se construye el AFD.

Si algun estado combinado contiene al final tambien sera final

comprobar si una G.2 genera el lenguaje vacío

- Se generan todos los árboles de derivación con caminos ≤ m = C (Σ_{NT}) cardinalidad

1. m=0 conjunto de árboles de long 0 (árbol con S como raíz y sin ramas.)
2. a partir del conjunto de árboles de long n, generamos el conjunto de long n+1 < m+1 aplicando al conjunto de partida una producción que no haga duplicarse algún NT en el camino considerado.
3. aplicamos el paso 2 hasta que no se duplique o se llegue al límite.

=> L(G2) = ∅ si ninguno de los árboles genera una sentencia.

FORMA NORMAL DE CHOMSKY. (FNC)

LIMPIAR Y BIEN FOMAR PRIMERO

cuando las reglas tienen la forma { S → λ ; A → b ; A → BC } A parte de S.
donde S, A, B, C ∈ Σ_{NT}, b ∈ Σ_T

a partir de G2, construir una equivalente en FNC con algoritmo corte y confección.

Ej: S → ab (A, B) => S → AB ✓, A → a ✓, B → b ✓

S → asb (A, C) => S → AC ✓, C → sb (lo hemos puesto antes)

C → sb (B) => C → SB ✓ (los no hace falta para es un NT)

configuración {q, w} (estado en el que estás; lo que queda por leer)

$f'(q, ax) = f'(f(q, a), x)$ AFD

y luego hay que limpiar las gramáticas después de FN

Equivalencia con AUTÓMATA SUMA.

- juntas las dos tablas de transición y lo tratas como si fuese un único autómata
- Hallas el conjunto cociente Q/E y si los dos estados iniciales se encuentran en la misma clase de equivalencia, ambos autómatas son equivalentes.

(hasta $Q/E_{n_1+n_2-2}$ siendo n_1 y n_2 el nº de estados de los autómatas que se comparan.

G3LD ↔ G3LI

S → T... (S) —T→ (λ)

- quitar el axioma inducido (cuando aparece S a la derecha de una regla de producción.
  1. añade un nuevo símbolo en Σ_{NT} ej: B
  2. ∀ S → x donde $x \in \Sigma^*$, se añade la regla B → x. (el nuevo símbolo tiene que tener las mismas reglas que el axioma.

  Cuando axioma da lambda, al copiarla en el nuevo actuar como regla no generativa, sustituir sus apariciones por lambda

  3. se transforma la regla del axioma inducido (eliminamos), cambiando S por el nuevo símbolo. (las reglas S → λ no se ven afectadas)
- construimos grafo dirigido a partir de la gramática
  - nº de nodos = nº de no terminales + 1, cada nodo etiquetado con su nombre y uno de ellos con λ.
  - cada A → aB ⇒ (A) —a→ (B)
  - cada A → a ⇒ (A) —a→ (λ) ; S → λ ⇒ (S) —λ→ (λ) ?
- transformamos el grafo dirigido anterior
  - intercambias las etiquetas S y λ
  - inviertes el sentido de los arcos
  - generas las nuevas reglas. ⇒ interpreta el grafo según la gramática que buscas o LI o LD

El lenguaje que reconocen es equivalente y si no quitas axioma inducido saldrían flechas de (λ).

si L(G2) ≠ ∅, comprobar si L(G2) = ∞ ?

Construimos un grafo cuyos nodos están etiquetados con los símbolos de (Σ_{NT}):

1. si ∃ una producción A → α B β, se crea un arco entre A y B.
2. si no existen ciclos en el grafo L(G2) = finito.
3. L(G2) = ∞, si existen ciclos accesibles desde el axioma. $A \rightarrow^+ \alpha A \beta$ con α y β sin ser λ a la vez.

Bien formar Gramáticas!

