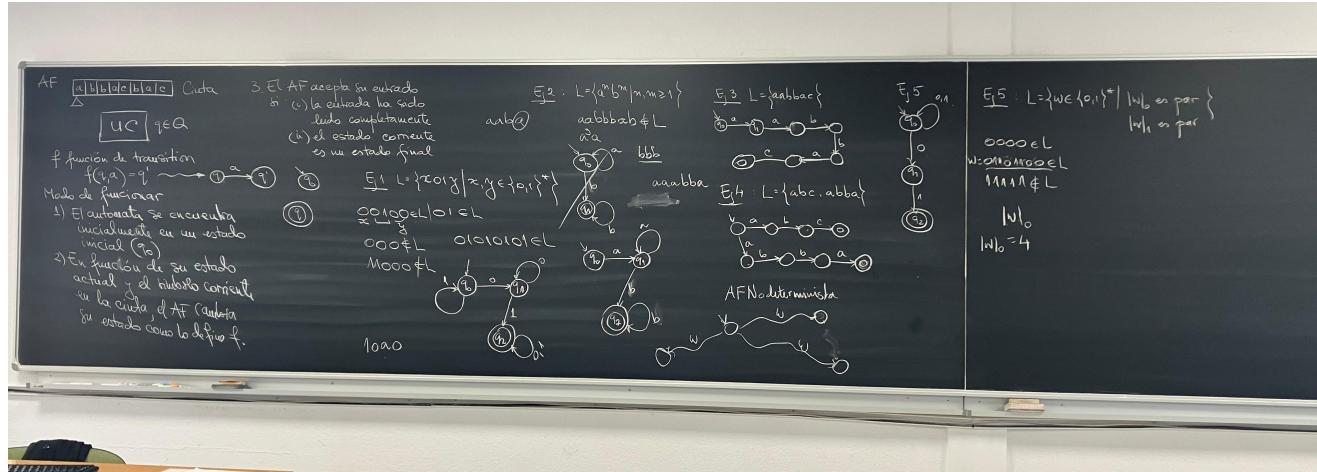


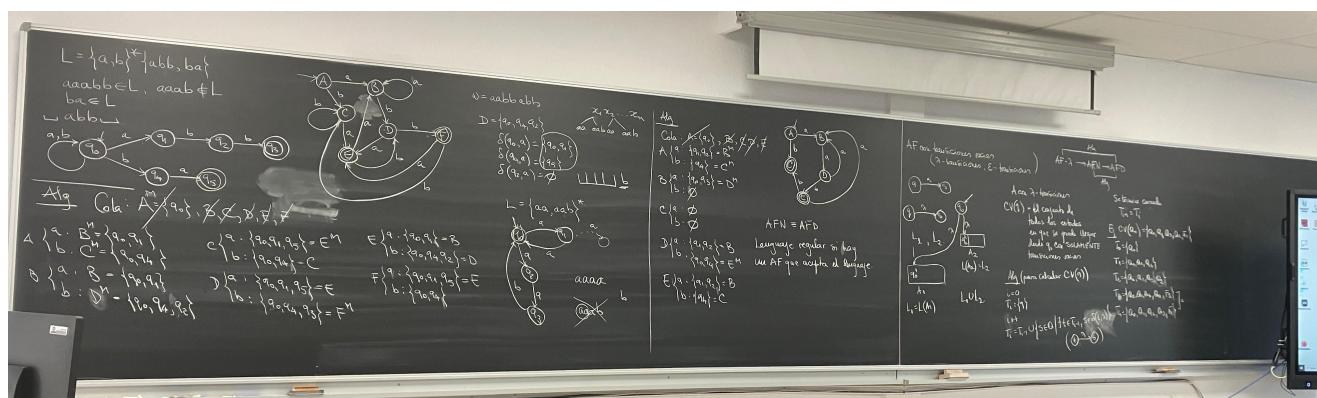
Class Notes

Class Notes

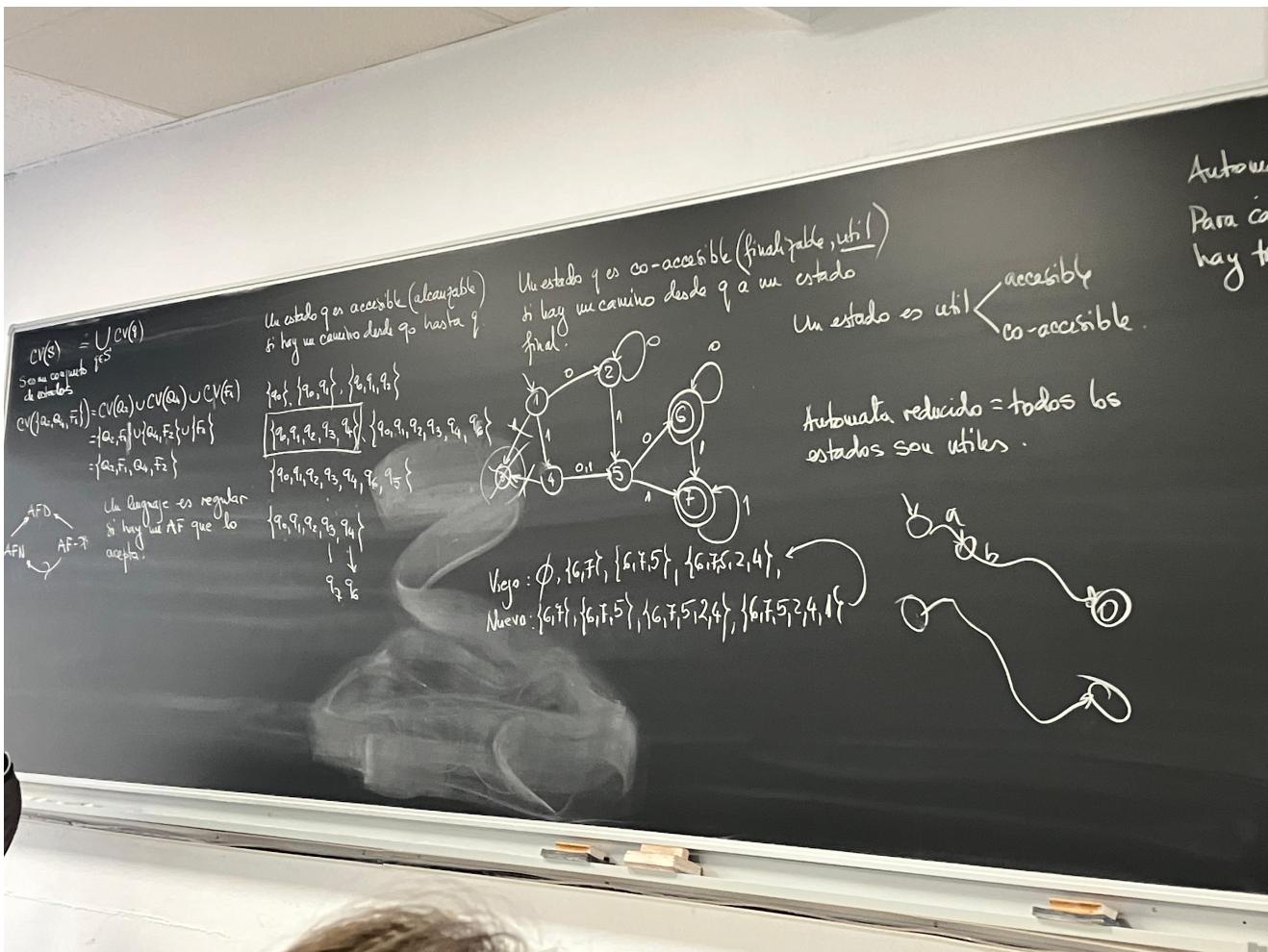
1. Basic Concepts



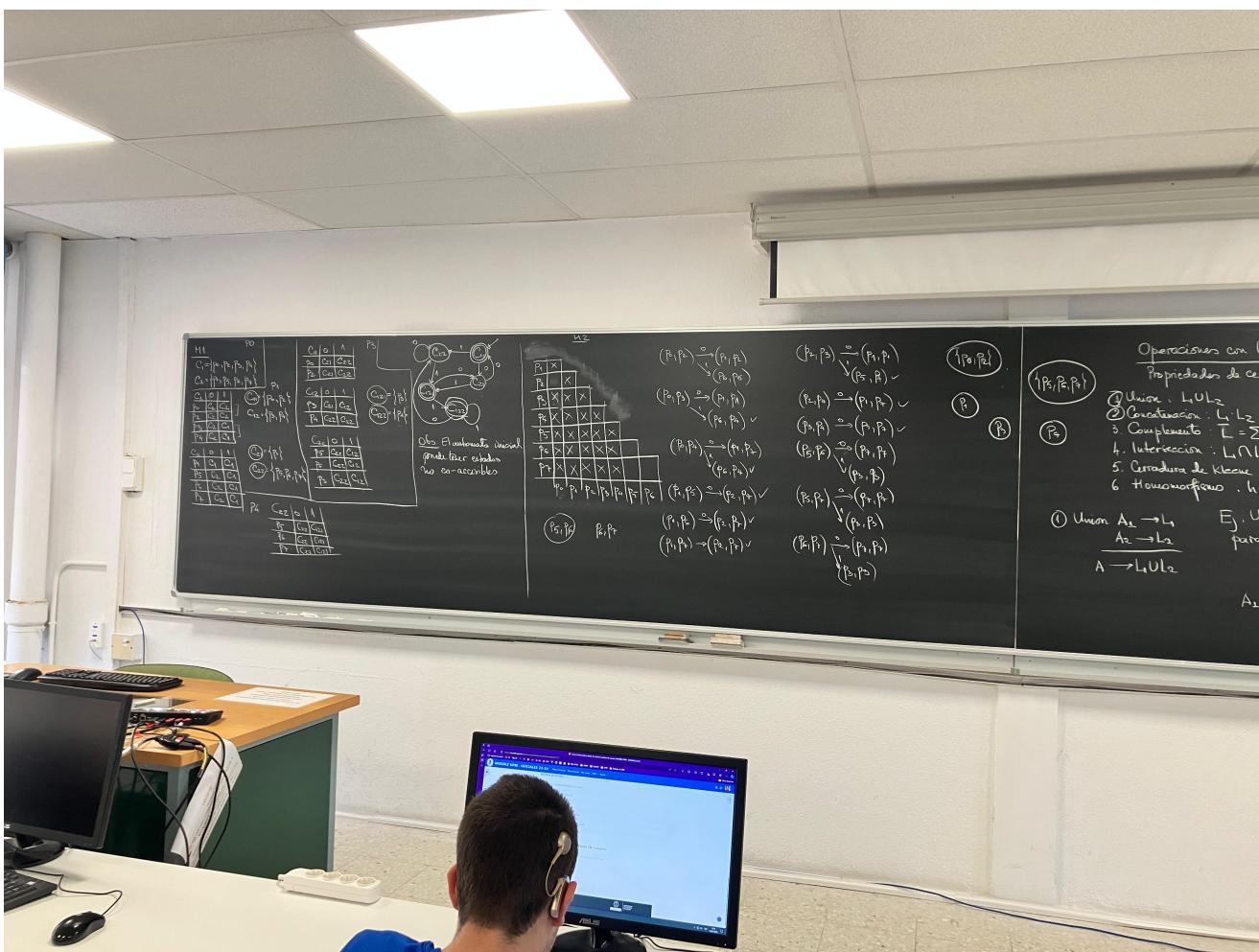
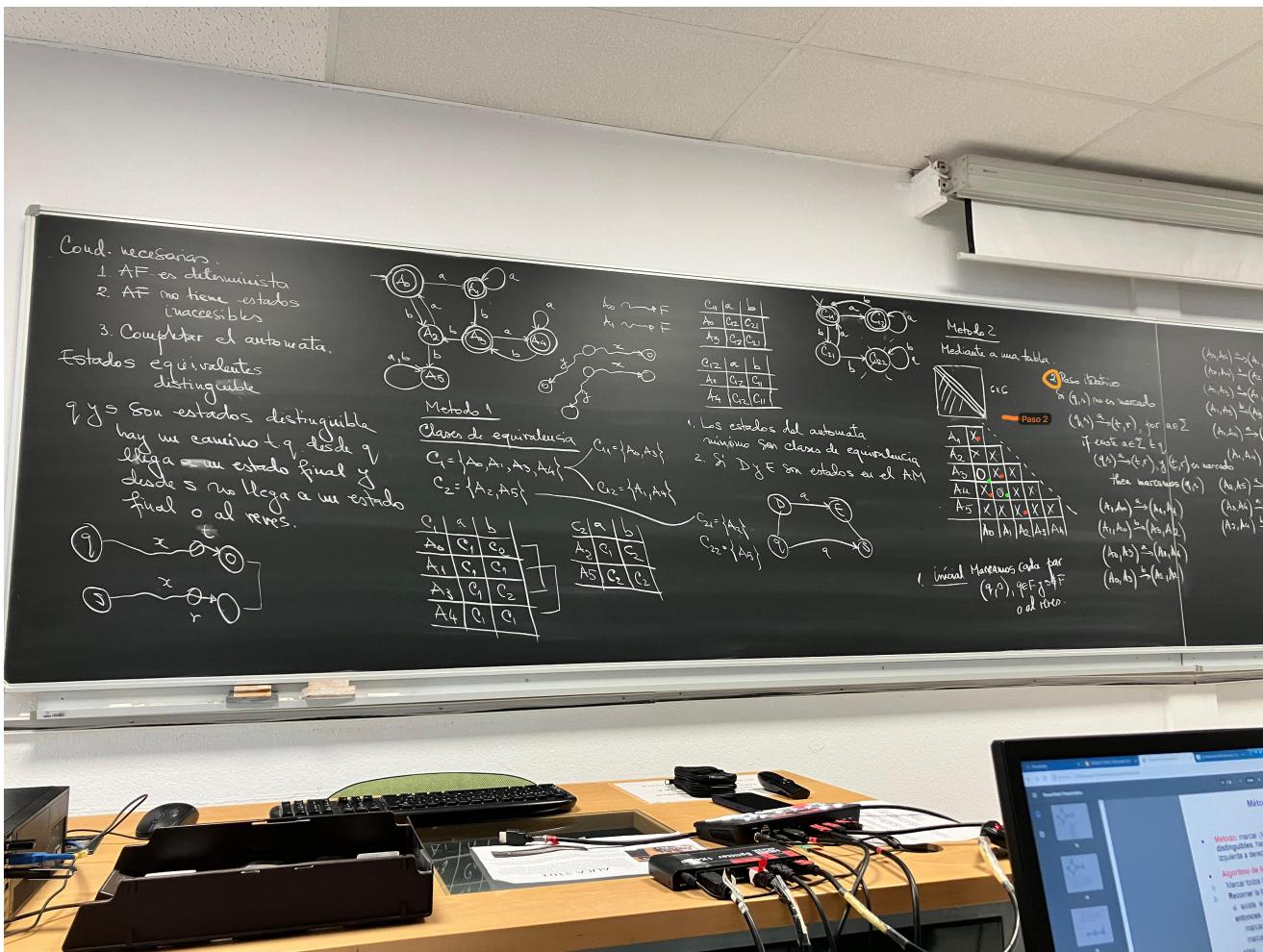
2. NFA to DFA



