Class Notes

CLASS NOTES

Contenidos de la Pizarra + Diapositivas + Apuntes.
Formato pdf *(Última Actualización).*

> **Note**
>
> *Apuntes bien hechos a partir del temario del segundo examen.*

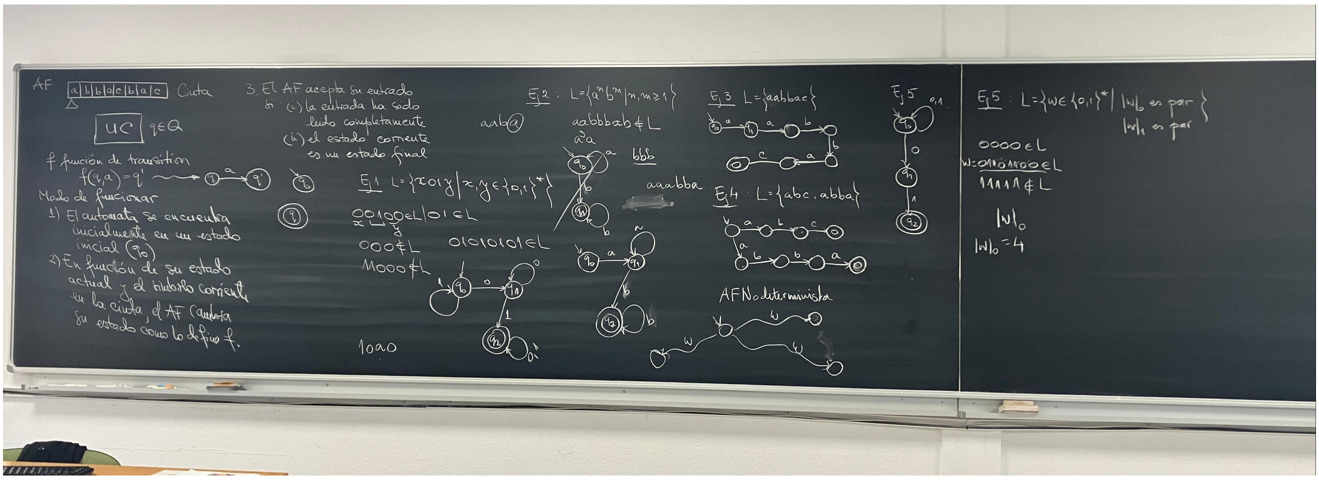
INDEX GITHUB

> **Tip**
>
> *Ciertas marcaciones se verán mal o distintas fuera de Obsidian.*
>
> *Recomendable descargar **Obsidian** y abrir este archivo haciendo un **clon** del repositorio.*