1. R innecesarias (A → A ∈ P las eliminamos)
2. simb. inaccesible (construimos el mismo grafo que para saber si L(G2) es ∞) y ver que todo sean accesibles desde el axioma, si no lo son, se eliminan. Construir vector con símbolos T y NT
3. R. superfluas. con el algoritmo de marcado:
   - primera pasada, marcas las reglas que den un terminal o λ.
   - combinaciones de los NT ya marcados y terminales.
   - si queda alguna regla sin marcar es superflua y el símbolo NT no marcado → 4. símbolo no generativo
5. R. no generativas. (A → λ con A ≠ S)
   - eliminas la regla A → λ
   - por cada regla donde A aparezca a la derecha, se añade la regla que quedaría si A valiese λ.
   - repetir hasta que no quede ninguna (S → λ si se permite)
6. R. de redenominación (A → B, S → B ...)
   - elimina la regla
   - por cada regla, añadimos las reglas de producción del símbolo eliminado (si no están ya) (quitas B y pones sus reglas).
   - repetir hasta que no quede ninguna.

Si cuando eliminamos solo quedaba un simbolo (C->M y eliminamos M), ponemos lambda (C->λ) y repetimos el proceso de eliminacion (para C).

FORMA NORMAL DE GREIBACH (FNG)

Las reglas tienen la pinta $\{ N_T \to T\alpha$ donde $\alpha \in \Sigma^*_{NT}$; $N_T \to T$

$A \to bCD \mid a$ $\quad S \to \lambda$ (si permito cadena de longitud 0)

Toda GL puede producir una equivalente sin reglas recursivas a izquierda.

$N \to NC \mid C$ — α (acompaña a la recursiva)

$C \to \wedge \mid /a$ — β (no acompaña, y es el valor por el que se sustituye al final.)

Transformamos R1 a RD.

$N \to \beta \mid \beta X$ ← nuevo símbolo NT que generará la RD.

$X \to \alpha X \mid \alpha$

en el ejemplo: $N \to C \mid CX$; $X \to CX \mid C$; $C \to \wedge \mid /a$

¡Si tenemos + a la der de los NT, los sustituimos por NT uno a uno y añadimos sus reglas!

Algoritmo FNG.

1. Limpiar y bienformar la gramática y eliminar RI

$A \to A\underline{a} \mid A\underline{b} \mid c \mid d \mid BA$ (α_1, α_2, β_1, β_2, β_3) — como solo hay RI en A solo es necesario A simb.

$A \to \beta_1 \mid \beta_2 \mid \beta_3 \mid \beta_1 X \mid \beta_2 X \mid \beta_3 X$

$X \to \alpha_1 \mid \alpha_2 \mid \alpha_1 X \mid \alpha_2 X$

2. Transformación

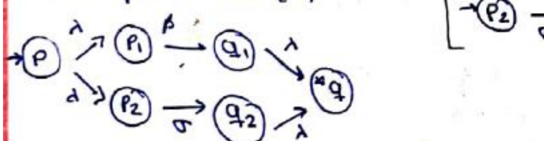
2.1 ordenar el alfabeto de NT dependiendo de qué NT va antes que el resto.

2.2 clasificamos en 3 grupos: G1: son las que ya están en FNG, G2: reglas bien colocadas respecto al orden, G3: reglas mal colocadas respecto al orden.

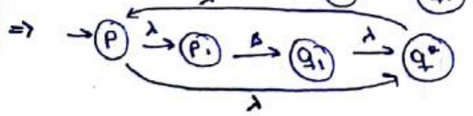
2.3 intentamos transformar de tipo G3 → G2 y los de G2 → G1, para ello sustituimos el NT más a la izq por su valor (empezamos por los de G3) ¡puede volver a salir RI y hay que quitarla, y tenemos que volver a limpiar la gramática.

Algoritmo recursivo de síntesis.