3. Epsilon Closures, Accessibility and Co-Accessibility

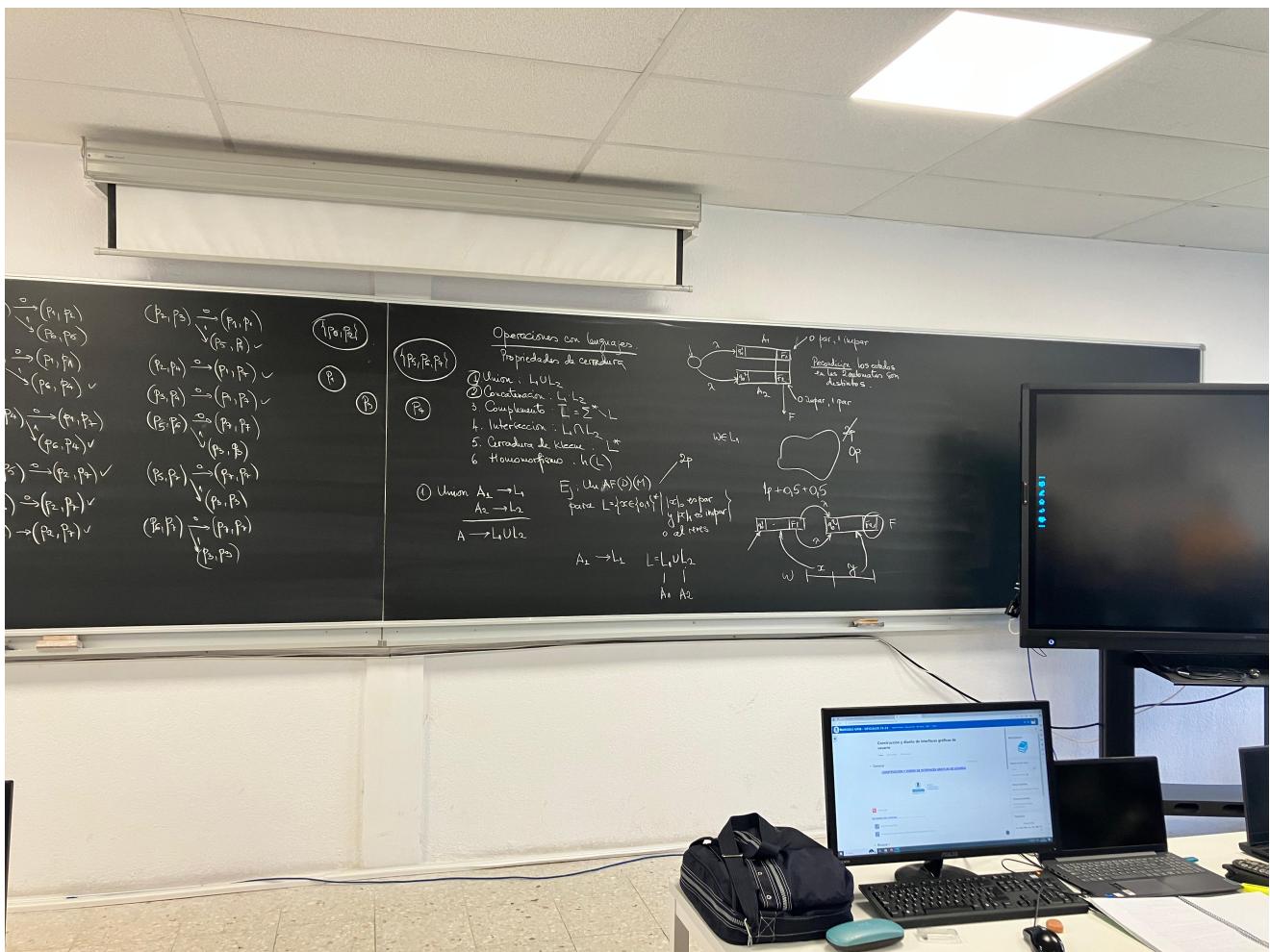


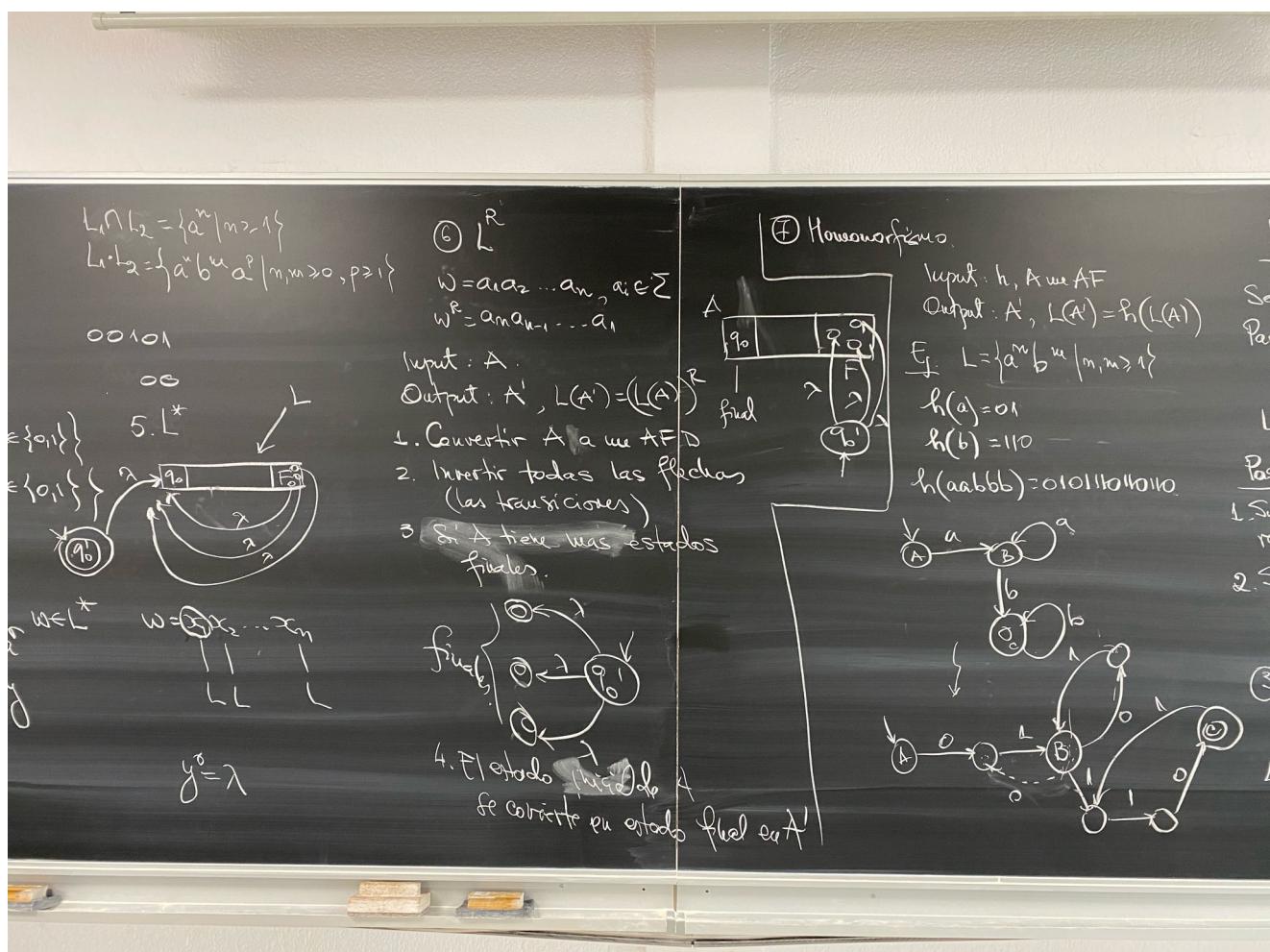
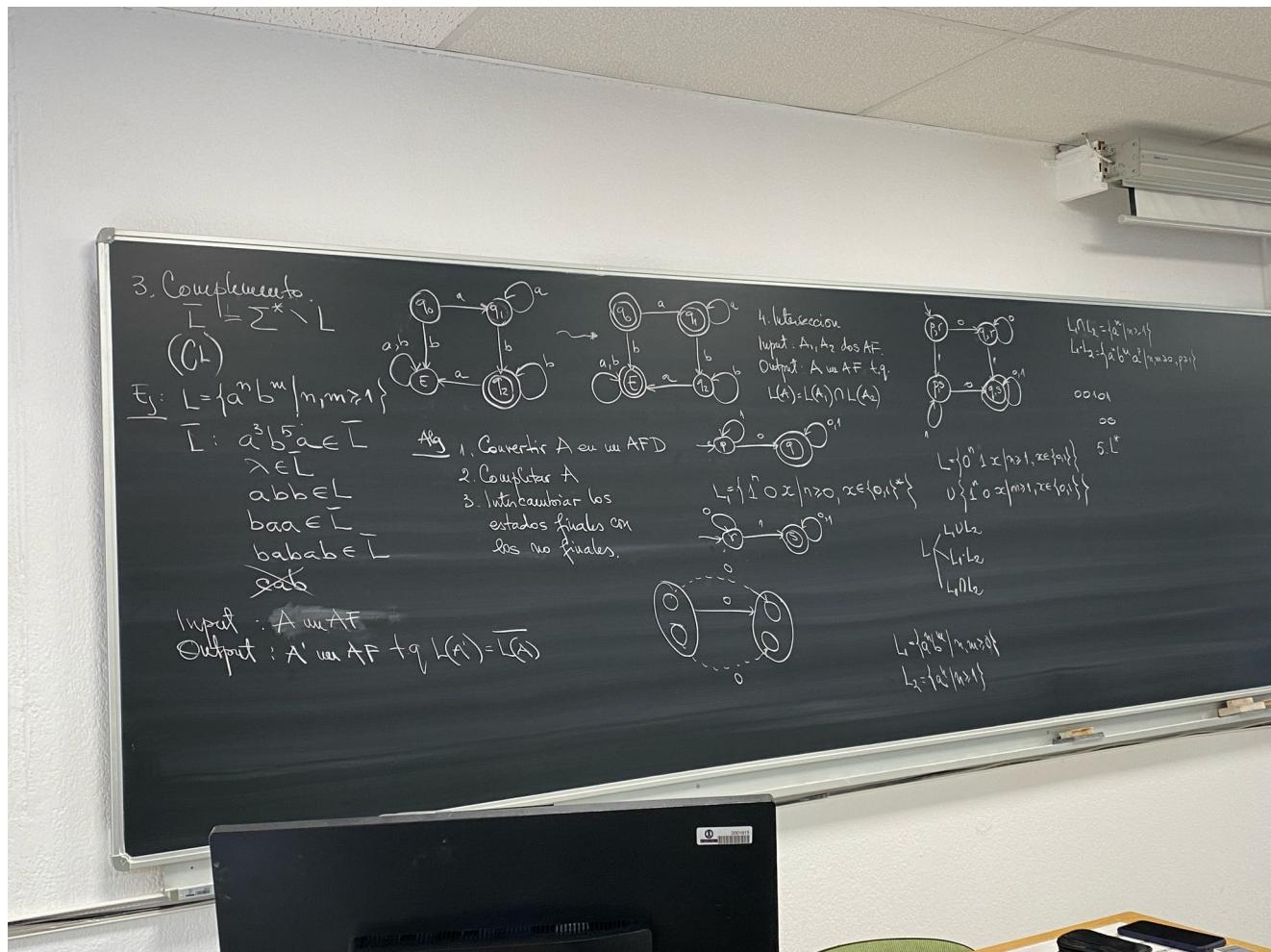
4. Minimization

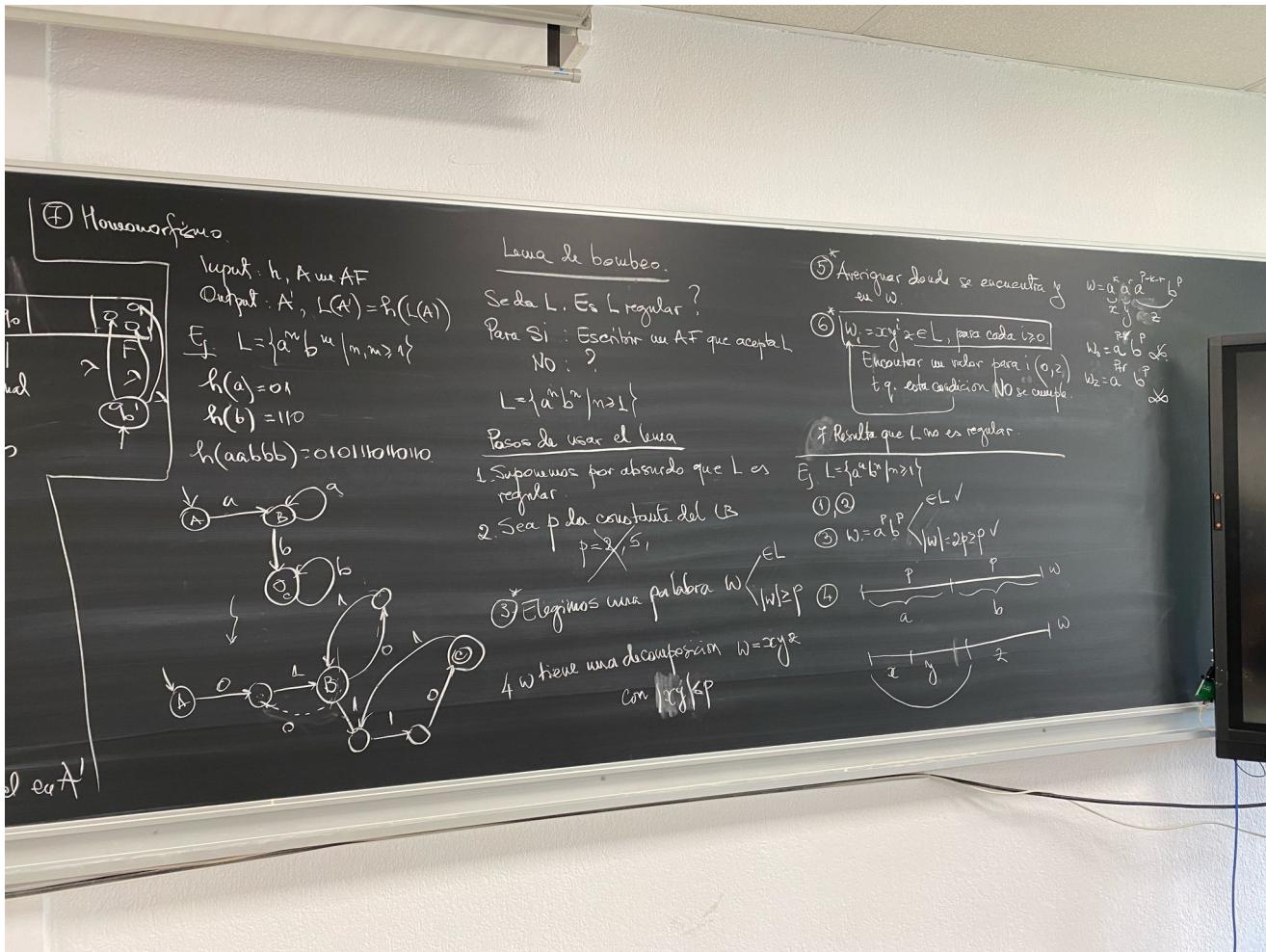


► Notes

5. Properties and Operations







6. Pumping

① Homomorfismos

Input: h , A we AF
Output: A' , $L(A') = h(L(A))$

Ej: $L = \{a^n b^m \mid n, m \geq 1\}$
 $h(a) = 01$
 $h(b) = 10$
 $h(aabb) = 0101101010$

Lema de bautros.
Sea L . Es L regular?
Para Si: Escribir un AF que acepta L
No: ?