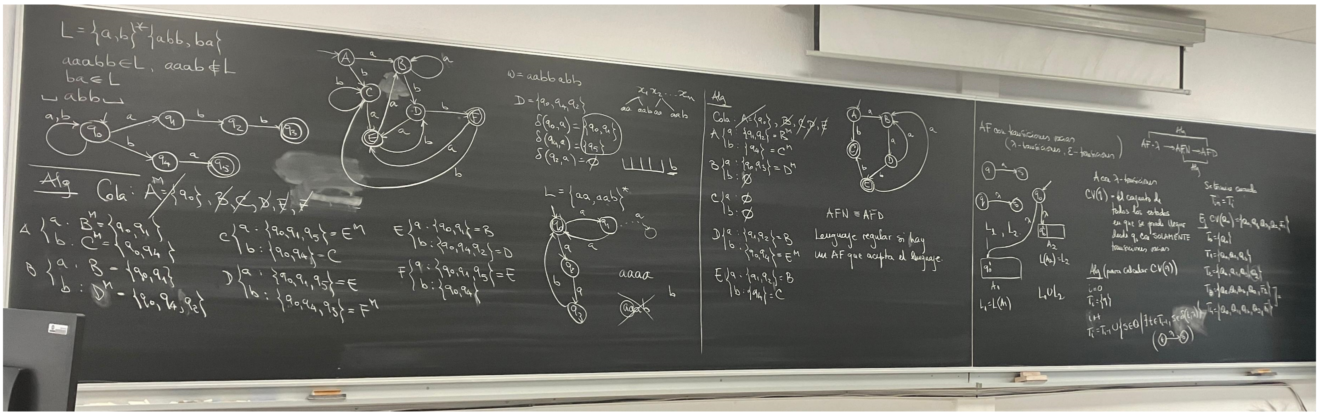
INDEX OBSIDIAN

NOTES

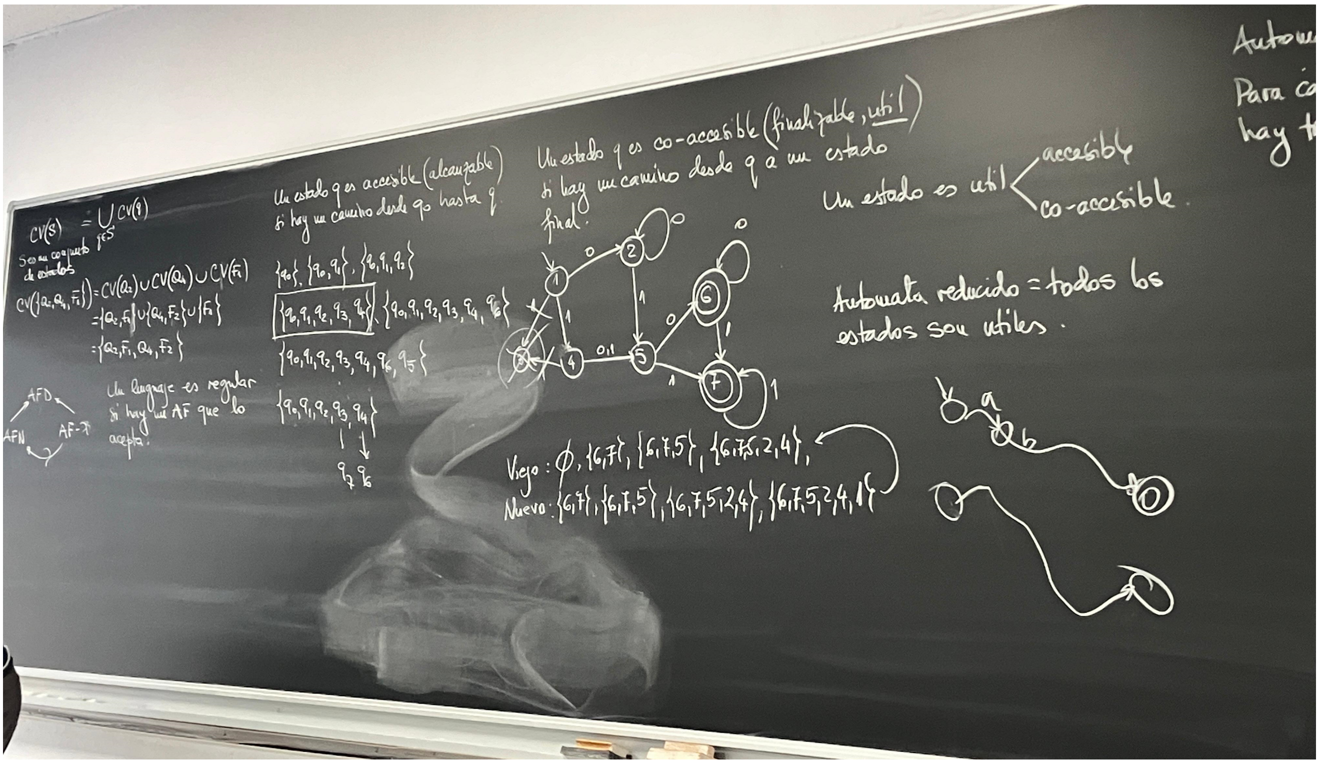
1. Basic Concepts



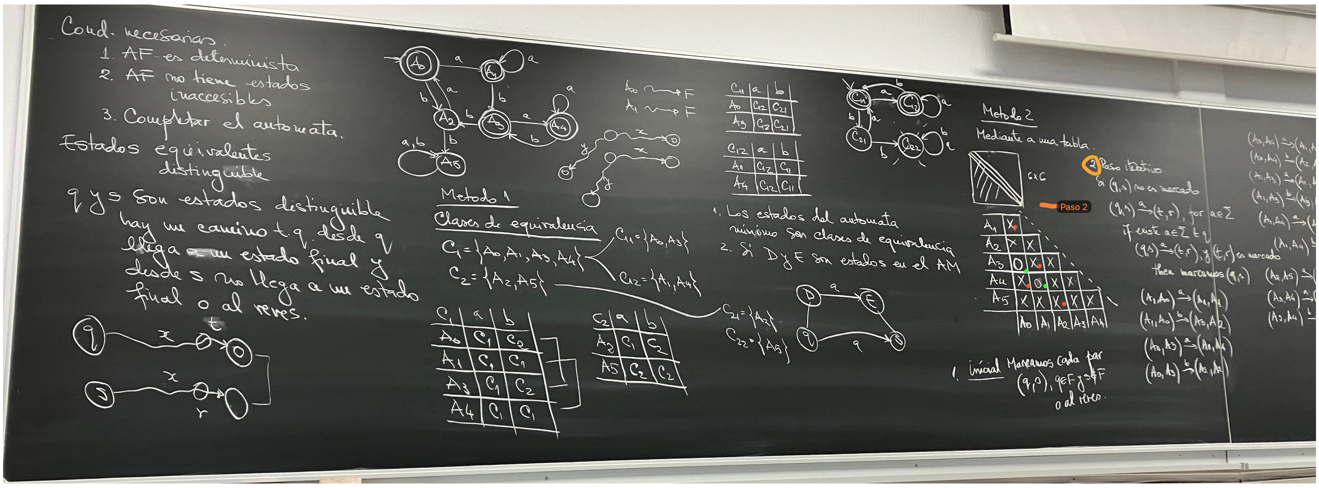
2. NFA to DFA

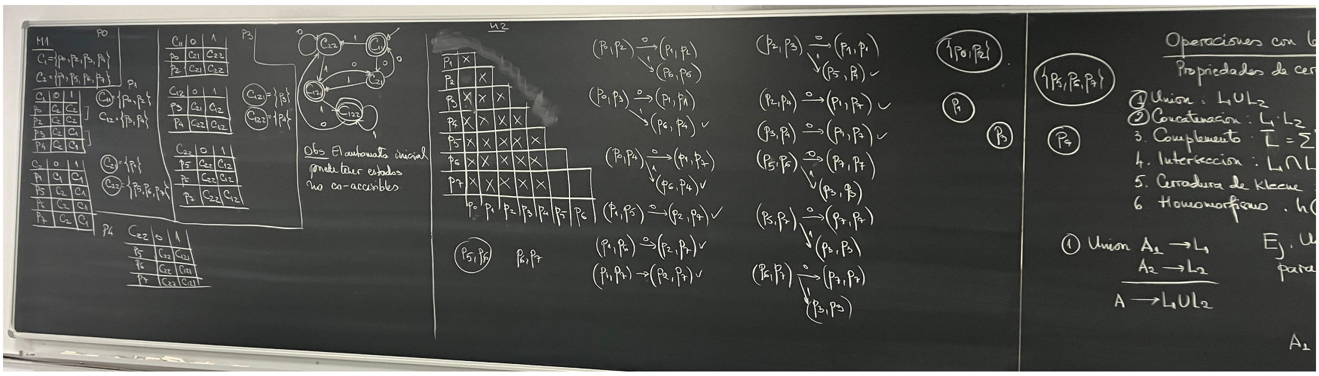


3. Epsilon Closures, Accessibility and Co-Accessibility



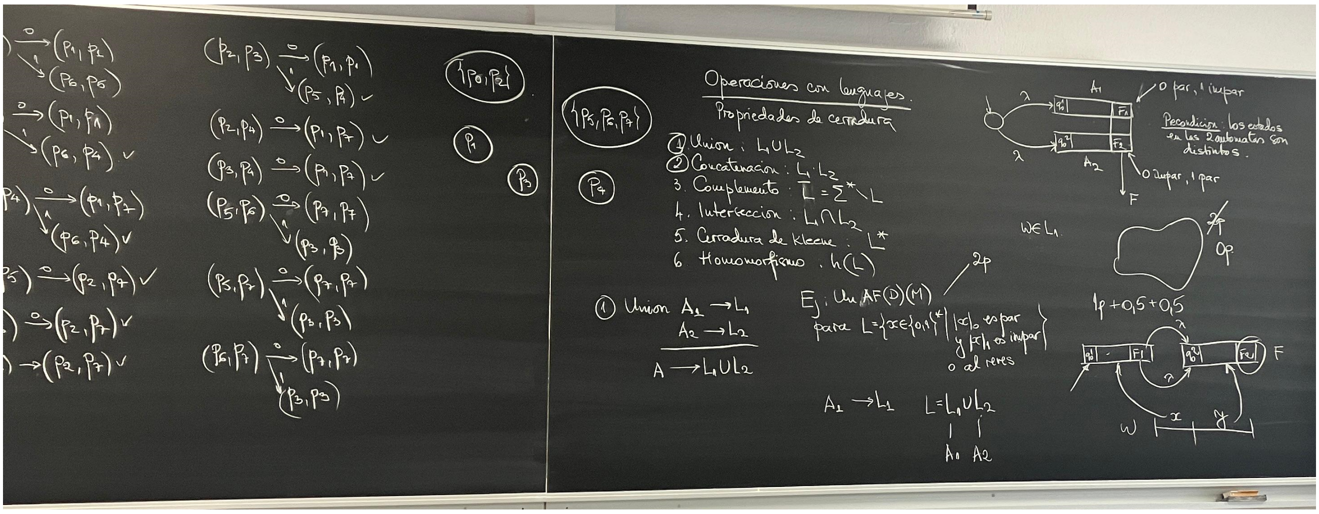
4. Minimization

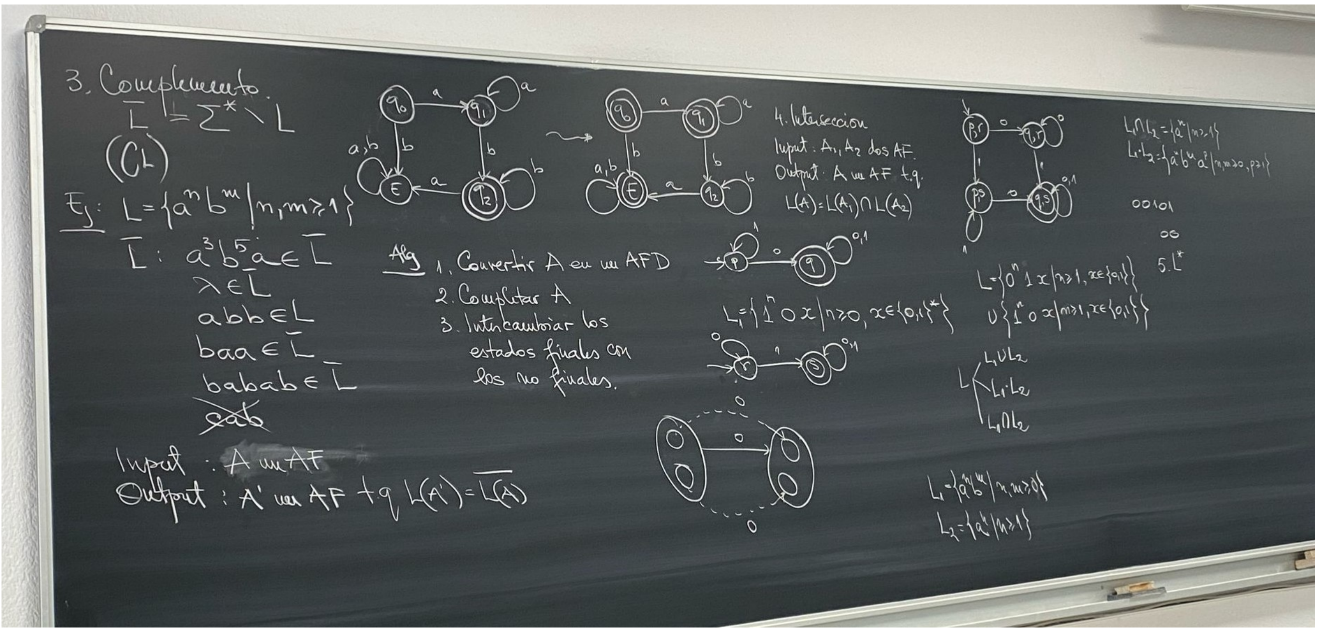


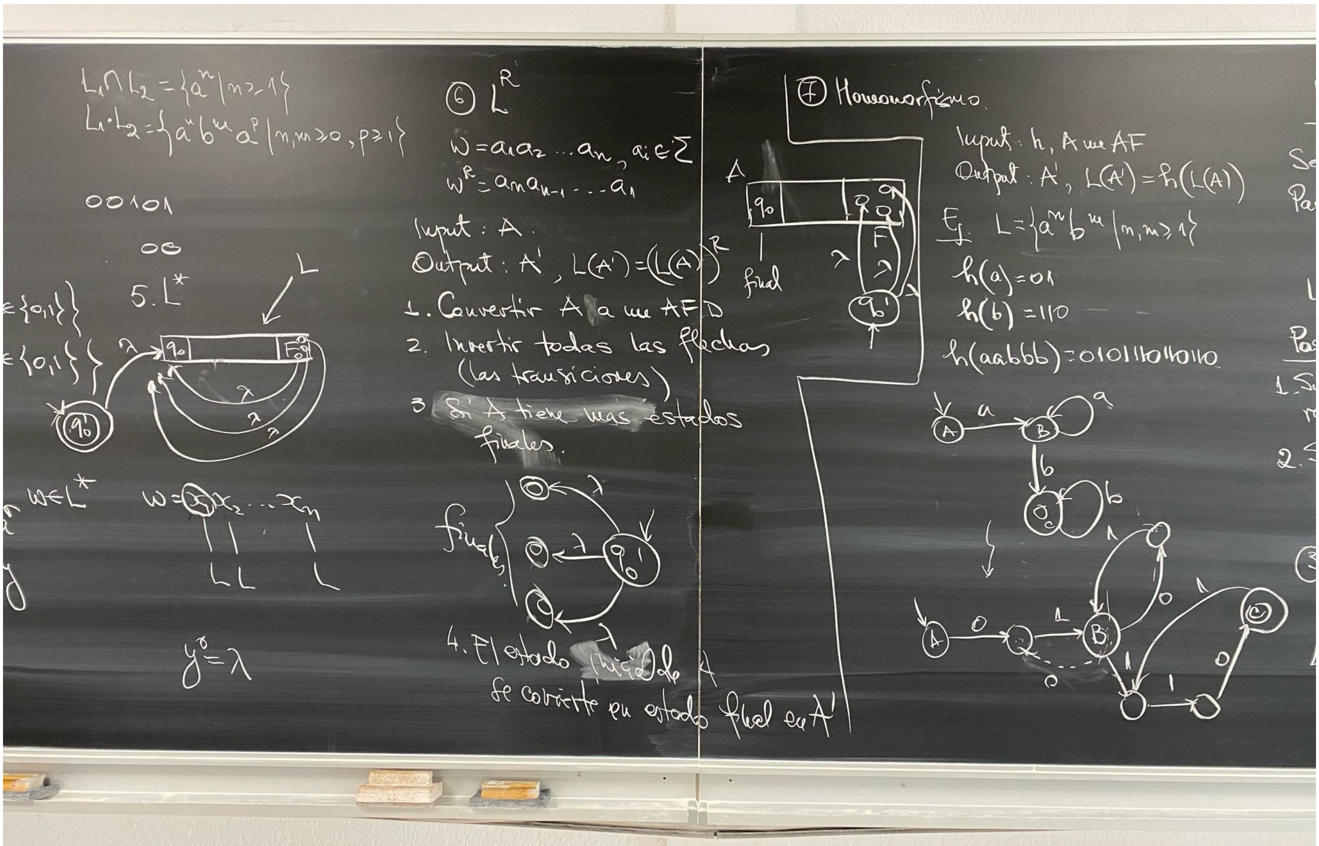


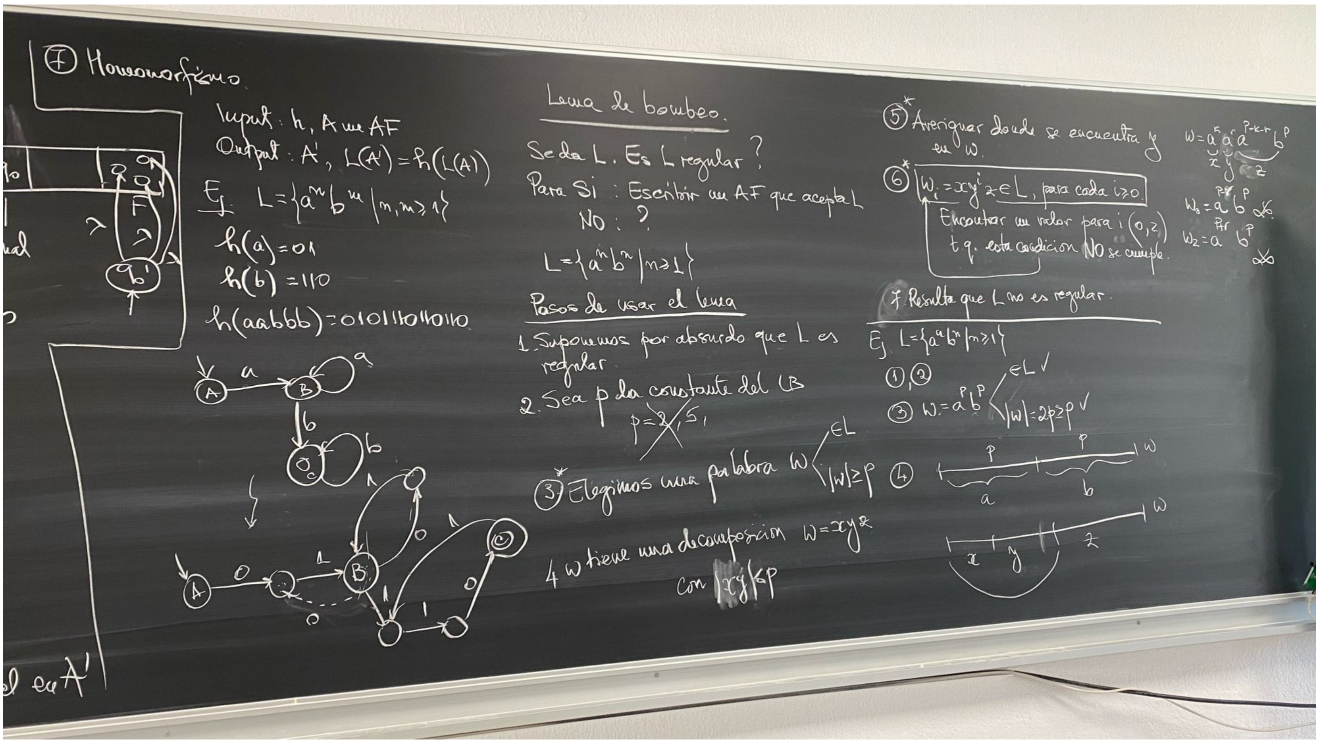
▶ **Notes**

5. Properties and Operations

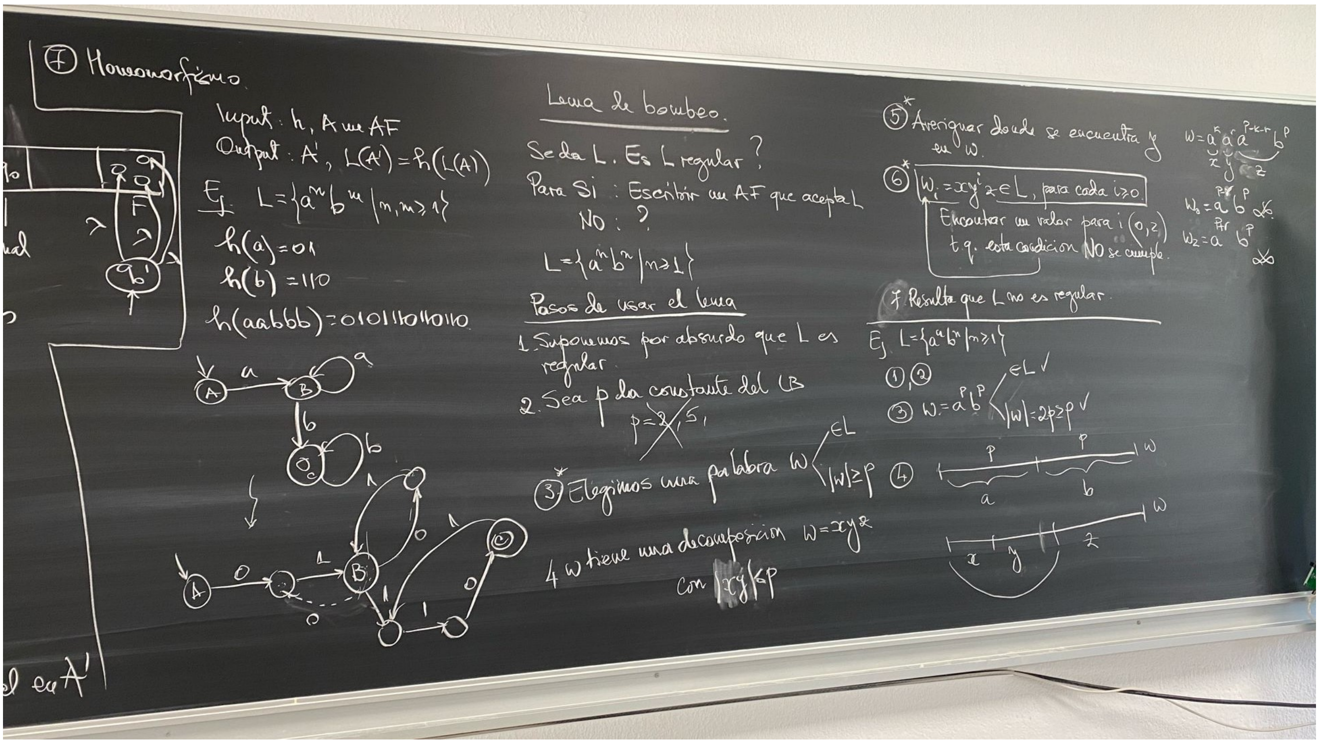

Operaciones con lenguajes.
Propiedades de cerradura
1 Unión : L1 U L2
2 Concatenación : L1 · L2
3. Complemento : L̄ = Σ* \ L
4. Intersección : L1 ∩ L2