- si $\alpha = \emptyset$ AF: → (p) (*q) · si $\alpha = \lambda$ AF: → (p) —λ→ (q*)
- si $\alpha = a$ AF: → (p) —a→ (*a)
- si $\alpha = \beta + \sigma$ con (*) [→(p₁) —β→ (q₁*); →(p₂) —σ→ (q₂*)] (*)



- si $\alpha = \beta \cdot \sigma$ => →(p₁) —β→ (q₁) —λ→ (p₂) —σ→ (*q₂)
- si $\alpha = \beta^*$ con →(p₁) —β→ (q₁*)



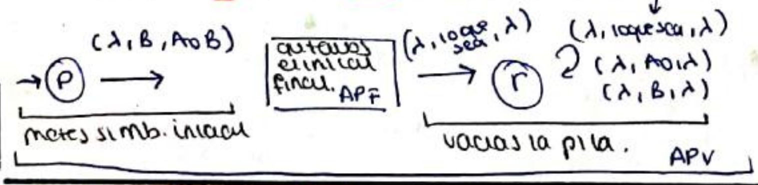
si los minimizas quedan los que pensarías.

PASO DE APF → APV

$AP_F = (\Sigma, \Gamma, Q, A_0, q_0, f, F)$

$AP_V = (\Sigma, \Gamma \cup \{B\}, Q \cup \{p, r\}, B, p, f', \emptyset)$

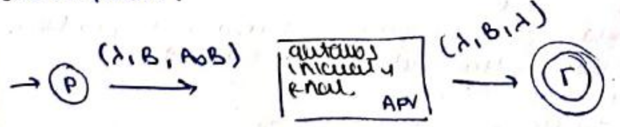
Todo lo que pueda haber en pila.



metes simb. inicial — quitamos el inicial final APF — vacías la pila. APV

PASO de APV → APF

necesitamos un símbolo inicial de pila y dos estados más (uno es final).



como es por vaciado sabremos que no habrá nada en la pila, solo el símbolo que hemos añadido.

GSLD → AF

$GSLD = (\{a,b\}, \{A,B\}, A, P = \{A \to aB;\ B \to bB \mid b\lambda\})$ — Σ_T, Σ_{NT}, S — P: limpia y bien formada

¿AF? $= (\Sigma = \Sigma_T, Q = \Sigma_{NT} + F, q_0 = A, f, F)$ — ↑ Añadimos un estado final. — ↳ se construye a partir de las reglas de producción

$f = \{A \to aB \sim f(A,a) = B$

$B \to bB \sim f(B,b) = B$

$B \to b \sim f(B,b) = F\ \lambda$

*si el estado inicial fuese inicial y final, hay que añadir axioma para dar λ $A \to \lambda$ $A = S$ y en los estados finales $\{F, A\}$.

AF → GSLD

Mejor AFD

$AF = (\Sigma, Q, q_0, f, F)$

La $GSLD = (\Sigma_T, \Sigma_{NT}, S, P)$

Si $f(P,a) = Q \to P \to aQ$; si Q es final $P \to a$ también

$P = \{f(A,a) = B \rightsquigarrow A \to aB$

$f(B,b) = B \sim B \to bB$

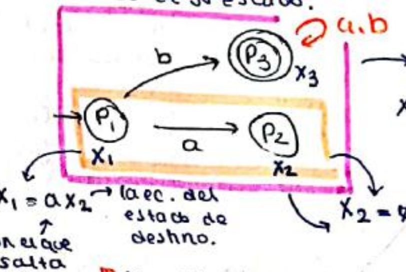
$f(B,b) = F \sim B \to b\ \lambda$ ((a) reglas del sumidero quitan (R superfluas)).

si estado inicial es final añadir $q_0 \to \lambda$

AF → ER (problema de análisis)

Analisis

- cada estado, le asignamos una ecuación característica X_n que describen cadenas que se pueden reconocer desde su estado.



$X_3 = \lambda \to$ no sale ninguna pero como es final λ

$X_1 = aX_2 + bX_3 + b$ (como X_3 es final) Añade

$X_1 = aX_2$ → la ec. del estado de destino. con el que salta

$X_2 = \emptyset$ no sale ninguna trans. y no es estado final.

$X_3 = aX_3 + bX_3 + a + b$

- Resolvemos por sustitución o por la regla de (Arden.) inferencia $\underline{X} = A\underline{X} + B \to X = A^*B$

$b^*b + \lambda = b^*$

X0 = aX0 X0 = Ø
X1 = bX1 + c X1 = b*c
X2 = c X2 = c

pueden haber varios sumandos $AX + B + b = X \to X = A^*(B+b)$

- siempre la ecuación del sumidero es Ø. $a\emptyset = \emptyset$.