$L = \{a^n b^m \mid n \geq 1\}$

Paso de usar el lema

- Suponemos por absurdo que L es regular.
- Sea p la constante del LB
 $p = 2 \leq 5$
- Elegimos una palabra $w \in L \setminus \{w \mid |w| \leq p\}$
- w tiene una decomposición $w = xyz$ con $|y| \leq p$

Resulta que L no es regular.

Ej: $L = \{a^n b^n \mid n \geq 1\}$

① ② $w = a^p b^p \in L$
③ $w = a^p b^p \not\in L$
④ $\overbrace{a}^p \overbrace{b}^p \overbrace{a}^p \overbrace{b}^p \dots \overbrace{a}^p \overbrace{b}^p$

$w = a^p b^p$

7. Grammars

$L(G) = \{w \in \Sigma^* \mid S \xrightarrow{*} w\}$

Ej: $A \xrightarrow{*} aBa$ no es reg
 $A \xrightarrow{*} aa \rightarrow \text{--}$
 $A \xrightarrow{*} B \rightarrow \text{--}$
 $a \xrightarrow{*} B \rightarrow \text{--}$
 $A \xrightarrow{*} b \rightarrow abbb$
 $A \xrightarrow{*} bA \rightarrow abb, bb$

$S \xrightarrow{*} aS \rightarrow aaS \rightarrow aabA$
 $S \xrightarrow{*} aS \rightarrow abA \rightarrow abbB \rightarrow$
 $bb \rightarrow \text{--}$
 $S \xrightarrow{*} bA \rightarrow bbbB \rightarrow$
 $S \xrightarrow{*} aS \rightarrow aaS$

$A \xrightarrow{*} \lambda$

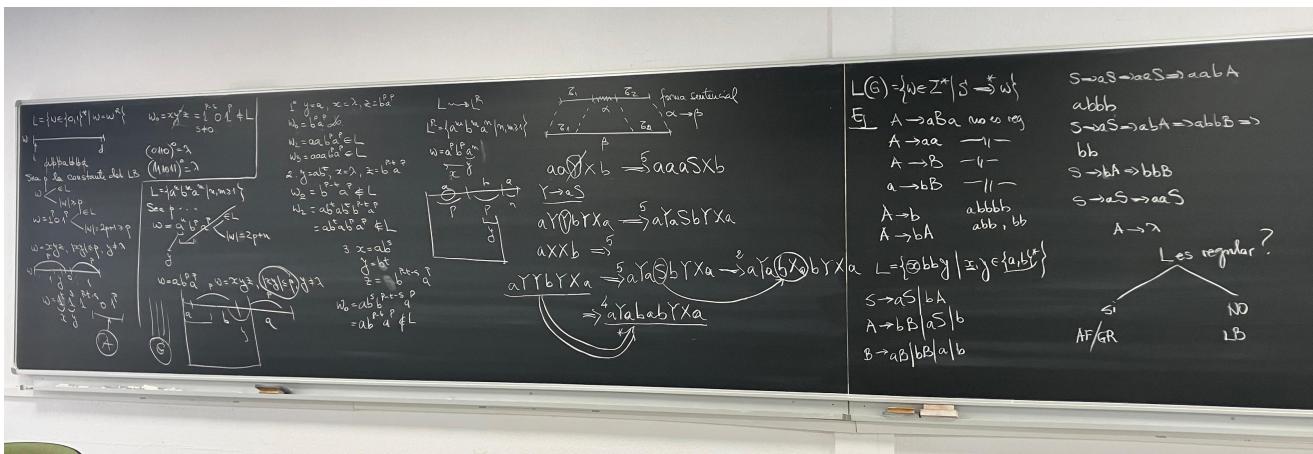
Los regulares?

$S \xrightarrow{*} aS \rightarrow aaS \rightarrow aabA$
 $A \xrightarrow{*} bB \rightarrow aS \rightarrow b$
 $B \xrightarrow{*} aB \rightarrow bB \rightarrow a/b$