5. Cerradura de Kleene : L*
6. Homomorfismo . h(L)
1 Unión A1 → L1
A2 → L2
A → L1 U L2
Ej: Un AF(D)(M)
para L = {x∈{0,1}* | |x|0 es par y |x|1 es impar} o al revés
A1 → L1
L = L1 U L2
A1 A2
Precondición: los estados en los 2 autómatas son distintos.
0 par, 1 impar
0 impar, 1 par
w ∈ L1
1p + 0,5 + 0,5

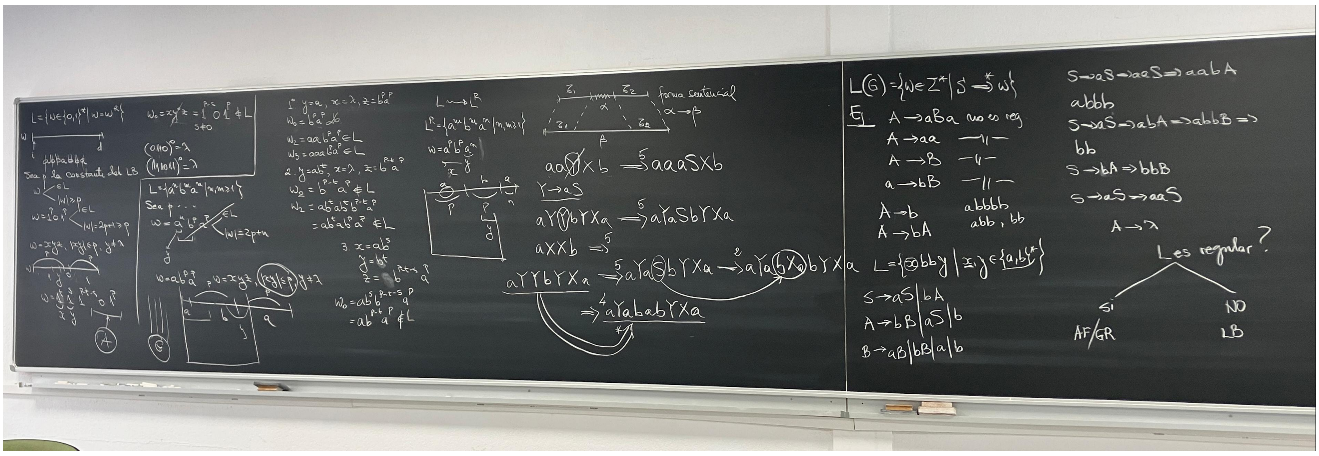

3. Complemento.
L̄ = Σ* \ L
(CL)
Ej: L = {a^n b^m | n,m ≥ 1}
L̄ : a³b⁵a ∈ L̄
λ ∈ L̄
abb ∈ L
baa ∈ L̄
babab ∈ L̄
Input : A un AF
Output : A' un AF tq L(A') = L(A)‾
Alg
1. Convertir A en un AFD
2. Completar A
3. Intercambiar los estados finales con los no finales.
4. Intersección
Input : A1, A2 dos AF.
Output : A un AF tq.
L(A) = L(A1) ∩ L(A2)
00101
00
5. L*