Algoritmo derivadas sucesivas de ER.

Sintesis

$Da(R)$ si $R = ax$, es quitarle el principio de la cadena el símbolo por el cual derivas.

$Da(R+S) = Da(R) + Da(S)$ → plantearse si el primer trozo puede ser λ

$Da(RS) = Da(R)S + \delta(R) \cdot Da(S)$ → si $\lambda \in R \Rightarrow \delta(R) = \lambda$; si $\lambda \notin R \Rightarrow \delta(R) = \emptyset$

$Da(R^*) = Da(R) \cdot R^*$

$Da(a^*) = a^*$ // $Da(\emptyset) = \emptyset$ // $Da(\lambda) = \emptyset$ / $Da(a) = \lambda$ //

$Da(b) = \emptyset$.

Derivar la expresion respecto de todos los simbolos y todas las que me vayan saliendo. Las voy nombrando y puede haber del tipo R2=R0+R1.

- Derivamos sobre los elementos de Σ, si $\Sigma = \{a, b\}$ se hace Da y Db.
- cogemos como primer trozo lo más directamente derivable.

$Da(R_0) = R_1$; $Db(R_0) = \emptyset$ ....

apuntamos también δ de cada ER.

⇒ transformamos $Da(R_0) = R_1 \Rightarrow (R_0 \to aR_1)$

si $\delta(R_0) = \lambda$ añades $S \to \lambda$ es decir $(R_0 \to \lambda)$

si $\delta(R_1) = \lambda$ $(R_0 \to aR_1 / a)$

Aplicamos algoritmo de GSLD → AF

PASAR DE ER A AF

cuando puede ser λ: { $a + b^*$, a^*b, $a + \lambda$, $(a+b)^*$ } — no { a, $a + b$, $a \cdot a^*$ }

$Dab(a^*ab) = Db(Da(a^*ab))$

DE G2 en FNG → APV (1 solo estado q)

$G = (\Sigma_T, \Sigma_{NT}, S, P)$

$APV = (\Sigma_T, \Sigma_{NT}, \{q\}, S, q, f, \emptyset)$

Dado G2FNG.

S → λ
A → aBCD
B → b

=D

Σ = ΣT   A0 = S
Γ = ΣNT   f
q0 = q   ∅.

símbolo inicial pila

- S → λ ⊔ : f(q, λ, S) → (q, λ) (desapila símbolo inicial)
- B → b ⊔ : f(q, b, B) → (q, λ) desapila
- A → aBCD : f(q, a, A) → (q, BCD) apila

2213012, 2113012, 1013012, 3012, 012, 12, 2 mod 0, mod 1, mod 2, mod 0, mod 1, mod 2, NSR

DE APV a gramática tipo 2.

λ, A, B ← λ, p, q, r
(pAp) (qAq) (pAq) (qAp)

estado*simbPila*estado

$\Sigma_T = \Sigma$; $\Sigma_N = S$, estados ; S = S ; P se saca de f

combinamos los estados y símb pila

1. S → (q0, A0, q) Aquellas que empiezan con el estado inicial y leen el símbolo de pila (redenominación.)

2.
- f(q, a, A) = (q, BCD)
  (q, A, ⊔) → a (q, B, ∪)(∪ C ^)(^ D ⊔)
- f(q, a, A) = (q, λ) → una de desplazamiento.
  (q, A, q) → a
- f(q, λ, Z) = (q, λ) → quita el símbolo inicial de pila (no generativa)
  (q, Z, q) → λ
- f(q, λ, A) = (q, B) → λ es raro si no estuviera
  (q, A, ⊔) = λ (q, B, ⊔)

cada hueco tiene que corresponderse.

(q A p) = a (q B p) (p C p) (p D p)
a (q B p) (p C q) (q D p)
a (q B q) (q C p) (p D p)
a (q B q) (q C q) (q D q)

B → λ compres. no se tiene en cuenta para determinar la gramática.