Si AF/GR No LB

► Apuntes

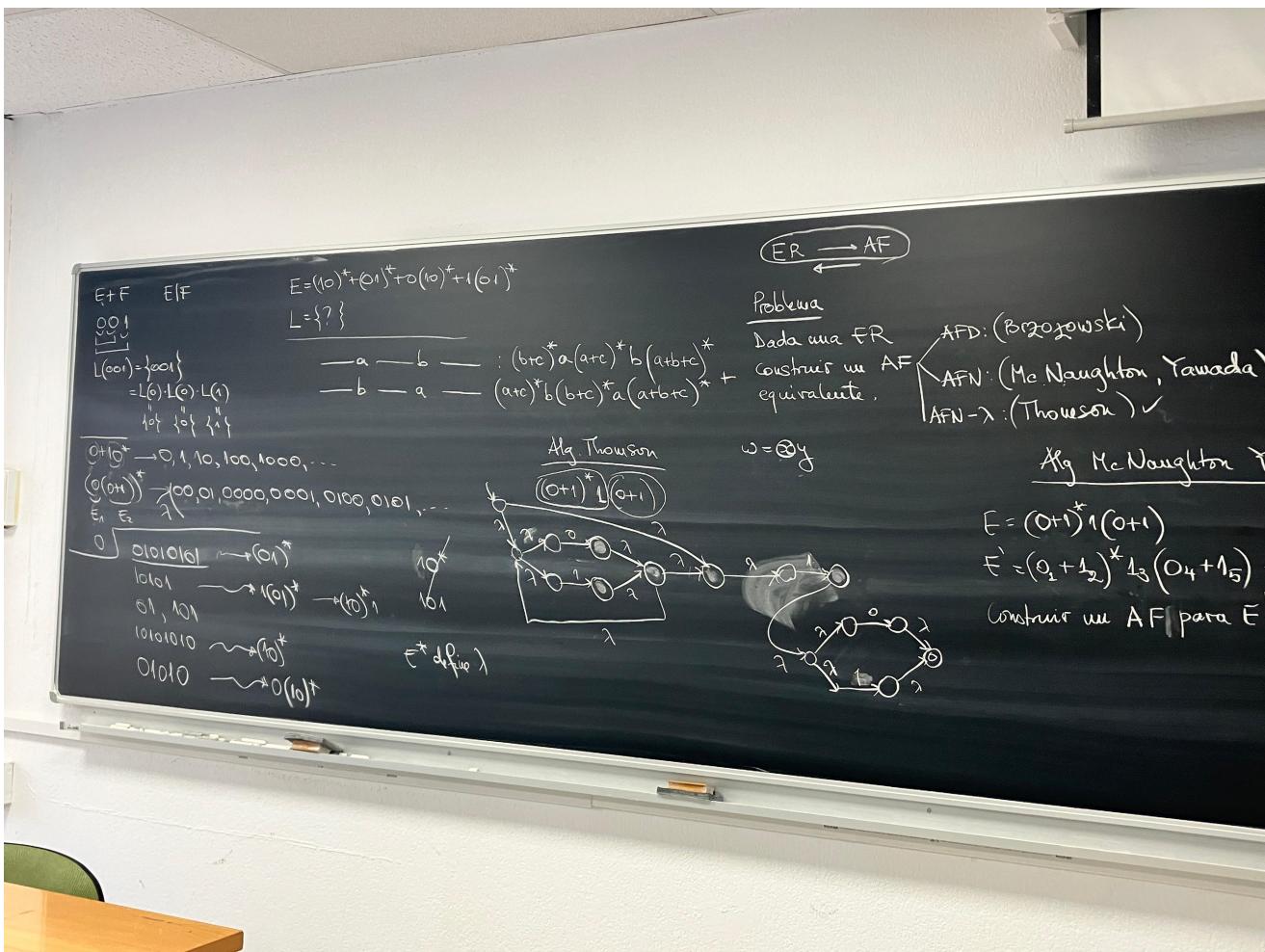
7. Grammar



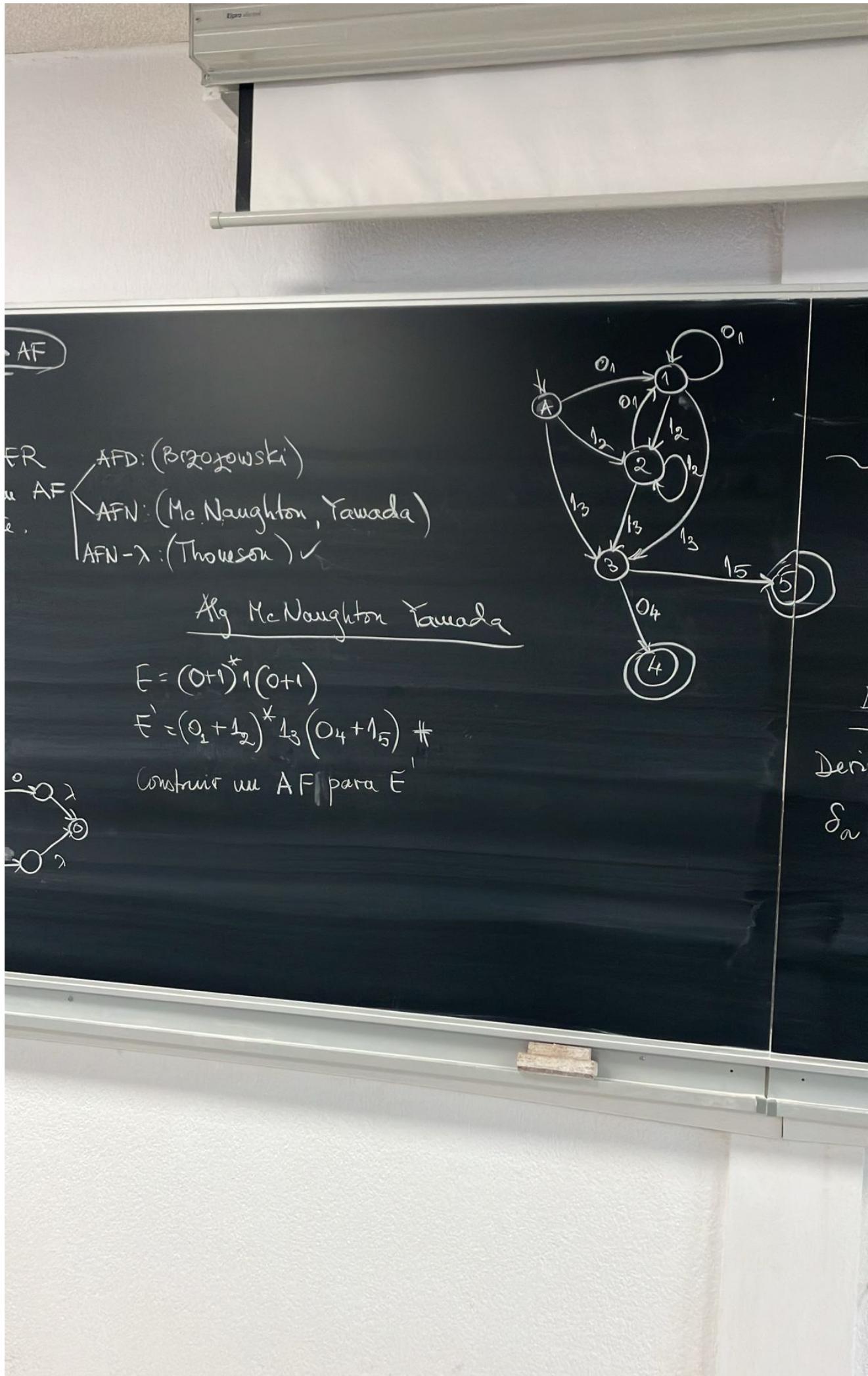
► Apuntes

8. ER a Automata

8.1. Thomson (NFA- ϵ)

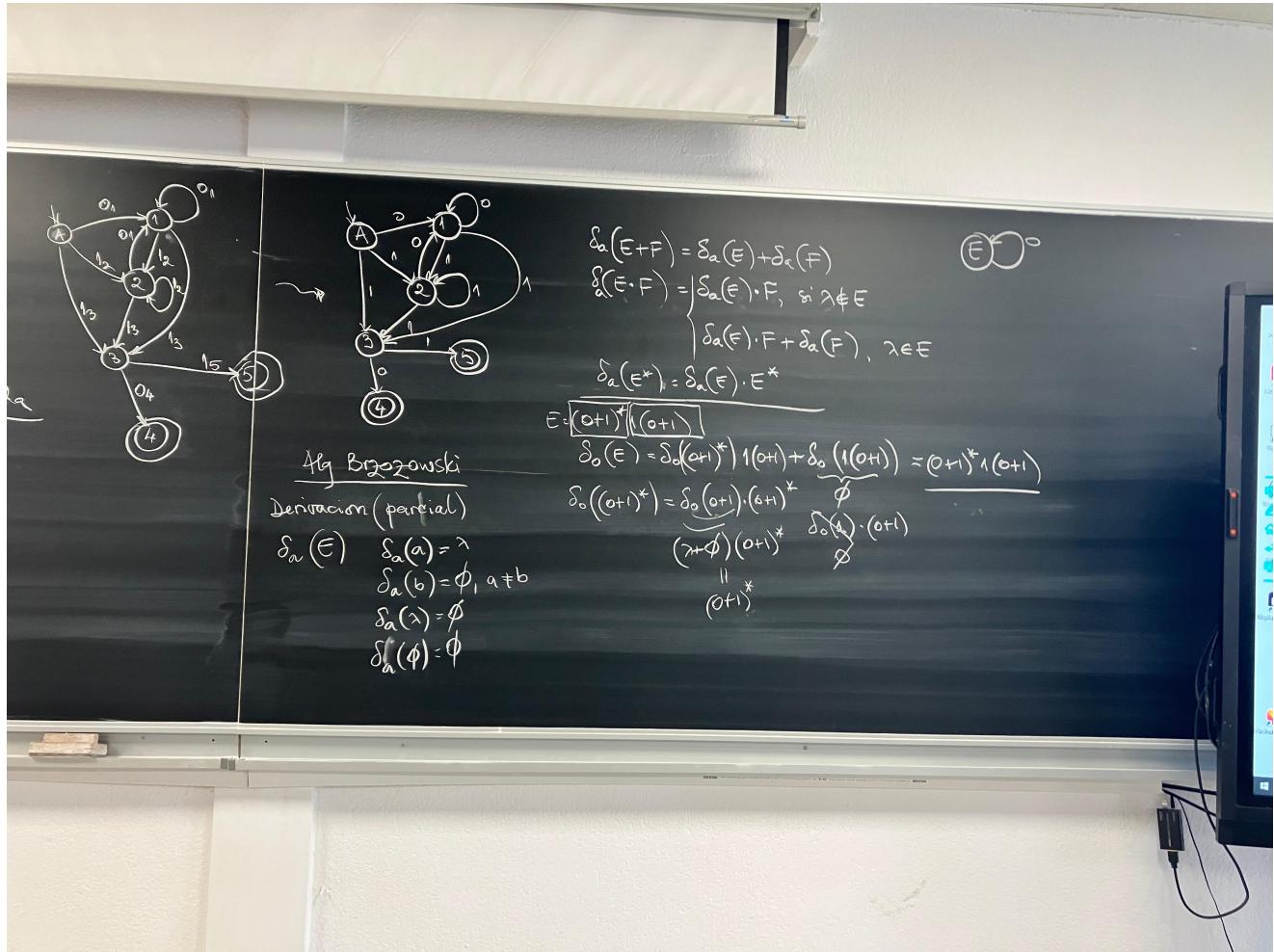


8.2. M'Naghten Yamada (NFA)





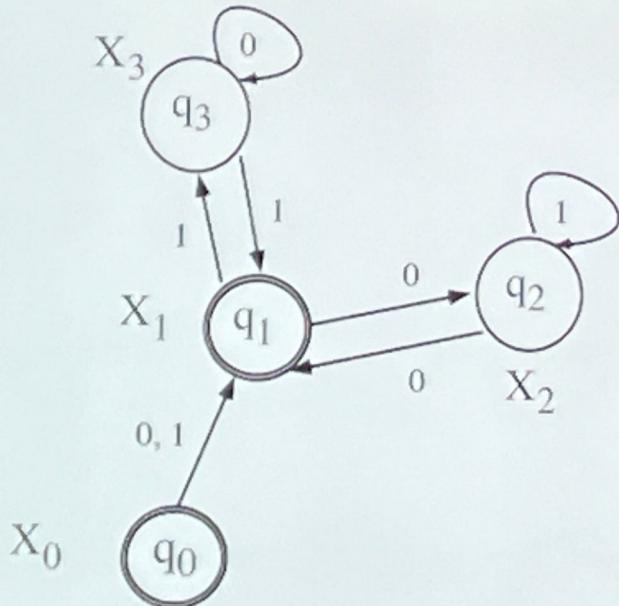
8.3 Brzozowski (DFA)



► Apuntes

9. Automata a ER

9.1. Mediante Sistema de Ecuaciones



$$X_0 = 0X_1 + 1X_1 + \lambda = (0+1)X_1 + \lambda$$

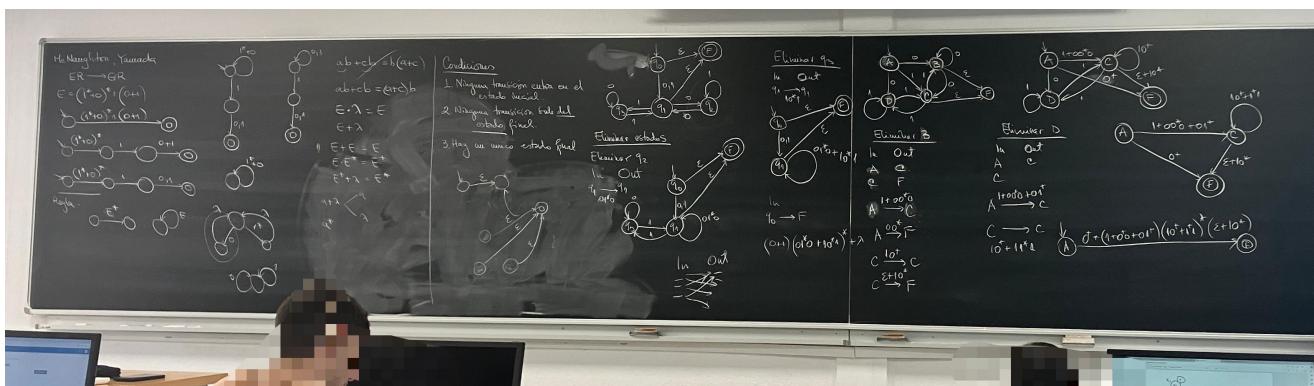
$$X_1 = 0X_2 + 1X_3 + \lambda$$

$$X_2 = 1X_2 + 0X_1$$

$$X_3 = 0X_3 + 1X_1$$

► Apuntes

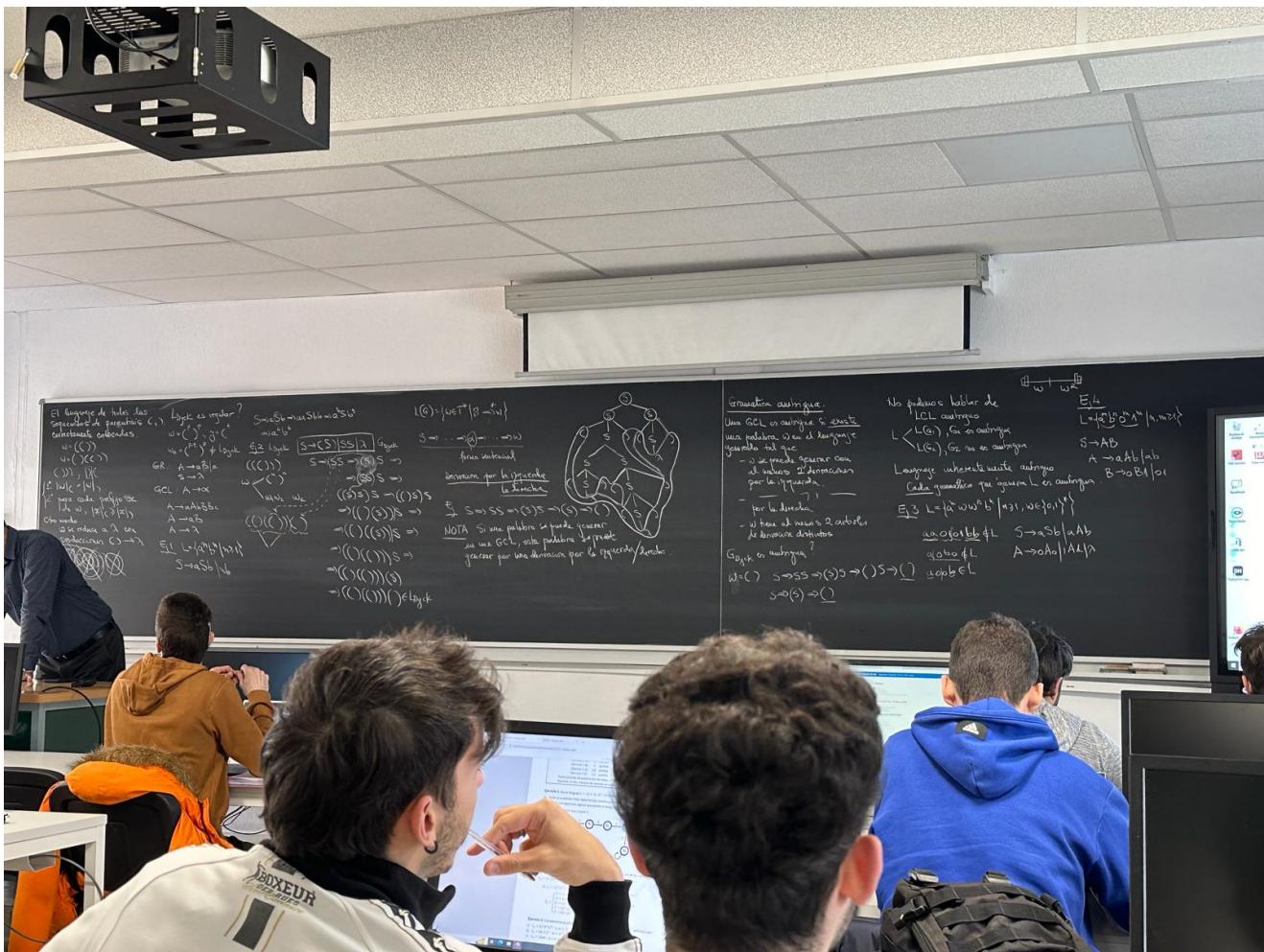
9.2. Mediante Eliminación de Estados



► Apuntes

10. ER a Gramatica - POR HACER

11. Gramáticas de Contexto Livre - GCL



11.1. Árbol de Derivación

Representación gráfica del proceso de derivación de una gramática.

- **Raíz:** Nodo inicial
- **Nodos:** Símbolos No Terminales
- **Hojas:** Símbolos Terminales

11.2 Gramáticas Ambiguas

Generan palabras por más de un árbol de derivación. No hay como detectar gramáticas ambiguas.

Es decir, se puede generar la misma palabra de distintas formas.

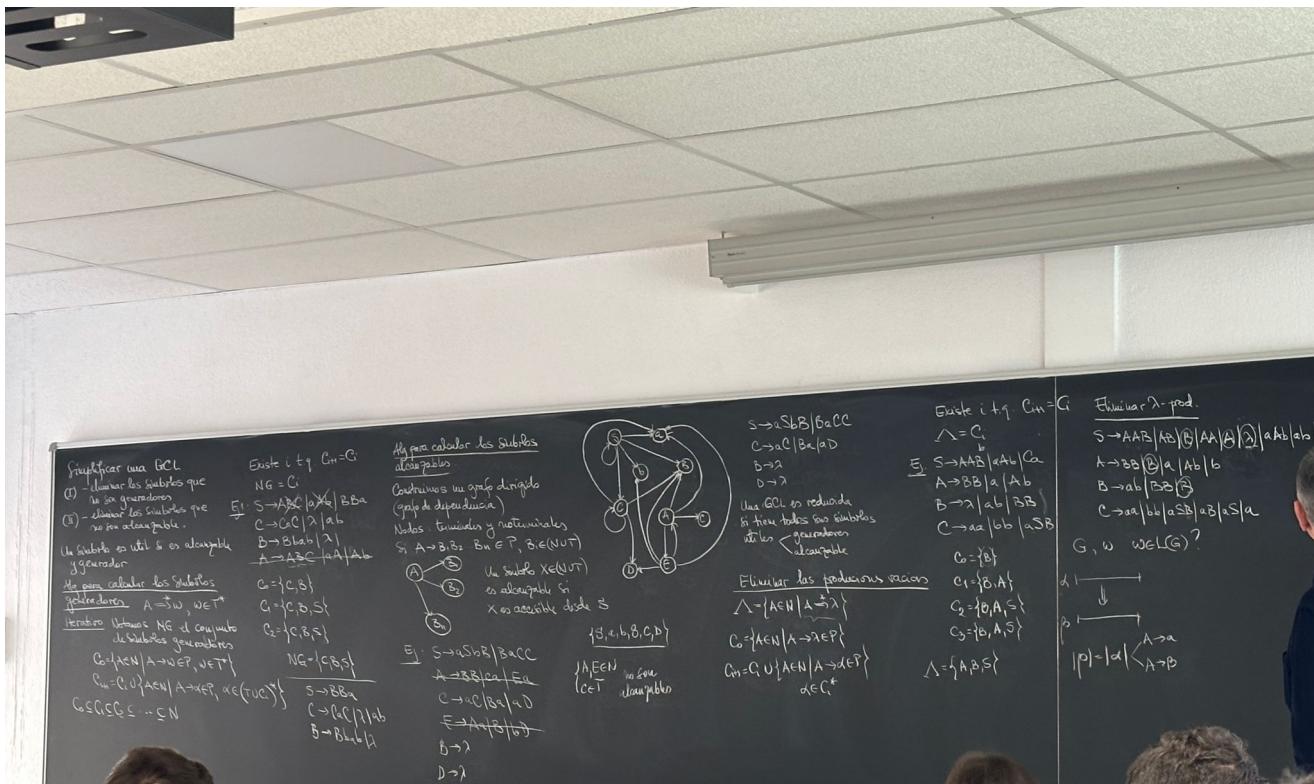
11.2.1. Lenguajes inherentemente ambiguos

Todas gramáticas que generan el lenguaje son ambiguas.

11.2.2. Grado de Ambigüedad

Cantidad de árboles de derivación que generan una palabra.

11.3 Limpiar Gramáticas



Consiste en quitar los símbolos inútiles e innaccesibles.

Note

El profe y las diapositivas ponen nombres distintos a términos de este apartado.

Diapositivas	Clase
Símbolos Inútiles	Símbolos No Generadores
Símbolos Inaccesibles	Símbolos No Alcanzables
Útil y Accesible	Útil

11.3.1. Símbolos Inútiles

Símbolos no terminales a través de los cuales no se puede llegar a una palabra.

11.3.2. Símbolos Inaccesibles

Símbolos que no se pueden alcanzar desde el símbolo inicial.

11.3.3. Eliminación de Símbolos Inútiles - No Generadores

El símbolo de la parte izquierda de una derivación directa $A \rightarrow w : w \in \sigma^*$ es útil si:

- Al menos un símbolo de la parte derecha de la derivación es útiles.

Note

El profe se complica demasiado la vida calculando C , solo hay q ir probando para cada A si se puede llegar a un terminal de alguna forma.

Repetir este proceso recursivamente.

11.3.4 Eliminación de Símbolos Inaccesibles - No Alcanzables

El símbolo de la parte derecha de una derivación directa $A \rightarrow w : w \in \sigma^*$ es accesible si:

- El símbolo de la parte izquierda de la derivación es accesible.

Repetir este proceso recursivamente.