L1 ∩ L2 = {a^n | n ≥ 1}
00101
00
5. L*
y⁰ = λ
6 L^R
w = a1a2 ... an , ai ∈ Σ
w^R = an an-1 ... a1
Input : A.
Output : A' , L(A') = (L(A))^R
1. Convertir A a un AFD
2. Invertir todas las flechas (las transiciones)
3. Si A tiene mas estados finales.
4. El estado inicial de A se convierte en estado final en A'
7 Homomorfismo
Input : h, A un AF
Output : A', L(A') = h(L(A))
Ej. L = {a^n b^m | n,m ≥ 1}
h(a) = 01
h(b) = 110
h(aabbb) = 0101110110110


Homomorfismo.
Input: h, A un AF
Output: A', L(A') = h(L(A))
Ej. L={a^n b^m | n,m ≥ 1}
h(a)=01
h(b)=110
h(aabbb)=0101110110110
en A'
Lema de bombeo.
Sea L. Es L regular?
Para Si: Escribir un AF que acepta L
No: ?
L={a^n b^n | n≥1}
Pasos de usar el lema
1. Suponemos por absurdo que L es regular.
2. Sea p la constante del LB
3. Elegimos una palabra w ∈L |w|≥p
4 w tiene una descomposición w=xyz con |xy|≤p
5 Averiguar donde se encuentra y en w.
6 w_i = xy^i z ∈ L, para cada i≥0.
Encontrar un valor para i (0,2) t.q. esta condición NO se cumple.
7 Resulta que L no es regular.
Ej L={a^n b^n | n≥1}
1, 2
3 w=a^p b^p ∈L ✓ |w|=2p≥p ✓

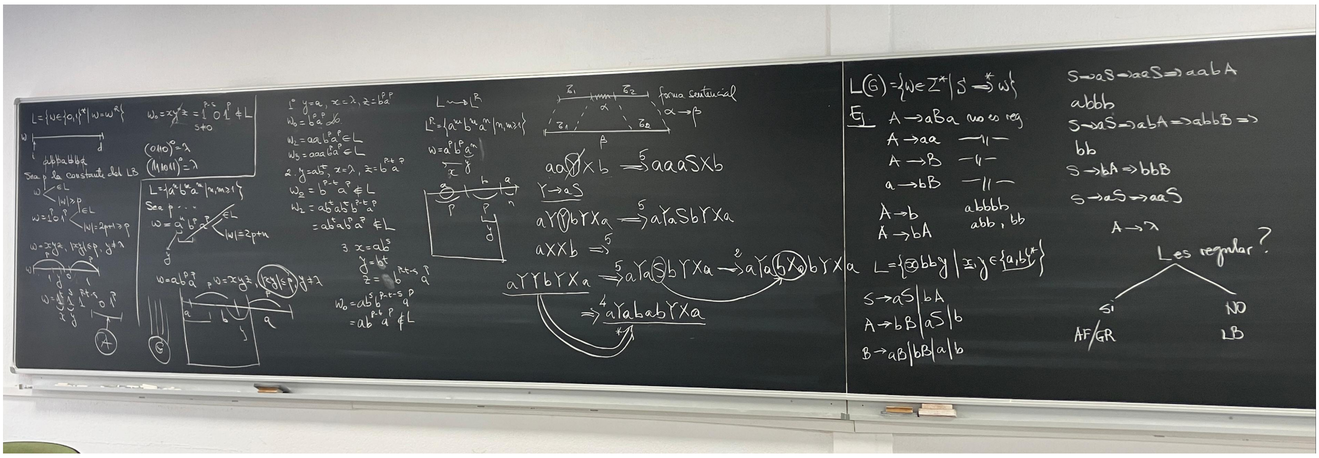
6. Pumping


Homomorfismo.
Input: h, A un AF
Output: A', L(A') = h(L(A))
Ej. L={a^n b^m | n,m ≥ 1}
h(a)=01
h(b)=110
h(aabbb)=0101110110110
en A'
Lema de bombeo.
Sea L. Es L regular?
Para Si: Escribir un AF que acepta L
No: ?
L={a^n b^n | n≥1}
Pasos de usar el lema
1. Suponemos por absurdo que L es regular.
2. Sea p la constante del LB
3. Elegimos una palabra w ∈L |w|≥p
4 w tiene una descomposición w=xyz con |xy|≤p
5 Averiguar donde se encuentra y en w.
6 w_i = xy^i z ∈ L, para cada i≥0.
Encontrar un valor para i (0,2) t.q. esta condición NO se cumple.
7 Resulta que L no es regular.
Ej L={a^n b^n | n≥1}
1, 2
3 w=a^p b^p ∈L ✓ |w|=2p≥p ✓