G1LD + G3LI = G2.

si B → λ y G∅2 → G∅2

TIPOS DE GRAMÁTICAS.

+ G3LD + G3LI = G2

Jerarquia Chomsky

- G∅ sin restricciones (lo que sea → lo que sea), no guardan contexto, reglas compresoras (aA → a)

Si B no es axioma, B->lambda es compresora. Pero eliminable por no generativa(Sustituyendo)

- G∅ con estructura de frases → igual G∅, sin reglas que cambien el contexto. (pueden ser representados en varios pasos) también hay compresoras.
- G1, sensible al contexto → como tipo ∅ con estruc. de frases, sin reglas compresoras.

CONTEXTO

NT → ◯

- G2, independiente de contexto ⇒ NT → lo que sea
  no pueden haber A → λ si A ≠ S.

Tambien la mezcla de G3LD y G3LI

- G3 Regulares lineal izq ⇒ NT → NtT ó NT → T
- G3 lineal derecha ⇒ NT → TNt ó NT → T

Máquina de Turing.

= (Σ, Γ, b, Q, q0, f, F)

cinta – estados – función
entrada – blanco – est. inicial – est. finales.

se mueve por la cinta, lee y según el contenido, escribe y se mueve.

EQUIVALENCIAS.

(α + β) + σ = α + (β + σ)
(α · β) · σ = α · (β · σ)
α + β = β + α
α (β + σ) = (α β) + (α · σ)
(β + σ) α = (β α) + (σ α)

aλ = λa = a
α + ∅ = ∅ + α = α
λ* = λ
α · ∅ = ∅ · α = ∅
la potencia 0 de cualquier cosa = λ
∅* = λ

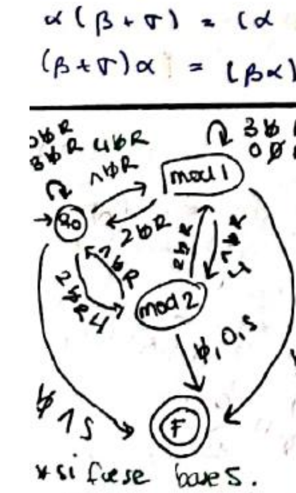
sumar base 3

resto 3 que contenga los 0 de 3 ceros

1 1 1
2 1 2 2
2 1 0 2
1 2 0 0 1

α* · α* = α*
α · α* = α* · α
(α*)* = α*
α* = λ + α · α*
(α* + β*)* = (α* · β*)* = (α + β)*
(α* β)* · α* = (α + β)*

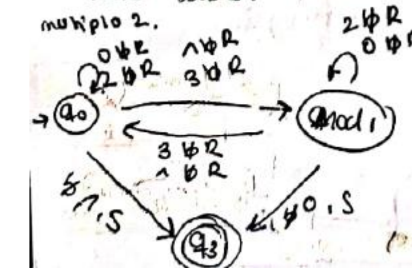
13. α* = λ + α + α² +… + αⁿ + αⁿ⁺¹ · α*
19. (α · β)* · α = α · (β · α)*
20. (α* · β)* · α* = (α+β)*
21. (α* · β)* = λ + (α+β)* · β
22. X = Ax + B → X = A* · B



2213
2 + 2 = 4 − 3 = 1
1 + 1 = 2
2 + 3 = 5 − 3 = 2
no es múltiplo.
(decir si un nº en base 4 es múltiplo de 3).