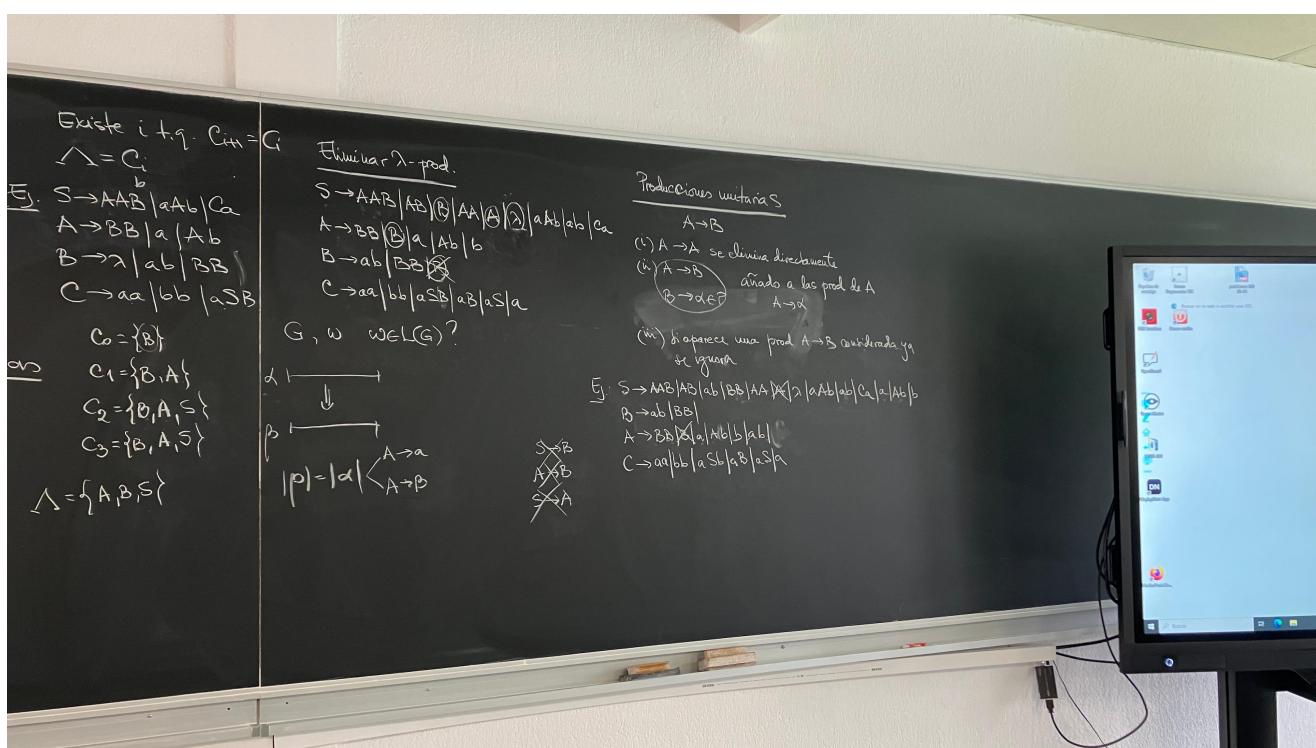
11.3.5 GCL Reducida

Todos sus símbolos son:

- **Alcanzables**
- **Generadores**

Es decir, son *Útiles*.

11.4 Transformar Gramáticas de Tipo 2



Se basa en eliminar **Producciones Unitarias** y **Producciones Vacías**.

11.4.1 Eliminar Producciones Vacías - λ -prod.

Se basa en actualizar las derivaciones quitando de cada Símbolo, transiciones lambda.

Note

Una forma que me sirve es escribir en una tabla el símbolo y su transición lambda y en filas las transiciones que derivan en el símbolo. Luego teniendo en cuenta que el símbolo ya no puede ser lambda, sacamos todas combinaciones posibles de las transiciones.

Prod. $B \rightarrow \lambda$	Prod. $A \rightarrow \lambda$	Prod. $S \rightarrow \lambda$	Prod $S \rightarrow \lambda$
$B \rightarrow BB$	B		
$S \rightarrow AAB$	AA	$A\ \lambda$	
$A \rightarrow BB$	$B\ \lambda$		
$C \rightarrow aSB$	aS		a
	$S \rightarrow aAb$	ab	
	$A \rightarrow Ab$	b	

Note

Al acabar, juntamos todas descomposiciones.

11.4.2 Eliminar Producciones Unitarias

Se basa en actualizar las derivaciones, de forma que no existan transiciones unitarias - del tipo $A \rightarrow B$.

- $A \rightarrow A$: Borramos.
- $A \rightarrow B$: Añadimos a A las derivaciones de B .

Forma guay: $\forall A \rightarrow B : B \rightarrow \alpha_n$, pasar a: $A \rightarrow \alpha_n$

11.5 Forma Normal de Chomsky - FNC

Cuando el LCL (Lenguaje de Contexto Libre) se genera por una gramática donde las producciones son de la forma:

- $A \rightarrow BC$
- $A \rightarrow a$

Para ello tenemos que:

1. Limpiar la Gramática

1. **Eliminar Producciones- λ**
2. **Eliminar Producciones Unitarias**
3. **Eliminar Símbolos Inútiles**
2. Producciones con **2 o más implicados** son siempre **no terminales**.
3. Producciones con **3 o más implicados** divididas en producciones de **dos variables**.

11.5.1 Paso 2

Para realizar el paso 2:

1. Creamos una nueva producción para cada producción existente donde haya dos o más implicados donde no son todos no terminales.
2. Intercambiamos en producciones originales los símbolos terminales por los nuevos símbolos no terminales.
3. Las producciones nuevas producirán los símbolos terminales.



Ejemplo:

- $A \rightarrow aAB \parallel B$
- $B \rightarrow b$

Como A produce tanto no terminales como terminales, crearemos una nueva producción que produzca a , quedando:

- $A \rightarrow CAB \parallel B$
- $B \rightarrow b$
- $C \rightarrow a$

11.5.2 Paso 3

Para realizar el paso 3:

1. Creamos una nueva producción para cada producción existente donde se produzca más de 2 terminales.
2. Intercambiamos en las producciones originales dos implicados por la nueva producción.
3. La producción nueva tendrá 2 de los implicados de la producción original.



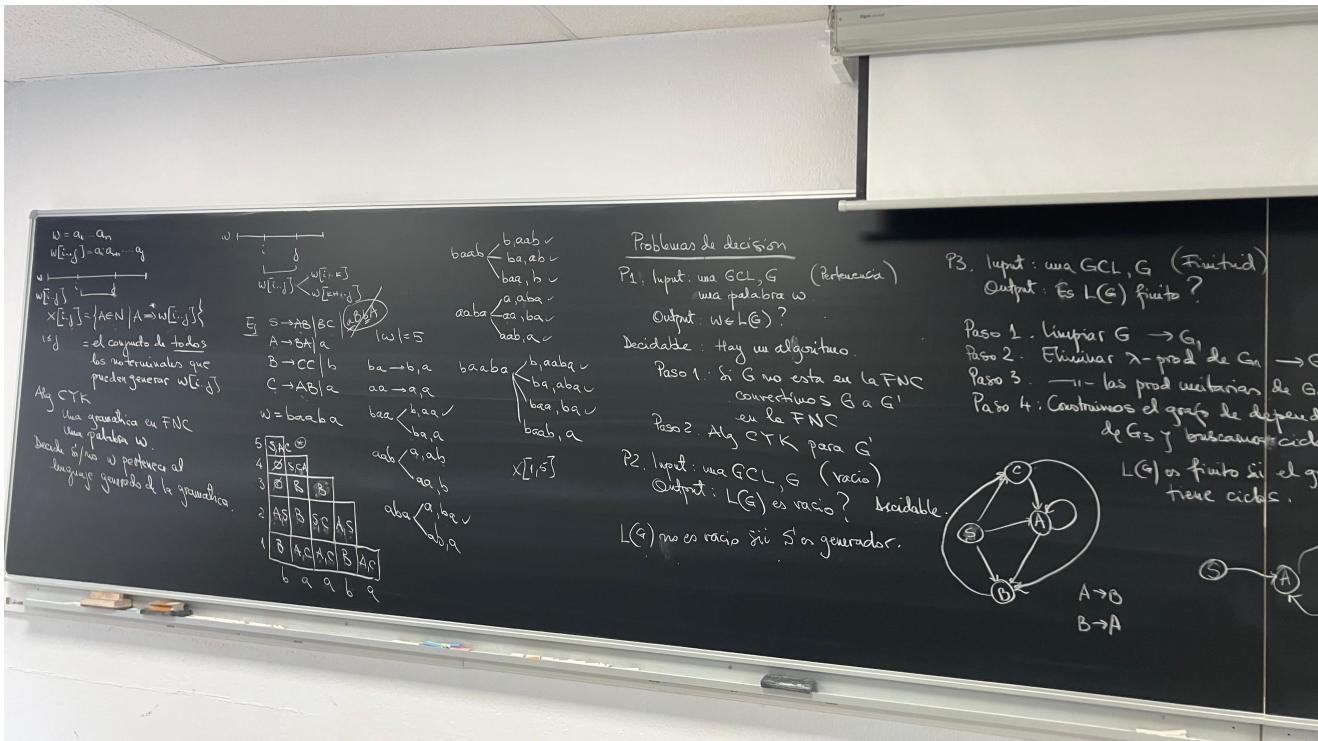
Ejemplo:

- $A \rightarrow CAB\|B$
 - $B \rightarrow b$
 - $C \rightarrow a$

Como *CAB* tiene tres implicados, crearemos una nueva producción que produzca 2 de ellos - **elegimos CA** -, quedando:

- $A \rightarrow DB \| B$
 - $B \rightarrow b$
 - $C \rightarrow a$
 - $D \rightarrow CA$

11.6 Algoritmo CYK



Dado una gramática en FNC y una palabra w , nos dice si la palabra pertenece al lenguaje generado por la gramática.

Warning

La gramática debe estar en FNC.

1. Construimos una matriz triangular inferior de $n \times x$.
 2. En el eje X escribimos carácter por carácter, la palabra a probar (w).
 3. Escribimos en la fila más baja el conjunto de símbolos que nos posibilita llegar directamente al carácter de abajo.

4. Escribimos en la fila arriba de la anterior, el conjunto de símbolos que nos posibilita llegar a la palabra actual - *teniendo en cuenta todas combinaciones posibles*. Para esto tenemos en cuenta los resultados que hemos sacado anteriormente de cada conjunto de caracteres.

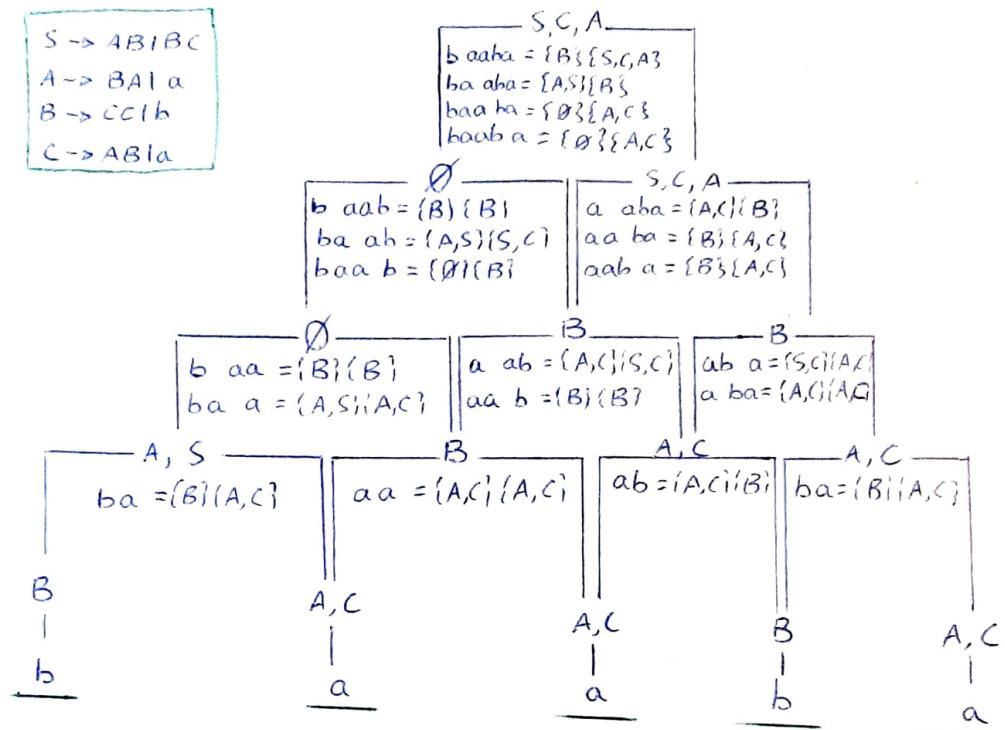
Note

Personalmente, prefiero hacer una pirámide que un triangulo. Es decir, poner el resultado de la fila superior entre las columnas de abajo.

Ejemplo:

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

$$w = baaba$$



1. Escribimos los caracteres.
2. Para cada carácter, buscamos símbolos que les produzcan. *Ej. Para "a", los símbolos que le producen son: A y C, por lo tanto lo apuntamos.*
3. Para cada palabra, para cada posible descomposición, buscamos símbolos que les produzcan. *Ej. Para la palabra "ba", su descomposición es: "b a", por lo tanto su conjunto de símbolos implicados será la combinatoria entre los símbolos que producen "b" y de los que producen "a", es decir: $\{B\} \times \{A, C\} = \{BA, BC\}$, ahora buscamos producciones que contienen estos implicados y apuntamos los implicantes.*
4. Repetimos el paso 3 hasta que toda fila de \emptyset o que lleguemos, a un valor.

Note

Podemos comprobar que $w \in L$, ya que tenemos una serie de símbolos de la gramática que juntos generan esa palabra. En este caso: S, C, A .

11.7 Ejemplo: $w \in L(G)$

Para probar que sí, tenemos que ser capaces de crear un árbol de derivación por la izquierda.