▶ **Apuntes**

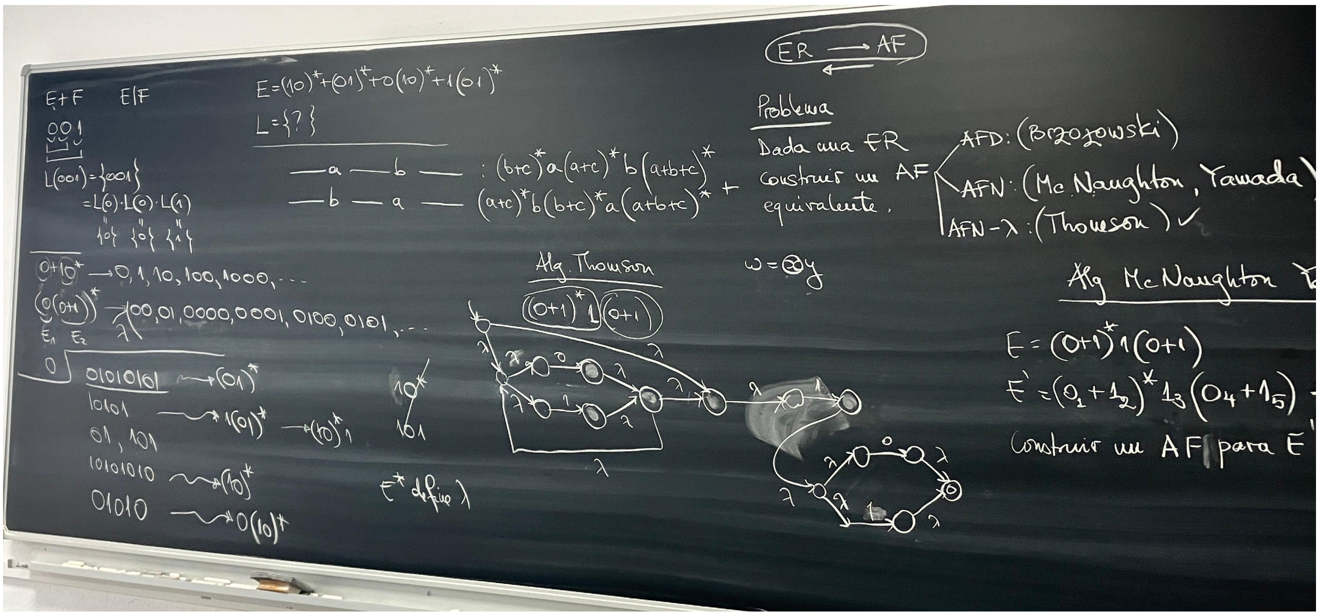
7. Grammar



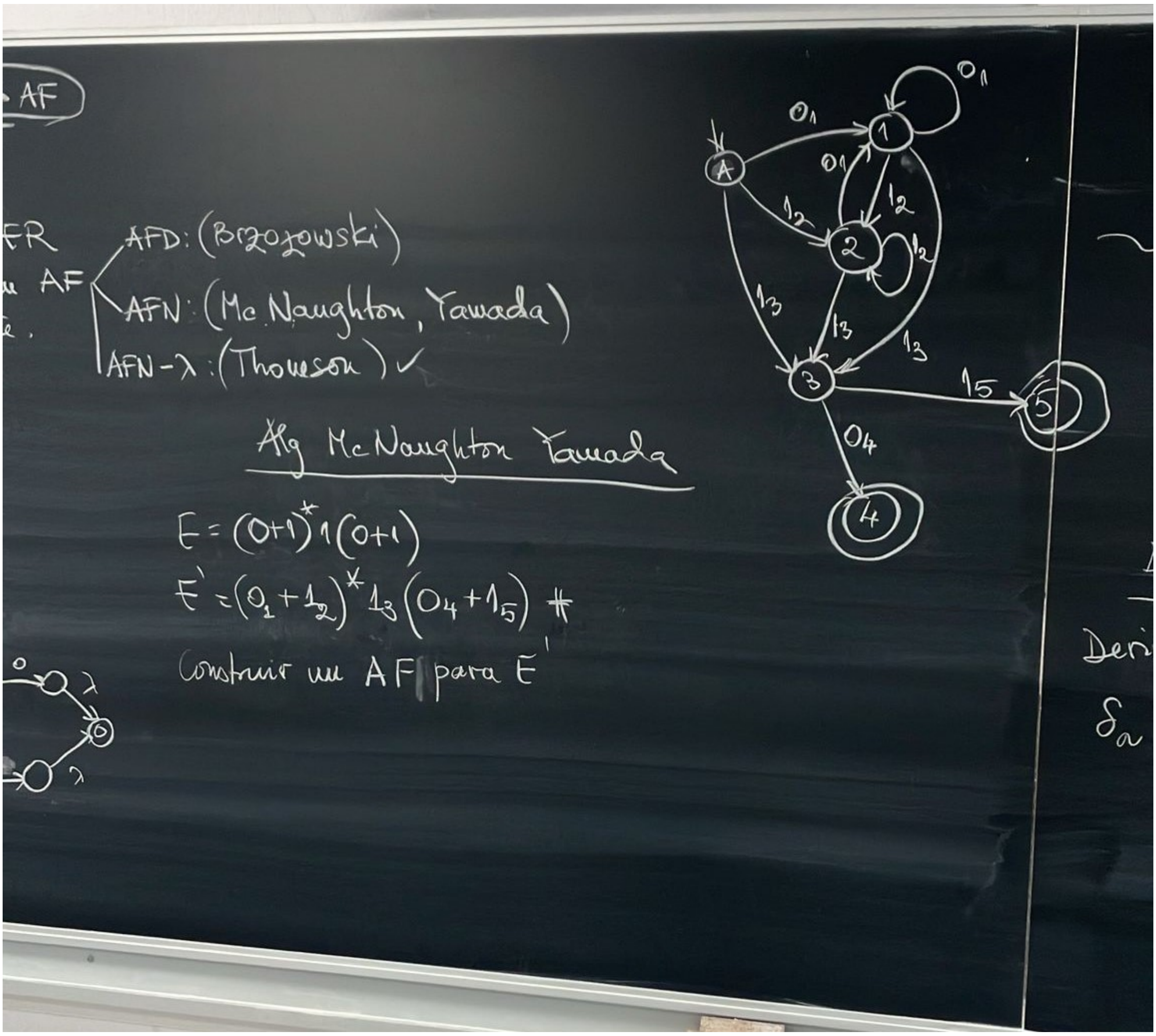
▶ **Apuntes**

8. ER a Automata

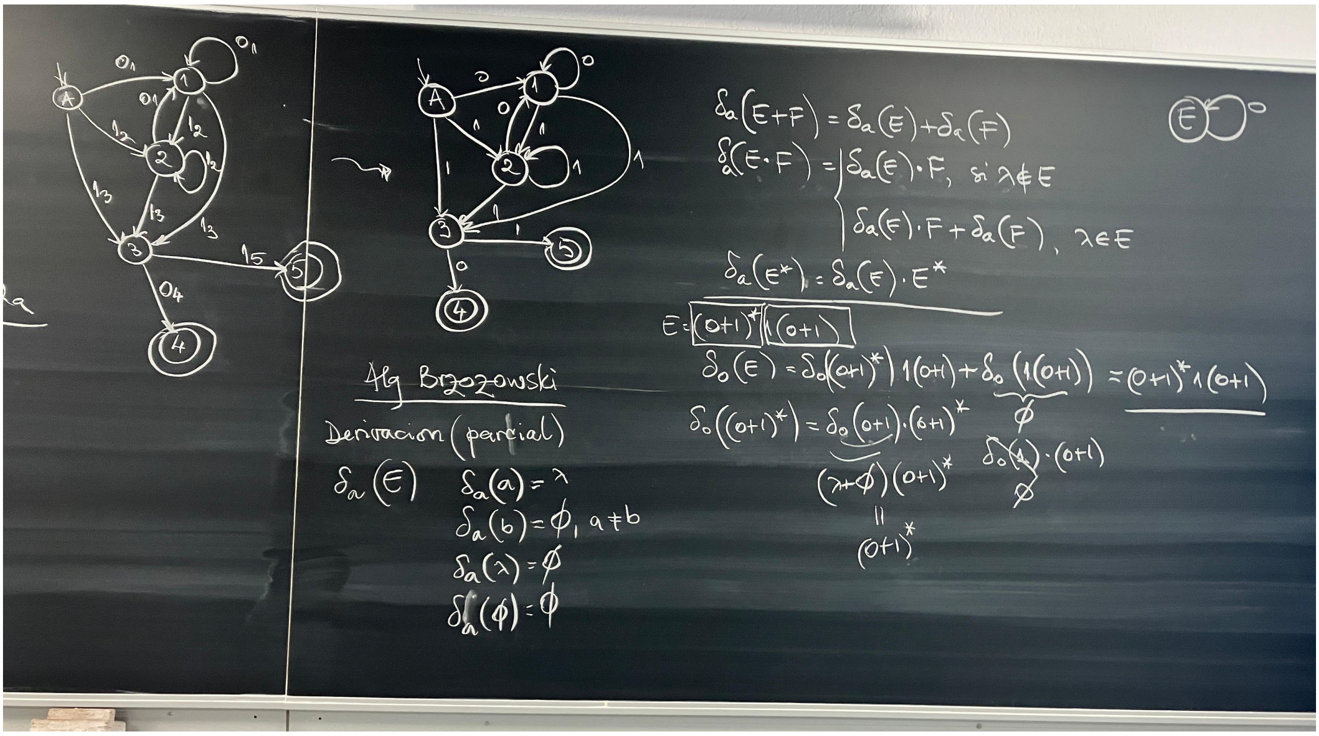
8.1. Thomson (NFA-ε)



8.2. M'Naghten Yamada (NFA)



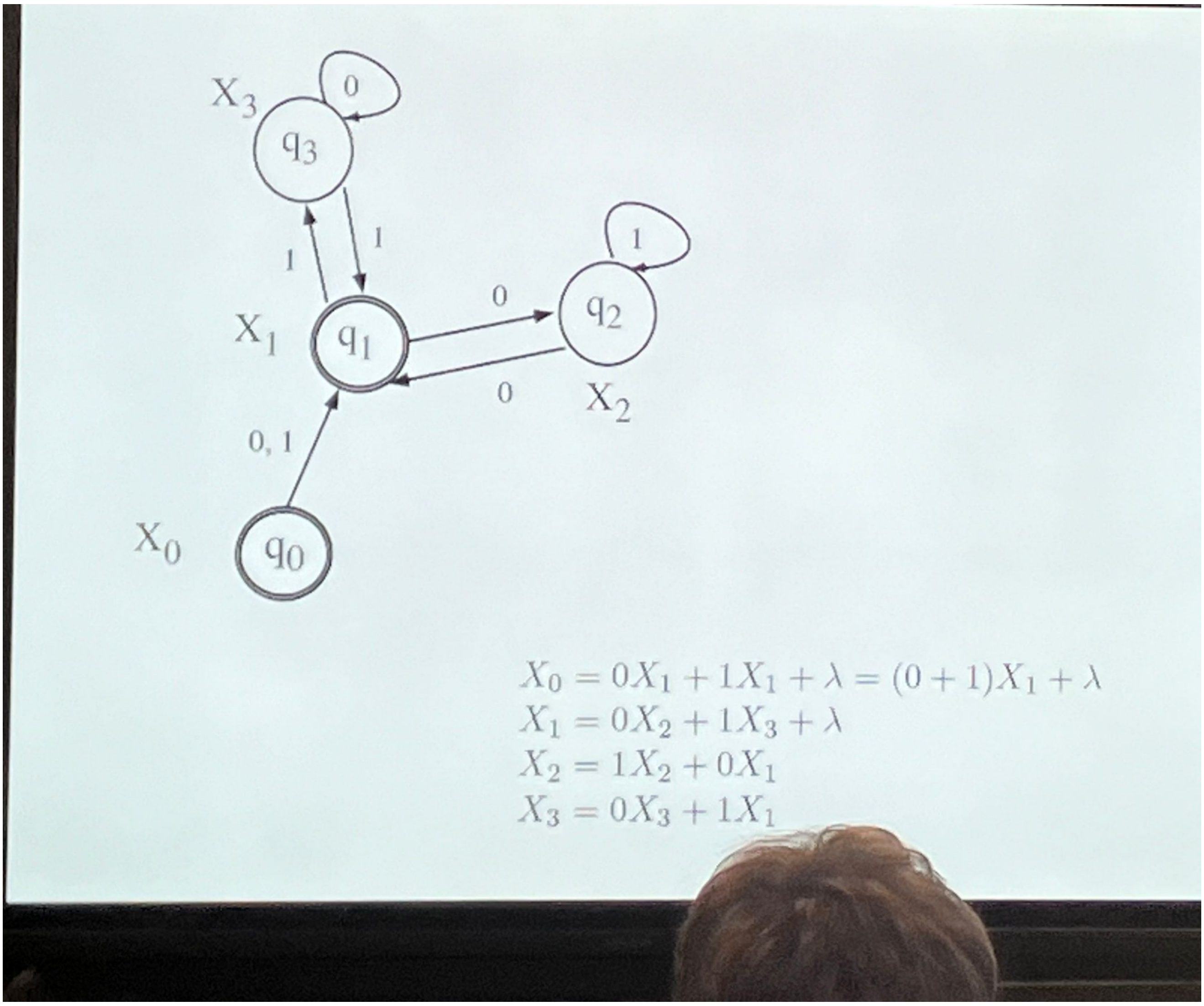
8.3 Brzozowski (DFA)



▶ **Apuntes**

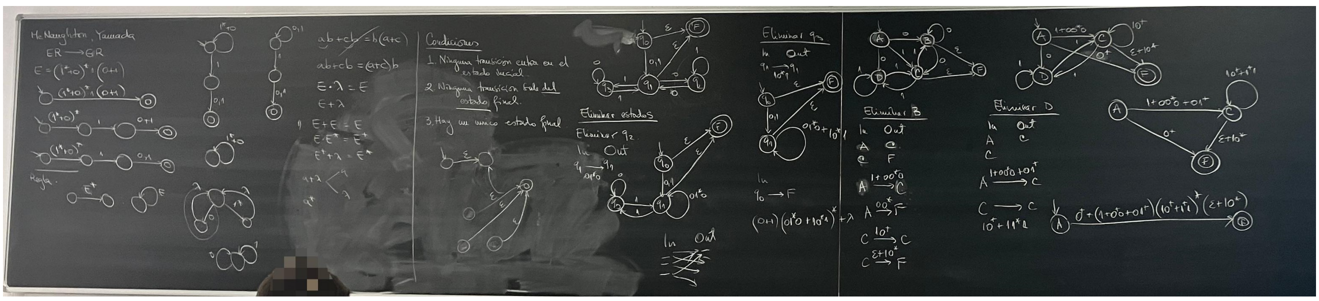
9. Automata a ER

9.1. Mediante Sistema de Ecuaciones



▶ **Apuntes**

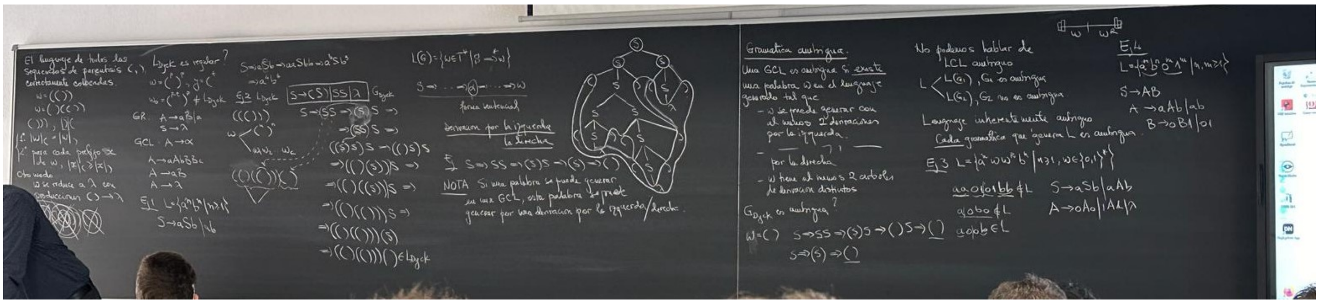
9.2. Mediante Eliminación de Estados



▶ **Apuntes**

10. ER a Gramatica POR HACER

11. Gramáticas de Contexto Libre - *GCL*



11.1. Árbol de Derivación

Representación gráfica del proceso de derivación de una gramática.