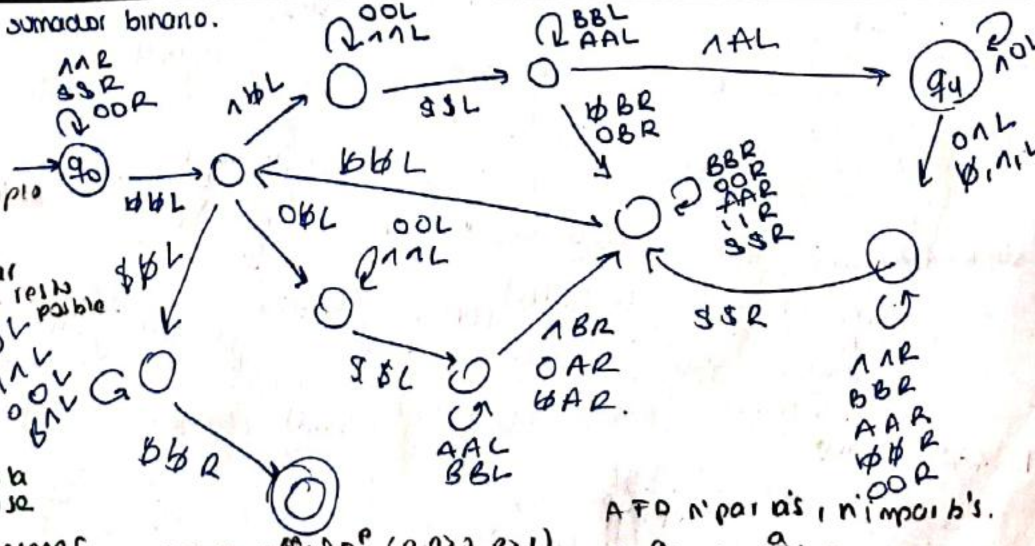
añadir o quitar estados por cada resto posible.

*si fuese base 5. múltiplo 2.

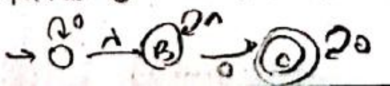


si es menor la base solo se quitan transiciones.

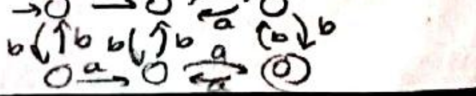
sumador binario.



AFND $a^{n} b^{n} c^{p}$ (n, n≥0, p≥1)



AFD nº par a's, nº impar b's.



Crear gramática para un lenguaje.

- Reglas que hacen crecer a partir de un eje $S \to ab \mid aSb$
- Reglas que generan mismo símbolo $A \to Aa \mid a$
- Reglas en los que hay una 'a' de más por cada bloque B

$B \to aBb \mid aab$ genera a

- Reglas para n mayor de 'a' que de 'b' $B \to aBb \mid aab \mid aB$
- para hacer crecer un símbolo en medio de otro

$S \to aa \mid aBa$

$B \to bB \mid b$

- Reglas de capa de cebolla.

Ej: $x^n y^n z^{m+n}$ $n,m \geq 0$ $= x^n y^m z^n z^m = x^n y^m z^m z^n$

primero creamos el corazón

$M \to yMz \mid yz$

para capa de fuera $N \to xNz \mid xz \mid xMz$ (vínculo al corazón)

$\Rightarrow$ {
$S \to \lambda \mid xz \mid yz \mid xMz \mid xNz$
$M \to yMz \mid yz$ capa interna
$N \to xNz \mid xz \mid xMz$ } capa externa

Gramática genera nº naturales:

$N \to CR \mid D$

$R \to DR \mid D$

$D \to \emptyset \mid C$

$C \to 1 \dots 9$ }

*para crear lenguaje a partir de gramática vamos haciendo árbol.

AFD = (Σ, Q, f, q_0, F)
alf. entrada, L función transición.

$G = (\Sigma_T, \Sigma_{NT}, S, P)$ reglas
L producciones.

MT=(Alfabetoentrada, AlfabetoCinta, b, Q, q0, f, F) b=SimbolosEspeciales

APV = $(\Sigma, \Gamma, Q, A_0, q_0, f, \emptyset)$

APF = $(\Sigma, \Gamma, Q, A_0, q_0, f, F)$

estado transición

$f(p, a, A) = (q, AA)$ lo que apila

L lees L símbolo lees L estado al que vas

$L = \{a^n b^n c^n\}$, sabemos $a^n b^n$, $b^n c^n$, $a^n c^n$ (principio, meter)

para crecer as y cs $\Rightarrow S \to aC / aSC$, para meter b's por el medio $\Rightarrow S \to abc / abSc$

$S \to abSc \to ababScc \to abababccc$ (bien)

planteamos. ✓ — necesitamos regla que $ba \to ab$.