Input: una gramática de contexto libre G
 una cadena w
 Output: $w \in L(G)$?
 Si \rightarrow Árbol de derivación
 Derivación por la izquierda

G es una GCL arbitraria
 ① Limpiar/simplificar G
 • eliminar los símbolos no generadores
 • eliminar los símbolos inaccesibles
 ② Eliminar las producciones vacías (λ -prod)
 ③ Eliminar las producciones unitarias

Forma Normal Chomsky
 $S \rightarrow \lambda$
 $A \rightarrow a, A \in N, a \in T$
 $A \rightarrow BC, A, B, C \in N$

$G \xrightarrow{①, ②, ③, ④, ⑤} G'$
 (en FNC)

④ $A \rightarrow \alpha \begin{cases} \alpha \neq \lambda \\ \alpha \in FNC \end{cases} (A \rightarrow B)$
 $A \rightarrow abBAaB \rightarrow \begin{cases} A \rightarrow XTBAXB \\ X \rightarrow a \\ Y \rightarrow b \end{cases}$
 $\boxed{A \rightarrow BBAB} \vee$
 $A \rightarrow aababc \rightarrow \begin{cases} A \rightarrow XXXYXYZ \\ Z \rightarrow c \end{cases}$

Árbol: transformar las prod no permitidas de la FNC en

$A \rightarrow B_1 B_2 \dots B_n, B_i \in N, n \geq 3$

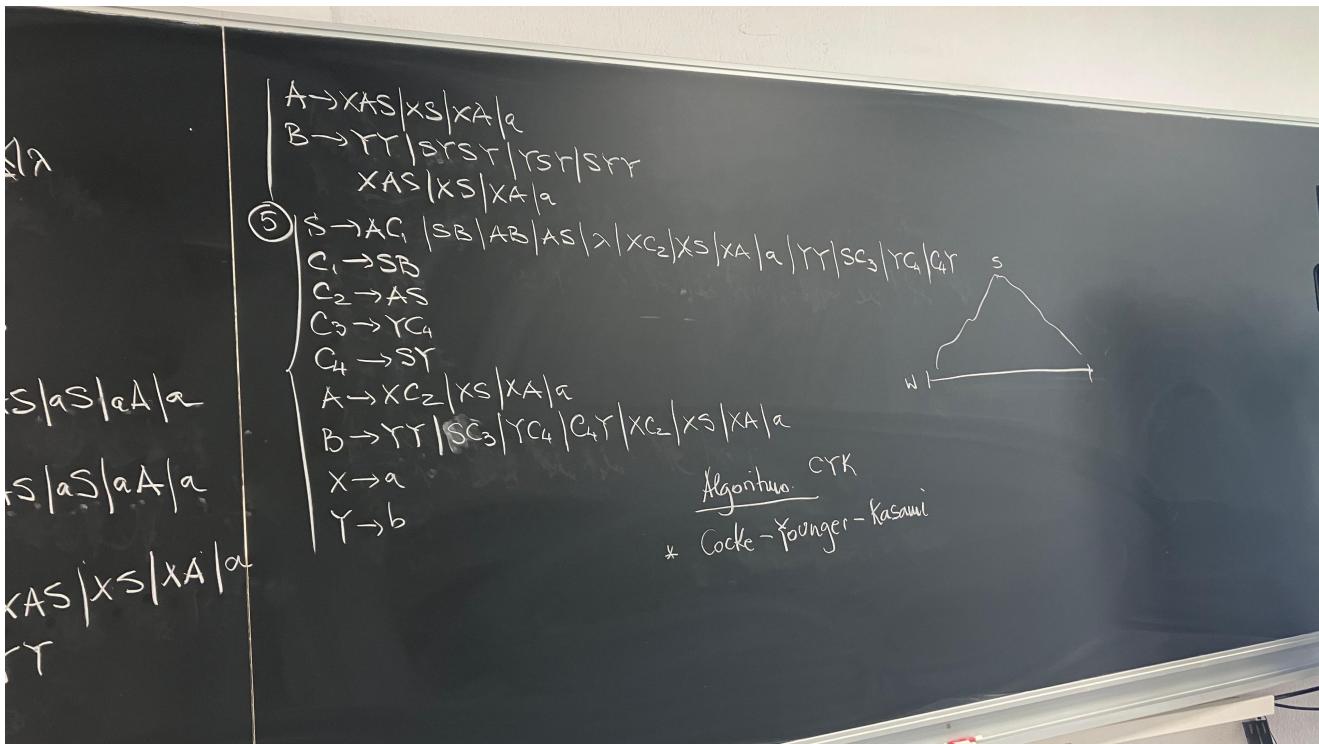
⑤ $A \rightarrow B_1 \dots B_n$
 $\left\{ \begin{array}{l} A \rightarrow B_1 C_1 \\ C_1 \rightarrow B_2 C_2 \\ \vdots \\ C_{n-2} \rightarrow B_{n-1} B_n \end{array} \right.$

Ej ⑤
 $\underline{A \rightarrow XTBAXB}$
 $\begin{cases} A \rightarrow XC_1 \\ C_1 \rightarrow YC_2 \\ C_2 \rightarrow BC_3 \\ C_3 \rightarrow AC_4 \\ C_4 \rightarrow XB \end{cases}$

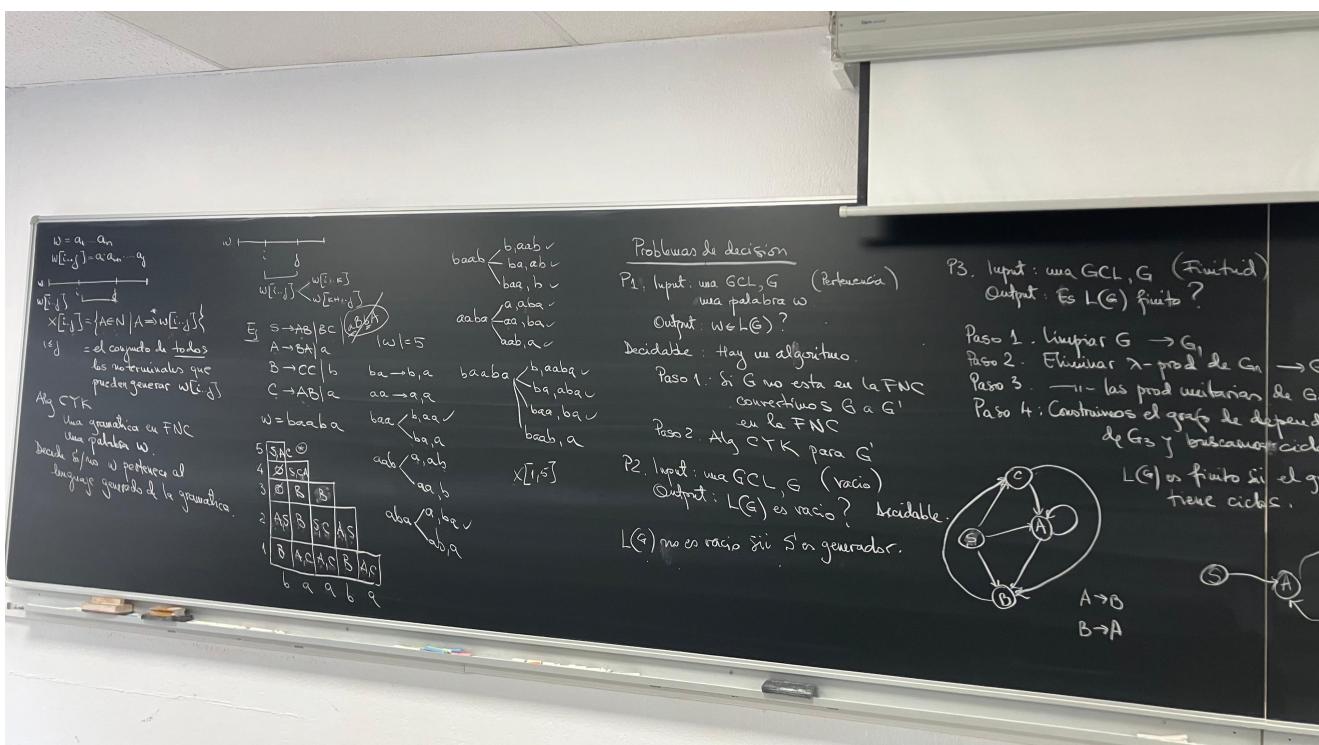
$A \Rightarrow XC_1 \Rightarrow XYC_2$
 $\Rightarrow XYBC_3 \Rightarrow XYTBAC_4$
 $\Rightarrow XYTBAXB$

Ej. FNC
 $S \rightarrow A$
 $A \rightarrow B$
 $B \rightarrow C$
 $C \rightarrow D$
 ① L
 Gen
 Acc

<p>une symétrie de contrôles être G mais l'autre w ne L(G) ?</p> <p>Si \rightarrow Autre le deuxième dénombrer par la droite</p> <p>GCL automata et transférer G l'autre les sorties les sorties éliminer les sorties (inaccessibles) sur les producteurs rares (> 1)</p> <p>les producteurs rares A \rightarrow B₁ B₂ B₃ B₄</p> <p>Forces Normal Chomsky</p> <p>$A \rightarrow A_1 A_2$ $B \in C$</p>	<p>G ① ② ③ ④ ⑤ G' (c) FNC</p> <p>Autre \rightarrow x $\notin N$ ($A \rightarrow B$) autre \rightarrow x $\in N$ ($A \rightarrow B$)</p> <p>$A \rightarrow abBaB \rightarrow$ {$A \rightarrow XTBAXB$, $X \rightarrow a$, $T \rightarrow b$, $B \rightarrow bC_1C_2$, $C_1 \rightarrow AC_1$, $C_2 \rightarrow BC_2$, $C_3 \rightarrow AC_3$, $C_4 \rightarrow XB$}</p> <p>$A \rightarrow abab \rightarrow$ {$A \rightarrow XXYYXYZV$, $Z \rightarrow c$}</p> <p>Autre, transférer les prod un peu toutes de FNC sur</p> <p>$A \rightarrow B_1 B_2 B_3 B_4$</p> <p>Autre, dénombrer par la droite</p>	<p>E ⑥</p> <p>$A \rightarrow XTBAXB$</p> <p>$A \rightarrow XG$ $C_1 \rightarrow YC_1$ $C_2 \rightarrow BC_2$ $C_3 \rightarrow AC_3$ $C_4 \rightarrow XB$</p> <p>E FNC</p> <p>$S \rightarrow ASB SB AB AS$ $A \rightarrow aS aA a$ $B \rightarrow bSb SbSb bSb Sbb$</p> <p>① Uniquer</p> <p>Generaliser {A, B, C, S} Accessibles {S, A, B}</p> <p>$A \Rightarrow XG \Rightarrow XTC_2$ $\Rightarrow XTC_3 \Rightarrow XTBAC_4$ $\Rightarrow XYBAZR$</p> <p>$S \rightarrow ASB SB AB$ $A \rightarrow aS a$ $B \rightarrow A bb Sb Sb$</p>	<p>② Flétrir λ-prod</p> <p>$\lambda = a, b, S$ $S \rightarrow ASB SB AB AS$ $A \rightarrow aS aA a$ $B \rightarrow bSb SbSb bSb Sbb$</p> <p>③ Flétrir les prod unitaires</p> <p>$A \rightarrow A$ $B \rightarrow bSb SbSb Sbb SbSb SbSb Sbb$ $S \rightarrow a$ $S \rightarrow ASB SB AB AS aS aA a$ $bSb Sbb SbSb Sbb$</p> <p>④ S $\rightarrow ASB SB AB AS a$ $XAS XS KA a$ $YY YY SYSY YSY YY$ $X \rightarrow a$ $Y \rightarrow b$</p>	<p>$A \rightarrow XAS XS KA a$ $B \rightarrow YY SYSY YSY YY SYYY$ $XAS XS KA a$</p> <p>⑤ S $\rightarrow AC_1 SB AB AS a XC_2 XS KA a YY$ $C_1 \rightarrow SB$ $C_2 \rightarrow AS$ $C_3 \rightarrow YC_1$ $C_4 \rightarrow SY$ $A \rightarrow XC_2 XS KA a$ $B \rightarrow YY SC_3 YC_4 CY XC_2 XS KA a$ $X \rightarrow a$ $Y \rightarrow b$</p> <p>Algorithm CTK * Cocke - Forger -</p>
---	---	--	---	---



11.8 Problemas de Decisión



11.8.1 Problema - Pertenencia de Palabra

$w \in L(G)$?

Para resolver este problema:

1. Pasar G a **FNC** $\rightarrow G_1$.
 2. Realizar **CYK** para w en G_1 .

11.8.2 Problema - Lenguaje Vacío

$L(G)$ vacío?

Para resolver este problema:

- Probar si S es **generador**.

11.8.3 Problema - Lenguaje Finito

$L(G)$ finito?

Para resolver este problema:

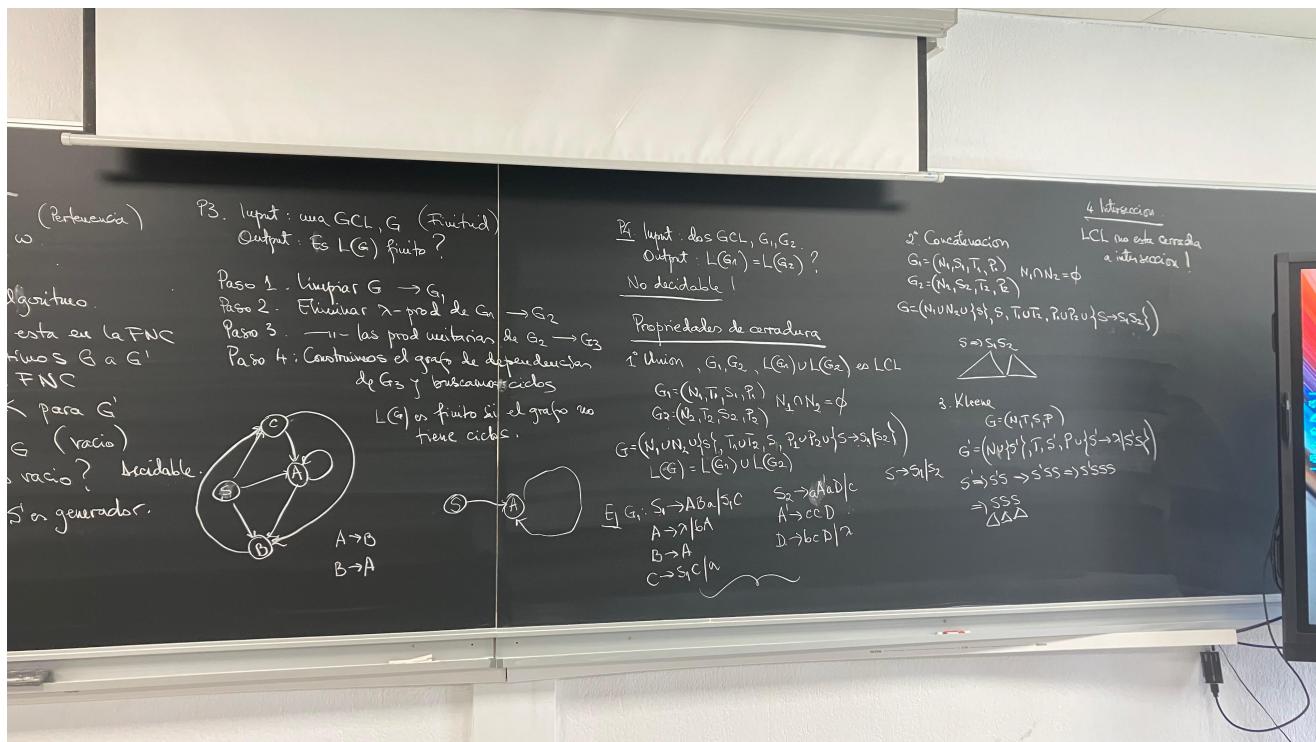
1. Limpiar: $G \rightarrow G_1$.
2. Eliminar λ -prod: $G_1 \rightarrow G_2$.
3. Eliminar prod. unitarias: $G_2 \rightarrow G_3$.
4. Construir grafo de dependencia de G_3 .

$L(G)$ será infinito cuando exista al menos un ciclo - Ej. $A \rightarrow AB$.

11.8.4 Problemas no Decidibles

- GCL **ambigua**?
- LLC **inherente amibiguo**?
- GLC \cap GLC = \emptyset ?
- GLC = GLC?