- **Raiz**: 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.

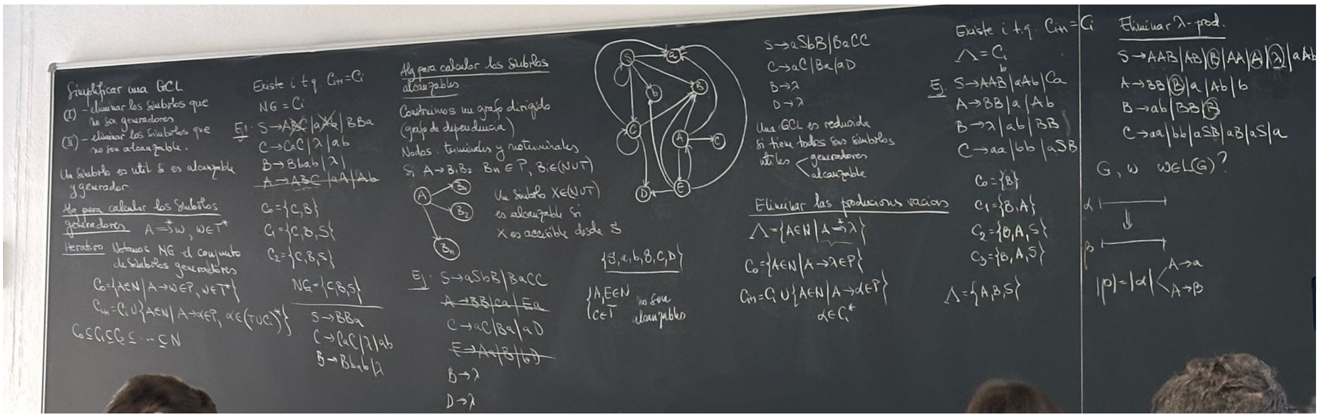
11.2.1. Lenguajes inherentemente ambiguos

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.

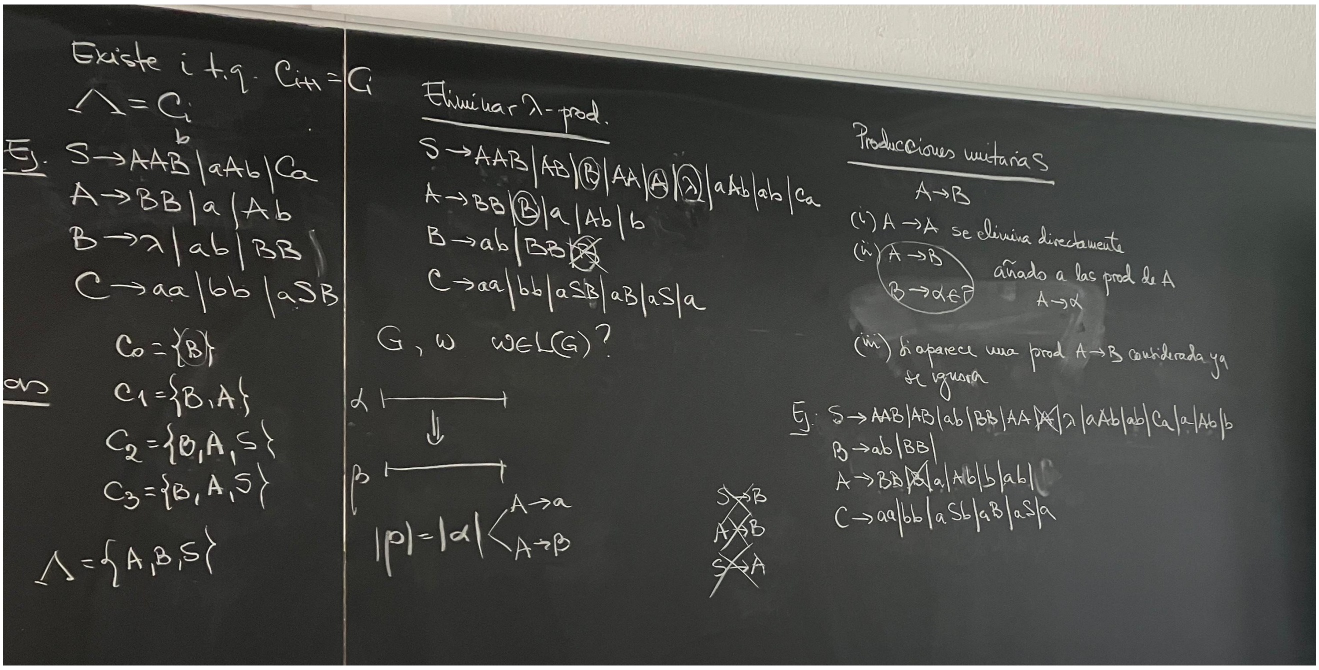
11.3.5 GCL Reducida

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. $B \to \lambda$	**Prod.** $A \to \lambda$	**Prod.** $S \to \lambda$	**Prod** $S \to \lambda$
$B \to BB$	B		
$S \to AAB$	AA	$A\|\lambda$	
$A \to BB$	$B\|\lambda$		
$C \to aSB$	aS		a
	$S \to aAb$	ab	
	$A \to Ab$	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 $A \to B$.

- $A \to A$: Borramos.

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

> *Forma guay*: $\forall A \to B : B \to \alpha_n$, pasar a: $A \to \alpha_n$

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:

- $A \to BC$
- $A \to a$

Para ello tenemos que:

1. **Limpiar la Gramática**
   - **Eliminar Prefijos Comunes (Eliminar Ambiguedad)**
   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**.

> **Tip**
>
> ***Video Recomendado - Teoría***
>
> ***Video Recomendado - Ejemplos***

11.5.1 Eliminar Ambiguedad de GCL

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

Prefijos comunes.

$A \to \alpha\beta \mid \alpha\gamma$.

Eliminar los prefijos comunes

$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \alpha\beta_n \mid \gamma_1 \mid \dots \mid \gamma_m$.

$A \to \gamma_1 \mid \dots \mid \gamma_m \mid \alpha X$

$X \to \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$.

Ej. $S \to abAaB \mid abC \mid aS$

✓ $A \to bAbA \mid bAC$

✓ $B \to \lambda$

✓ $C \to abc \mid acb$

$S \to abX \mid aS$

✓ $X \to AaB \mid C$

$S \to aY$

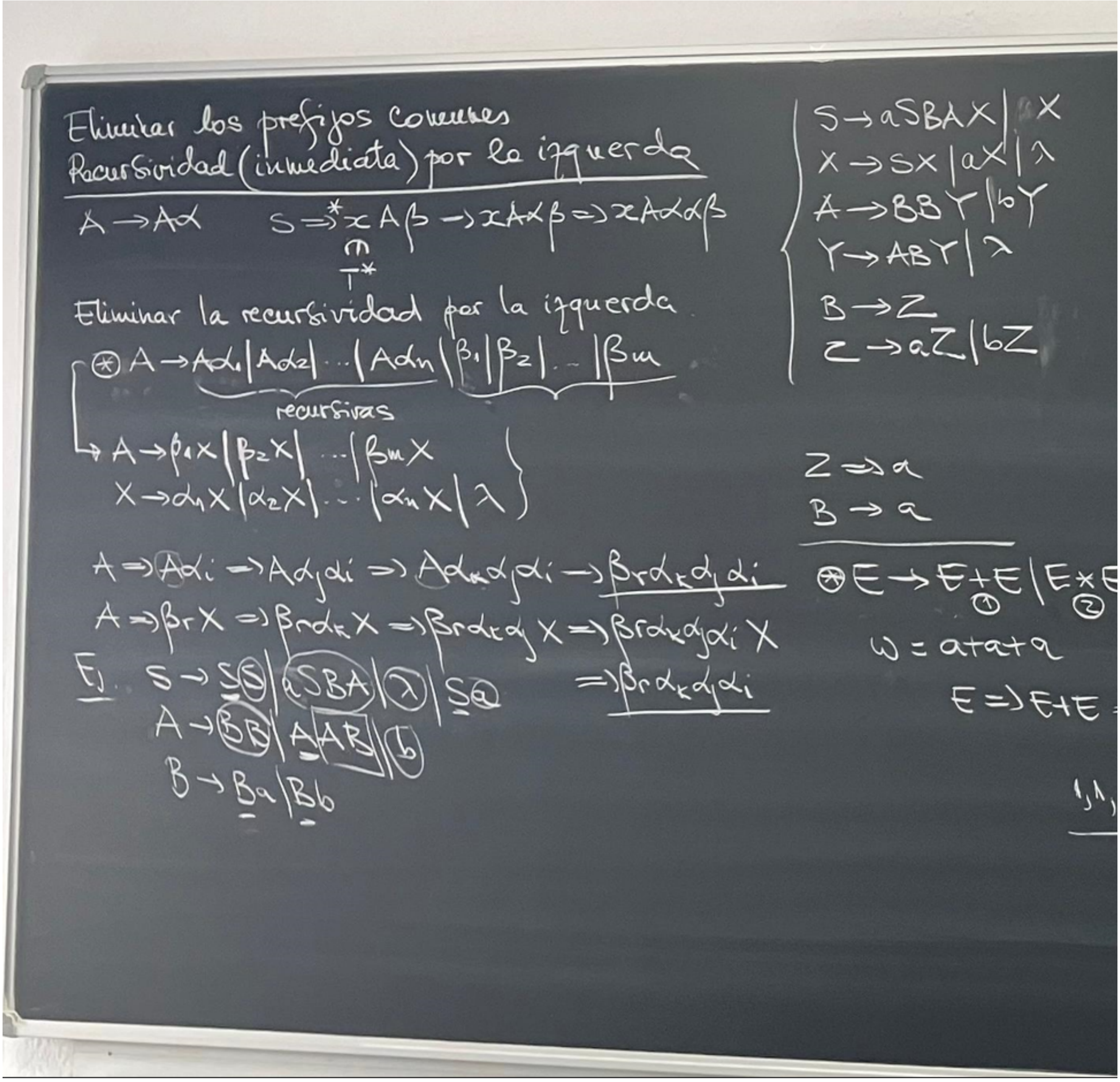
$Y \to bX \mid S$

$A \to bAE$

$E \to bA \mid C$

$C \to aD$

$D \to bc \mid cb$



Para pasar de una [Gramática Ambigua]() a una no ambigua, buscamos eliminar los [Prefijos Comunes]().