$S \to abc / aBSc$

$Ba \to aB$

$B \to b$

$S \to aBSC \to aBabcc$ — ($Ba \to aB$) $\to aaBbcc$ — ($B \to b$) $\to aabbcc$

GØ sin restricciones. $B \to b$ $\to ababcc$ mal colocado.

* obtener la gramática del siguiente tipo más restringido si es $T0 \to T1$ (autos compresores)

$T1 \to T2$

$T2 \to T3$.

$a^n b^{2n} b^m c^m$

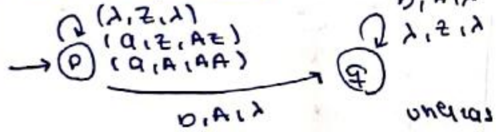
L $M \to bc \mid bMc$

$N \to abb \mid aNbb$

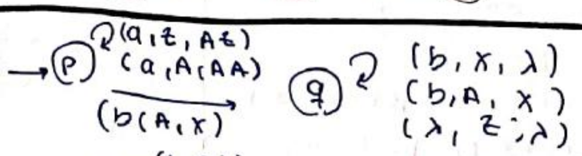
$S \to NM \Rightarrow$ juntas

APV a partir lenguaje.

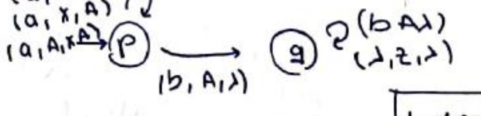
$L = \{a^n b^n / n \geq 0\}$.



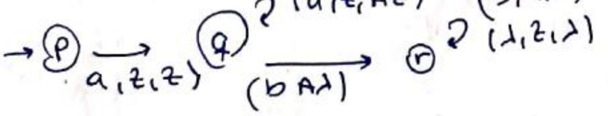
si $L = \{a^n b^{2n} \mid n > 0\} \to$ vueltas intermedias
b impar (1º) → b_i → A → X
b par (2º) → b_2 → X → λ



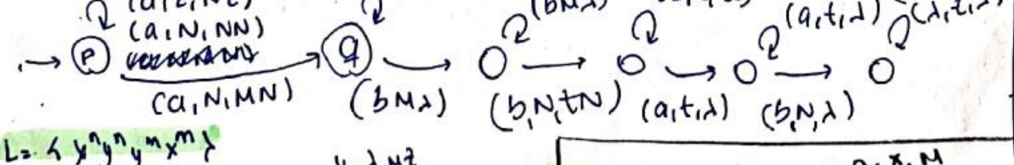
si $L = \{a^{2n} b^n \mid n > 0\}$ —
a – X
a – X – A
B – A – λ.



si $L = \{a^{n+1} b^n / n > 0\}$

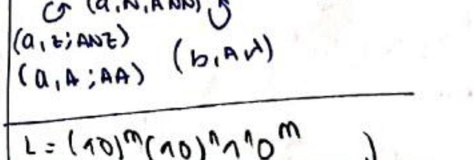


si $L = \{a^{n+m} b^{m+t} a^t b^n / n, t > 0, m \geq 0\} \to a^n a^m b^m b^t a^t b^n$ (m, t, N)

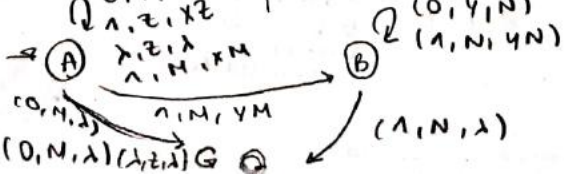


$L = \{(a^{m_i} b^{m_i})^n b^n / n > 0, m_i > 0$, $i = 1, 2 \dots\}$

(aabb)(aaabbb)bb



$L = (10)^m (10)^n 1^n 0^m$



$L = \{y^n y^n y^m x^m\}$