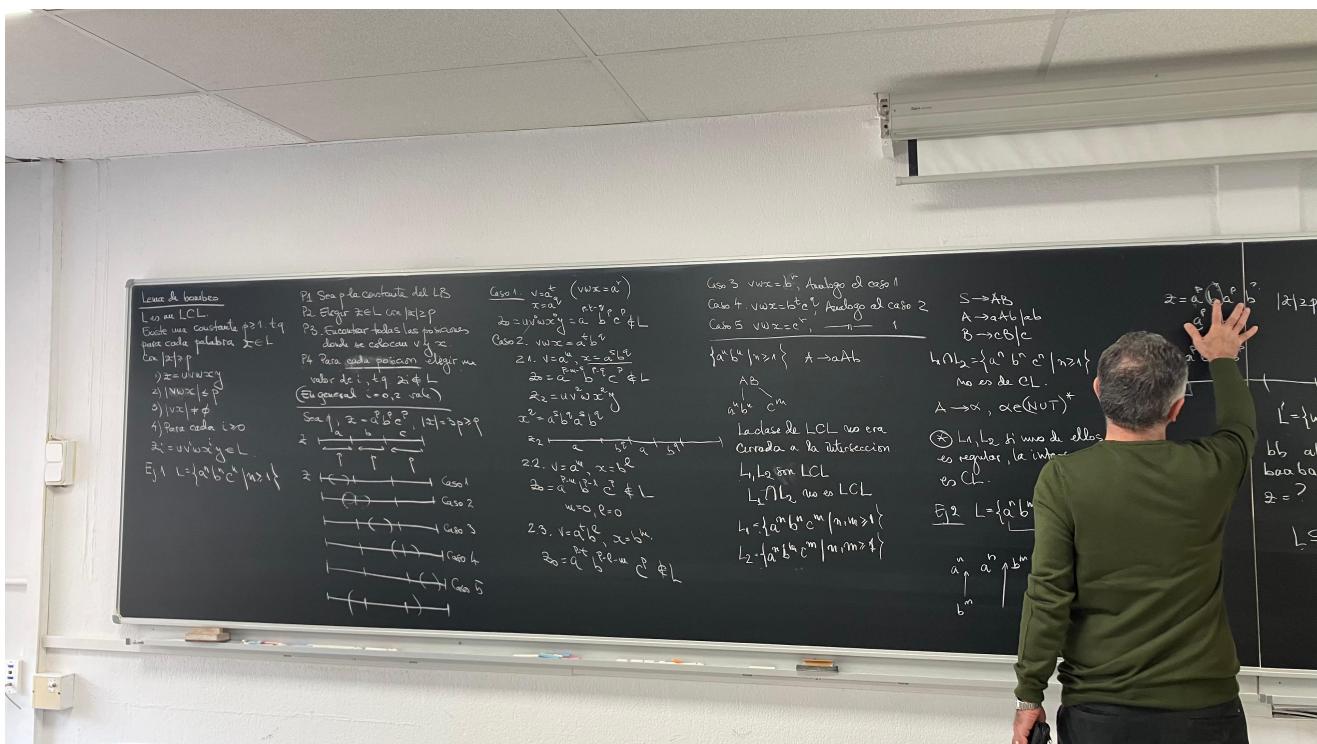
11.9 Propiedades de Cerradura

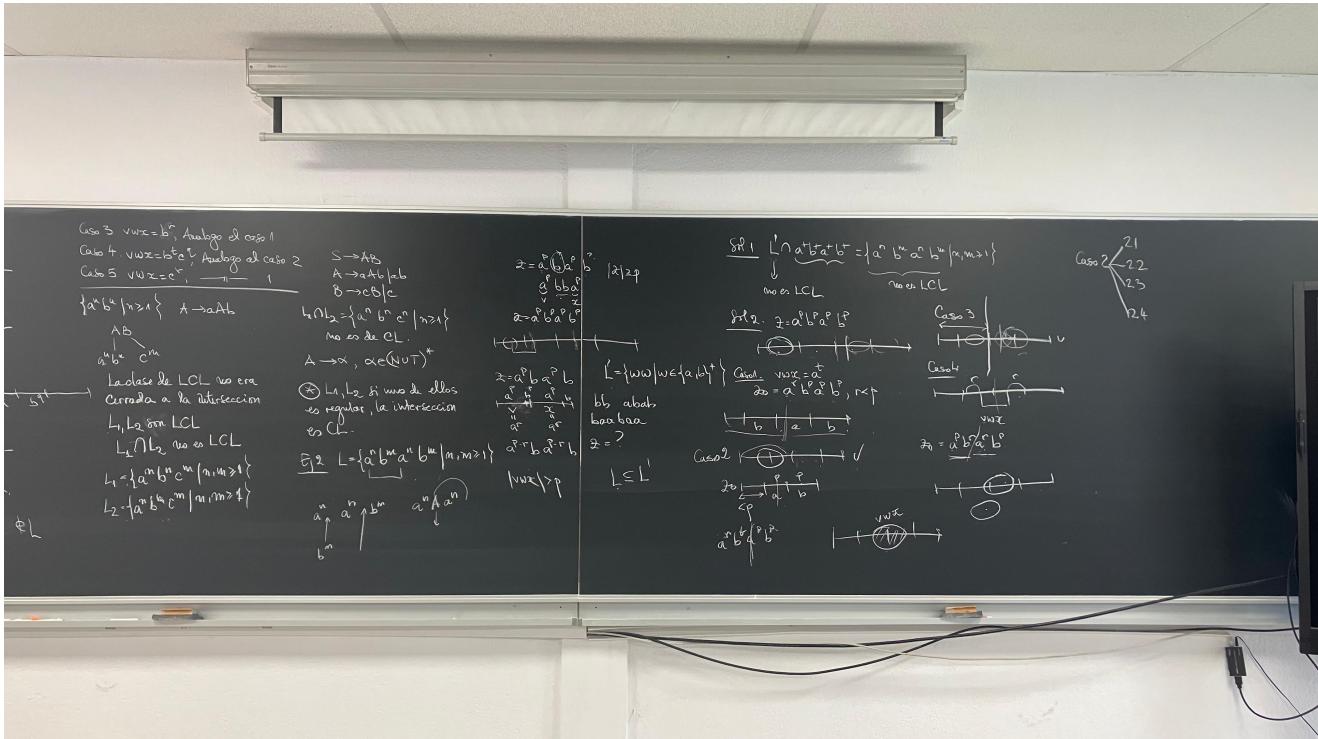


Sean L_1, L_2 LLC's y L_3 LR :

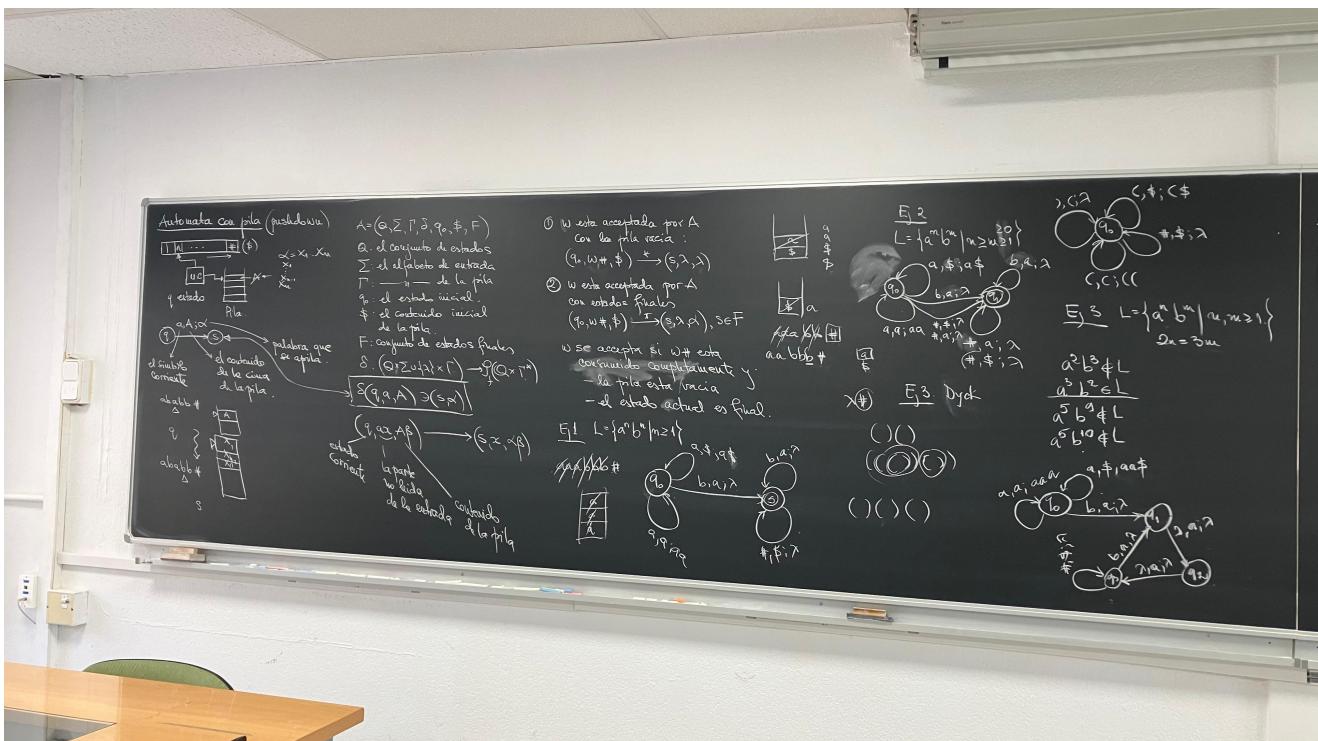
PROPIEDAD	OPERACIÓN	RESULTADO
Unión	$L = L_1 \cup L_2$	LLC
Intersección - LLC	$L = L_1 \cap L_3$	LLC
Intersección - LR	$L = L_1 \cap L_2$	¿?
Concatenación	$L = L_1 L_2$	LLC
Kleene	$L = L_1^*$	LLC
Inversión	$L = L_1^R$	LLC
Complementario	$L = L_1^C$	¿?
Diferencia	$L = L_1 - L_2$	¿?

11.10 Lema de Bombeo - GCL [POR HACER]





11.10 Autómata de Pila - AP



Se define como:

$$AP = (\Sigma, Q, \Gamma, \delta, q_0,], F)$$

- Σ : Alfabeto de las palabras.
 - Q : Conjunto de todos estados.
 - Γ : Alfabeto de la pila.
 - δ : Función de transición - $(q_i, a, A) \rightarrow (q_j, BB)$.
 - q_i : Estado actual.

- a : Símbolo de entrada.
- A : Elemento que se quita de la pila.
- q_j : Estado destino.
- BB : Elementos para poner en la pila.
- q_0 : Estado inicial del autómata.
- $]$: Símbolo inicial de la pila.
- F : Estados finales del autómata.

 Note

En mis apuntes los "Centinelas" - *caracteres que delimitan un valor* - son:

- Centinela de **Entrada** : $\# \rightarrow]$
- Centinela de **Pila** : $\$ \rightarrow \}$

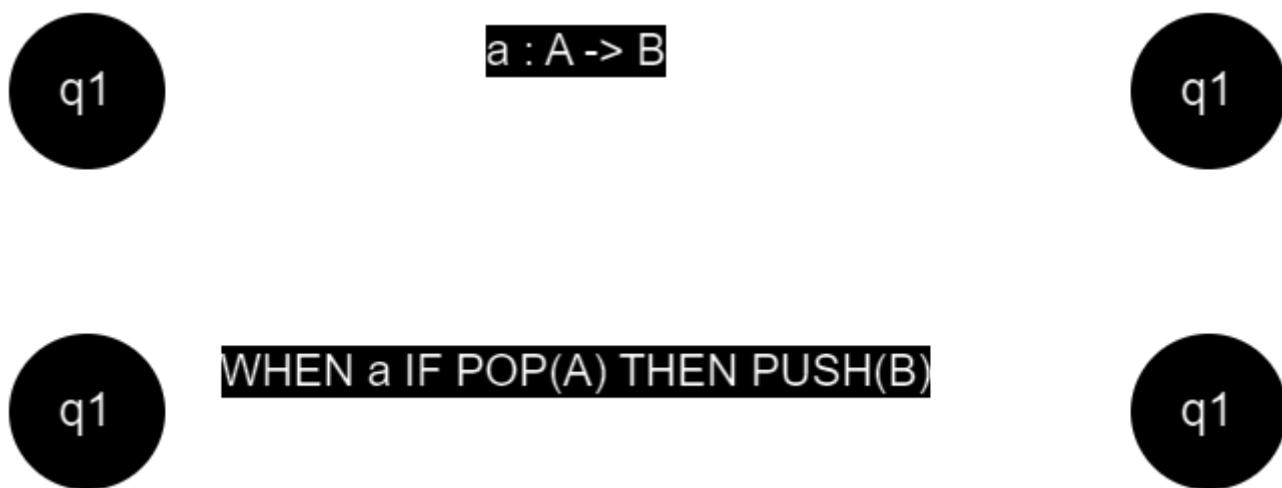
11.10.1 Representación Gráfica - Grafos

Se hace igual a que en grafos de expresiones regulares, excepto:

- Transiciones (δ) : Flechas. Con etiquetas $a; A; BB$
 - a : Símbolo de entrada.
 - A : Elemento que se quita de la pila.
 - BB : Elementos para poner en la pila.

 Note

Unas formas más intuitivas - *para mí* - de entender las transiciones en los grafos son:



11.10.2 Inicialización

Antes de empezar a consumir entradas, el AP inicializa la:

- **Entrada**, añadiendo al final de la palabra el **Centinela de Entrada**.
- **Pila**, añadiendo - *pusheando* - el **Centinela de Pila**.