*Para **Prefijos Comunes Normales**:*

1. Sacar "**Factor Común**" del **Símbolo**.
2. Crear nuevo **No Terminal** con los implicados del original.

> **Note**
>
> *El paso 2 solo se realiza en producciones com más de un símbolo en cada implicado.*

*Para **Prefijos Comunes Recursivos Inmediatos por la Izquierda**:*

Input:

- $A \to A\alpha_1|\dots|A\alpha_n|\beta_1|\dots|\beta_n$

Output:

- $A \to \beta_1 B|\ldots|\beta_n B$
- $B \to \alpha_1 B|\ldots|\alpha_n B|\lambda$

Donde:

- A: Símbolo No terminal.
- B: Símbolo No terminal.
- α: Subpalabra que procede A.
- β: Subpalabra que no procede A.

> **Note**
>
> ***Mi razonamiento:***
> Sabemos que en una transición $A \to pAs|a|b$, p actúa como prefijos, s como sufijos y que el cuerpo de A podría ser o a o b.
>
> En el caso de **Recursividad Inmediata**, los prefijos no están presentes, quedando: $A \to As|a|b$. Es decir, el cuerpo de A estará compuesto por o a o b y luego sufijos s, podemos traducir esto a: $A \to aB|bB$, siendo B los sufijos s: $B \to sB|\lambda$.
>
> Ahora podemos generalizar para cualquier producción y sacar la fórmula dada. s será cualquier subpalabra que proceda A (α) y las subpalabras a y b serán cualquier subpalabra que no proceda A (β).

11.5.2 Prefijo Común

Dada una producción $A \to X|Y$, esta producción es ambigua si los símbolos iniciales de X e Y son iguales.

> **Note**
>
> *Ejemplo:*
> $A \to ab|a$
> $A \to Ab|Aa|a$ - *Recursividad Inmediata por la Izquierda!*

11.5.3 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.

> **Note**
>
> ***Ejemplo***:

- $A \to aAB|B$
- $B \to b$

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

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

11.5.4 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.

Note

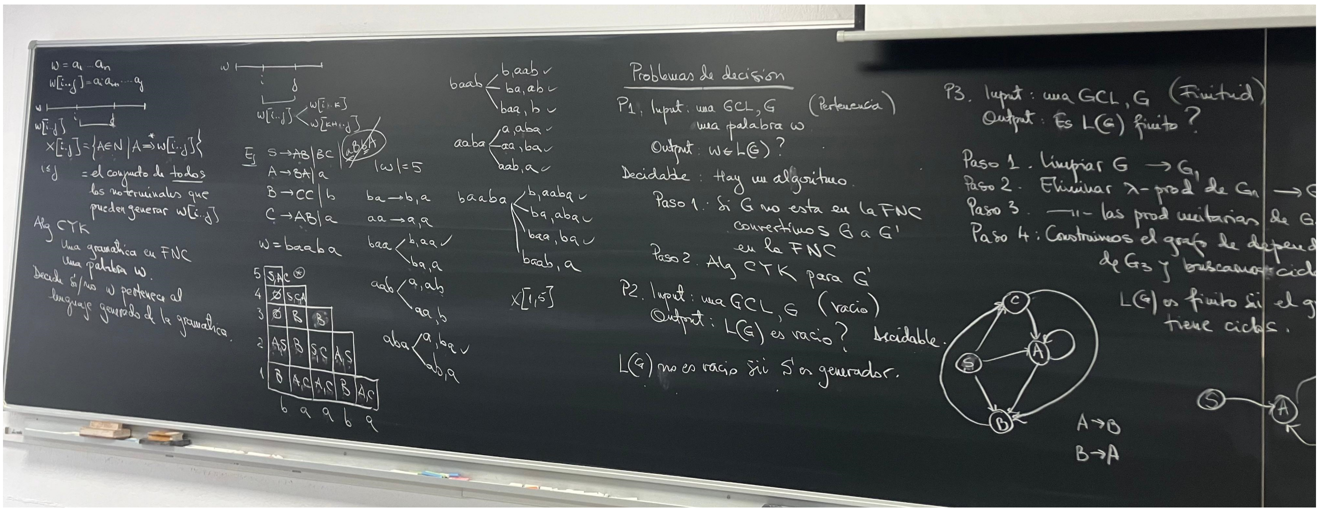
Ejemplo:

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

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

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

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.

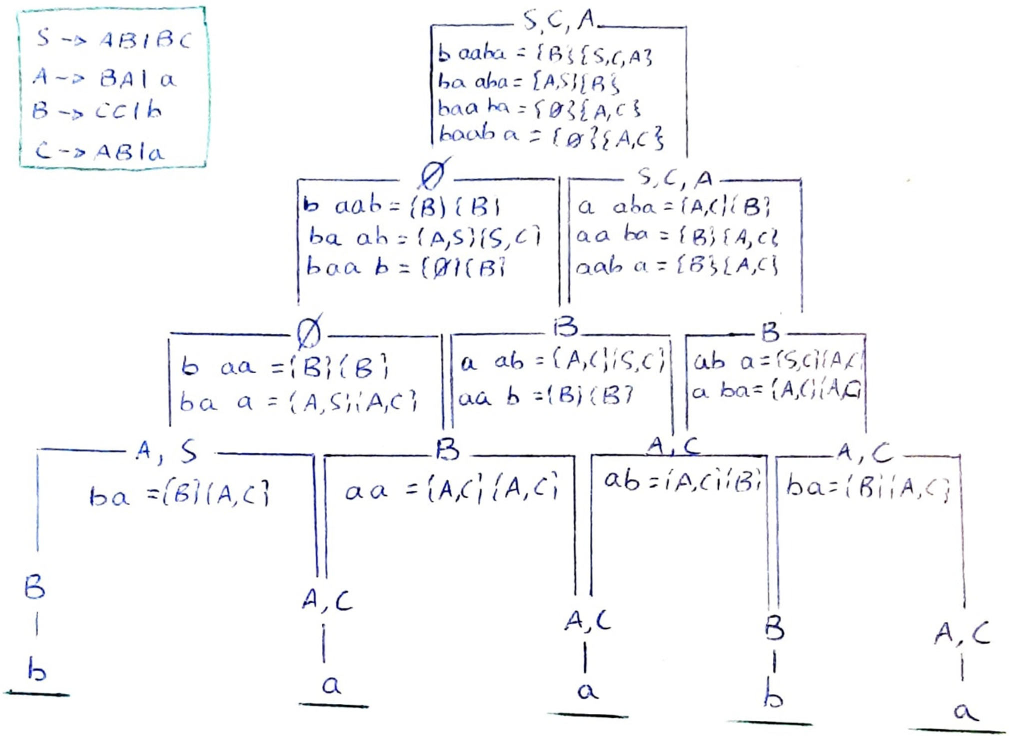
1. Construimos una matriz triangular inferior de $n \times x$.
2. En el eje X escribimos carácter por carácter, la palabra a probar (w).
3. Escribimos en la fila más baja el conjunto de símbolos que nos posibilita llegar directamente al carácter de abajo.
4. Escribimos en la fila arriba de la anterior,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

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

$w = baaba$



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 file 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.