11.10.3 Consumición

Un AP funciona de la siguiente forma:

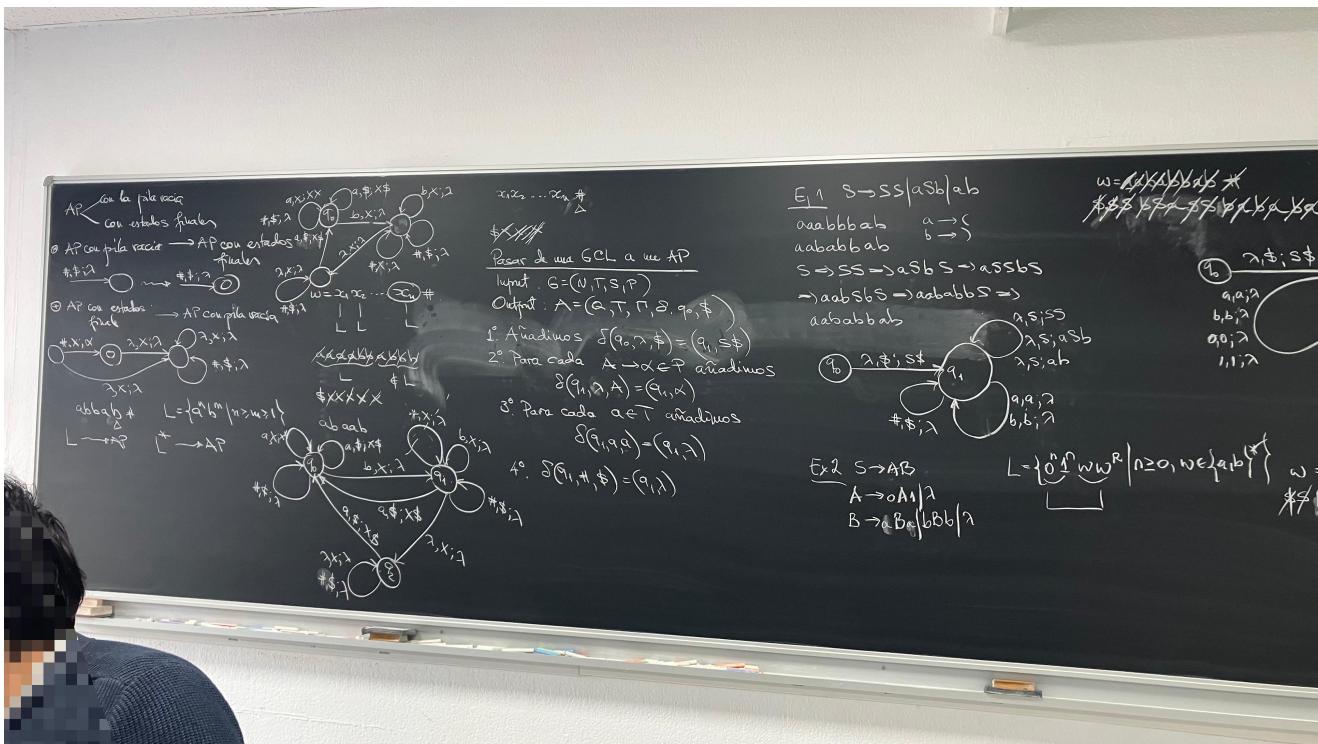
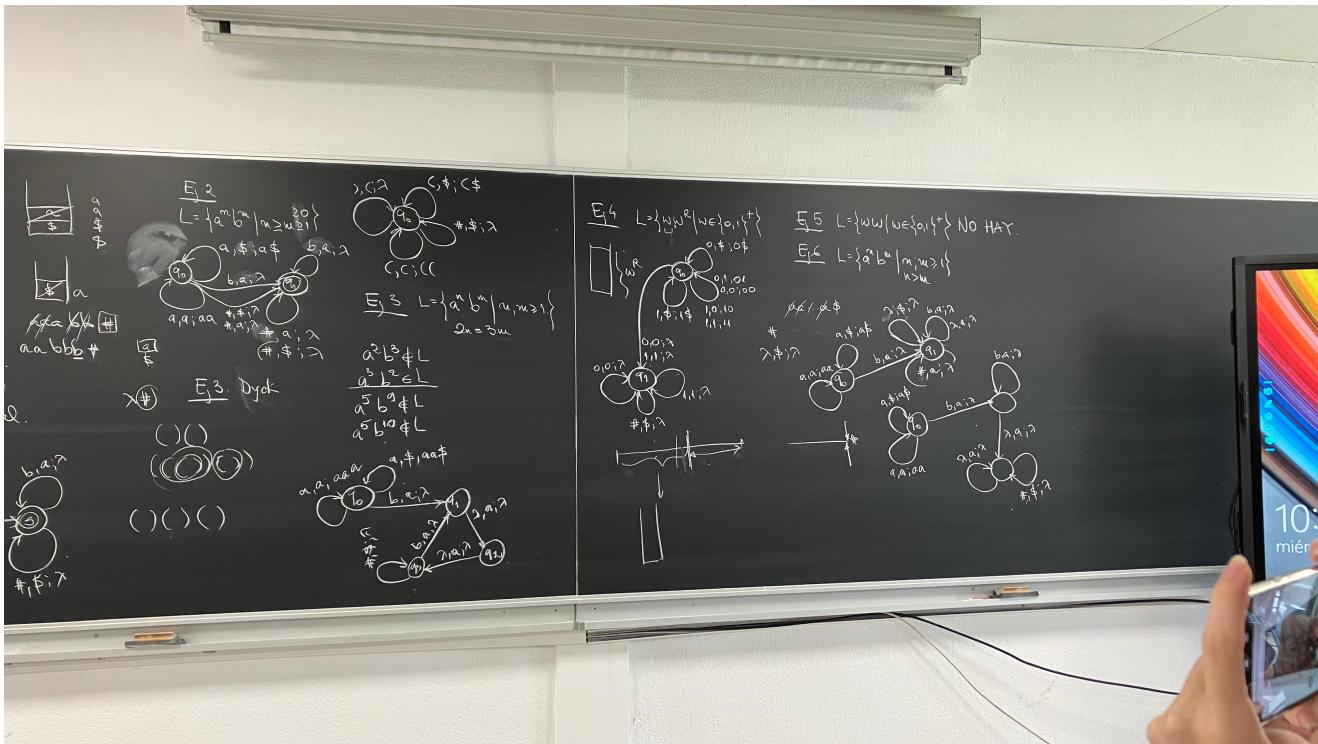
1. Recibe una palabra para que sea consumida y consume la primera entrada.
2. Busca transiciones en el estado actual que esperan esa entrada.
3. De las transiciones encontradas, prueba a consumir su símbolo definido. Si puede consumir:
 1. Se realiza la transición.
 2. Se elimina de la pila el símbolo definido.
 3. Se añade a la pila el símbolo definido.
 4. Si puede avanzar la entrada, pasa a 2.
 5. Pasa a **Aceptación**.
4. Pasa a **Aceptación**.

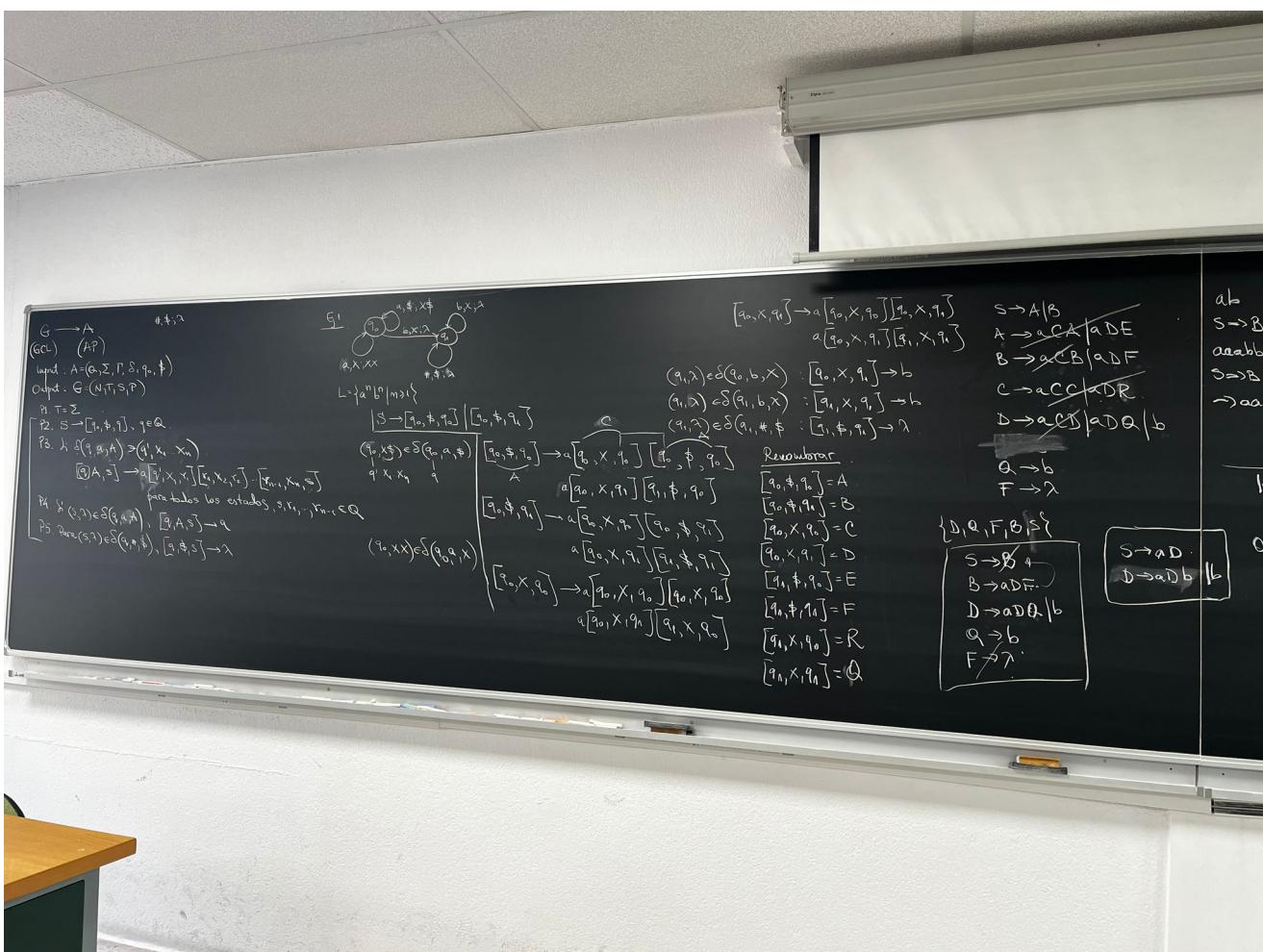
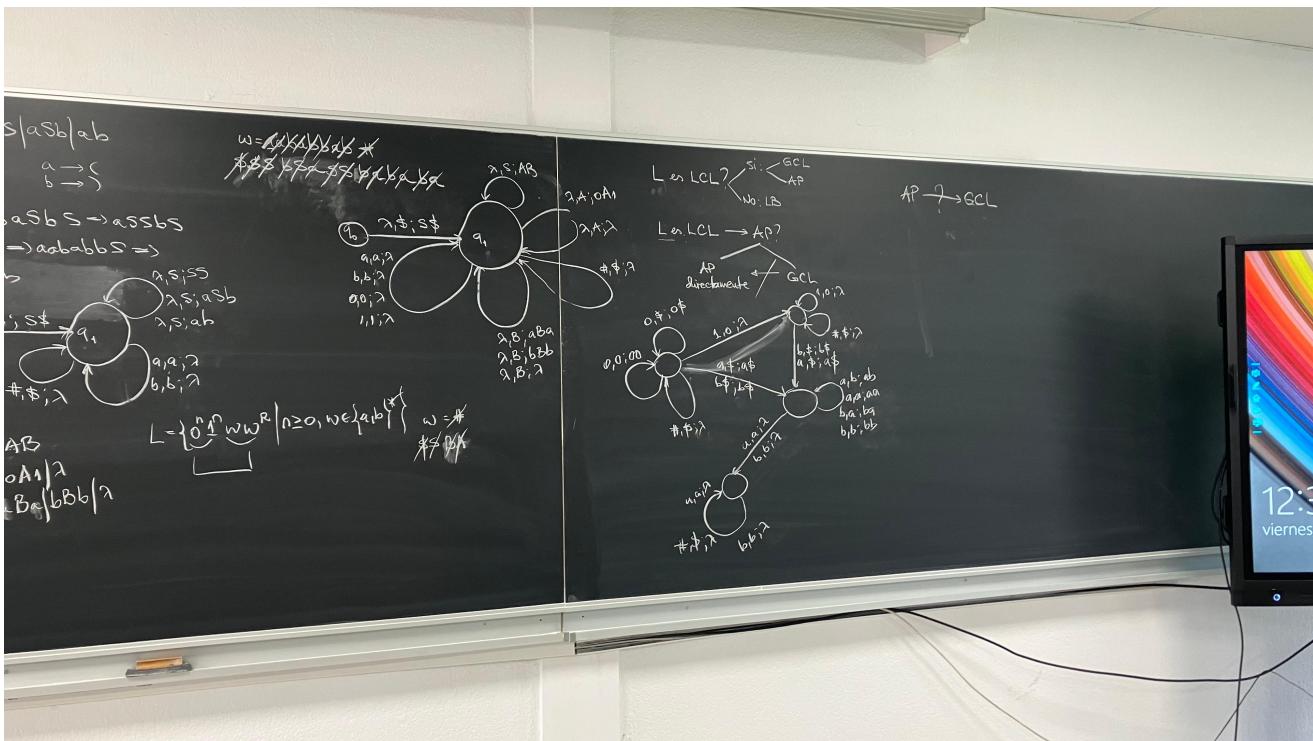
11.10.4 Aceptación

Un Pushdown Automaton acepta palabras - w - cuando:

- Al consumir totalmente w , tiene la **pila vacía**.
- Al consumir totalmente w , está en un **estado final**.

[POR HACER]





ab

$$S \rightarrow B \Rightarrow aDF \rightarrow abF \Rightarrow ab$$

$$aabbbb$$

$$S \rightarrow B \Rightarrow aDF \rightarrow aaDQF$$

$$\rightarrow aaaDQQF$$

$$\downarrow \downarrow \downarrow \downarrow$$

$$b b b \lambda$$

Input: una GCL, G
una palabra, w .

Output: $w \in L(G)$?

- Si $w \in L(G)$, una derivación por la izquierda para w .
- algoritmo eficiente $O(n^2)$!

$$A \rightarrow ab \mid abab \quad A \rightarrow abX$$

$$X \rightarrow ab \mid \lambda$$

G tiene prefijo comunes $\rightarrow G$ es ambiguia

$w \rightarrow$ el código en un LP
 $G \rightarrow$ la gramática que genera LP

1) Es w correcto?
2) $w \rightsquigarrow$ $\begin{cases} S \\ w \end{cases}$ \rightarrow Analizador sintáctico

Gramática no ambigua:
cada palabra es el lenguaje generado tiene una UNICA derivación por la izquierda.

Prefijos comunes:

$$A \rightarrow \alpha p_1 \mid \alpha p_2$$

$$S \rightarrow aY$$

$$Y \rightarrow bX \mid \lambda$$

Analizador semántico. Eliminar los prefijos comunes

Generar el código (intermedio) machine

$$\begin{array}{l} \rightarrow \lambda \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_m \mid \lambda \\ \rightarrow A \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n \mid \alpha X \\ \rightarrow X \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n \end{array}$$

$$S \rightarrow abX \mid as$$

$$X \rightarrow AaBC$$

$$A \rightarrow bAE$$

$$E \rightarrow bH \mid C$$

$$B \rightarrow \lambda$$

$$C \rightarrow abc \mid acb$$

$$D \rightarrow bac \mid cb$$

$$C \rightarrow aD$$

$$D \rightarrow bac \mid cb$$

10: miércoles