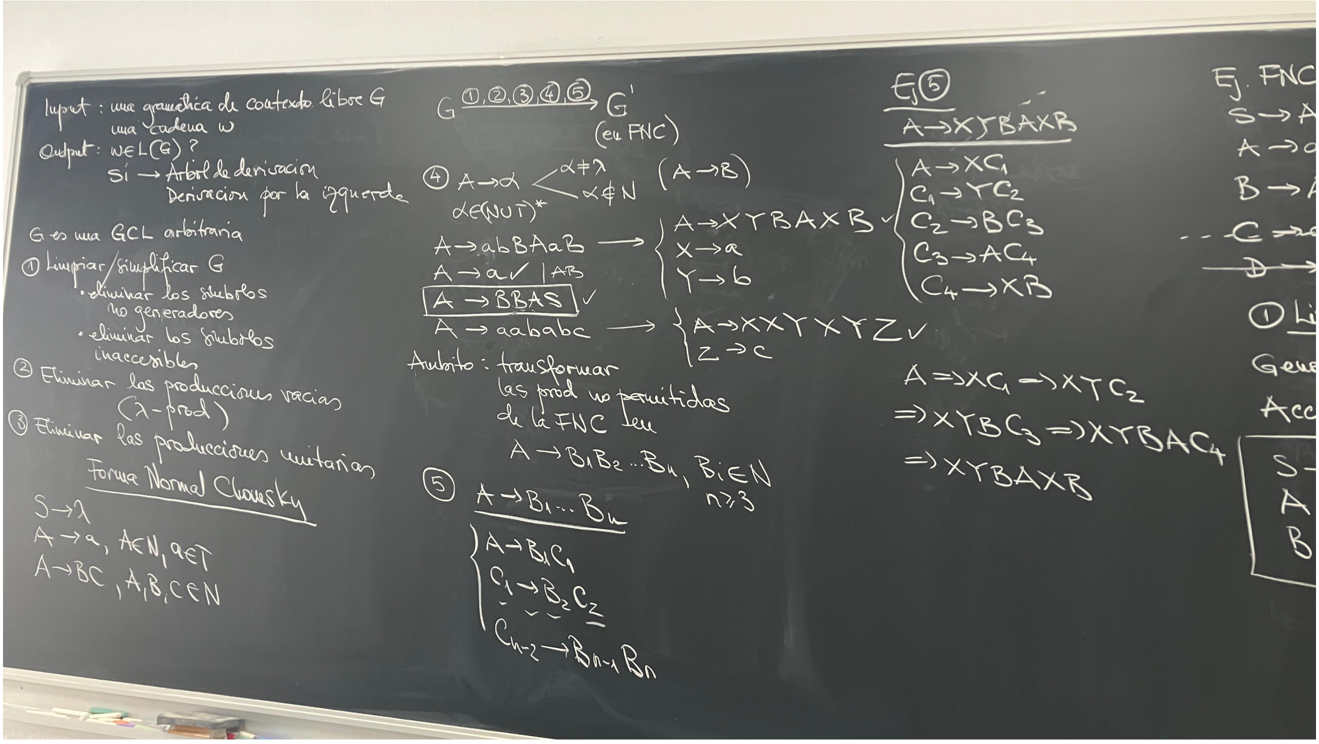
Tip

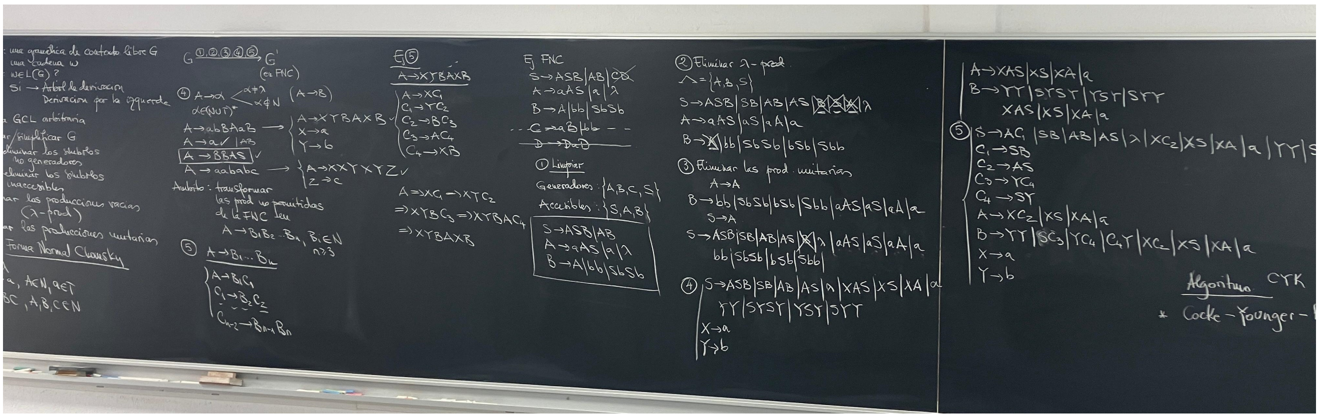
Video Recomendado - Teoría

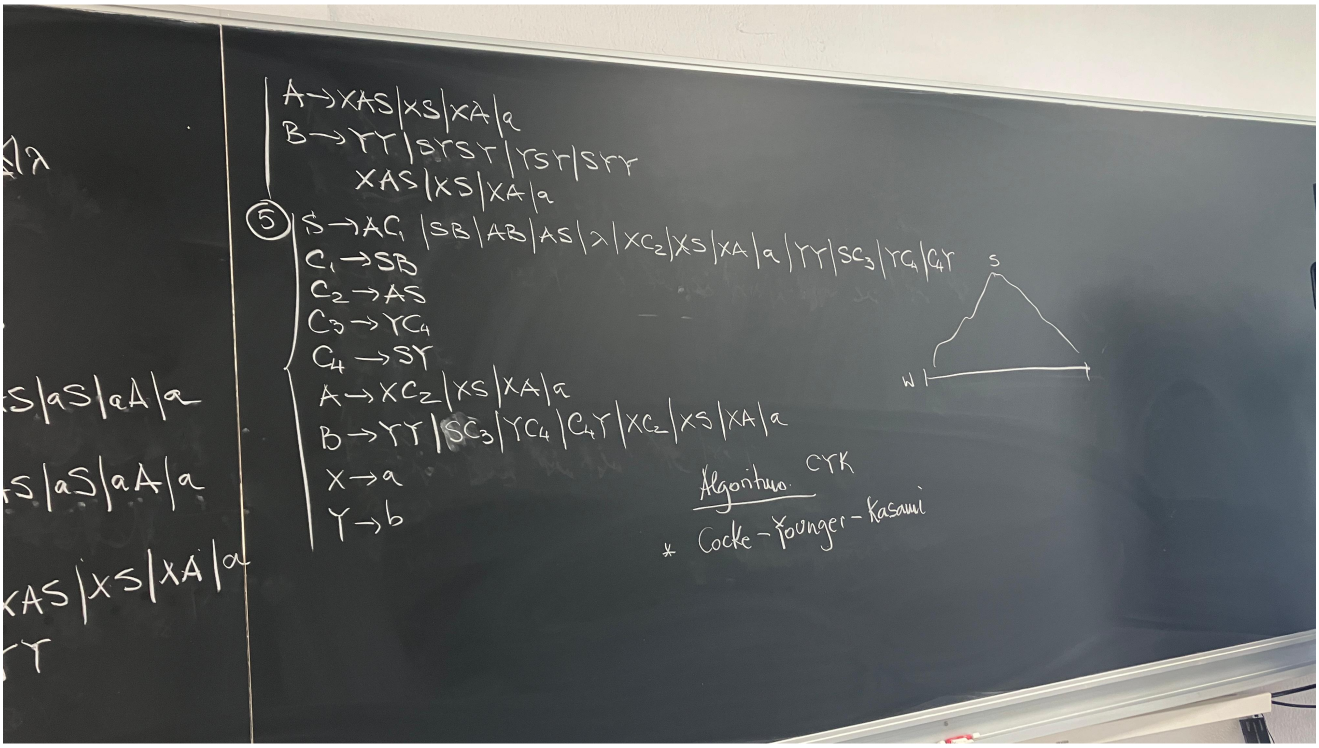
Video Recomendado - Ejemplos

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∈L(G) ?
sí → 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→λ
A→a, A∈N, a∈T
A→BC , A,B,C∈N
G ①,②,③,④,⑤ → G' (en FNC)
④ A→α
α≠λ
α∉N
α∈(N∪T)*
(A→B)
A→abBAaB
A→XYBAXB
X→a
Y→b
A→a
AB
A→BBAS
A→aababc
A→XXYXYZ
Z→c
Ámbito : transformar las prod no permitidas de la FNC len
A→B₁B₂…Bₙ, Bᵢ∈N
n≥3
⑤ A→B₁…Bₙ
A→B₁C₁
C₁→B₂C₂
Cₙ₋₂→Bₙ₋₁Bₙ
Ej ⑤
A→XYBAXB
A→XC₁
C₁→YC₂
C₂→BC₃
C₃→AC₄
C₄→XB
A⇒XC₁⇒XYC₂
⇒XYBC₃⇒XYBAC₄
⇒XYBAXB
Ej. FNC

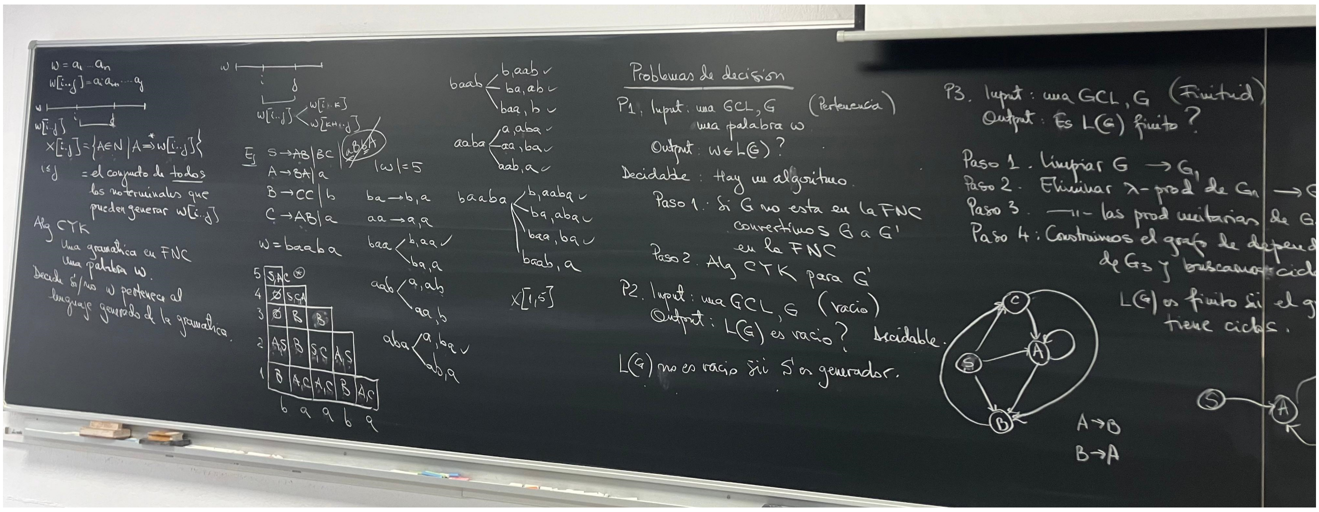



A→XAS|XS|XA|a
B→YY|SYSY|YSY|SYY
XAS|XS|XA|a
⑤ S→AC₁|SB|AB|AS|λ|XC₂|XS|XA|a|YY|SC₃|YC₄|C₄Y
C₁→SB
C₂→AS
C₃→YC₄
C₄→SY
A→XC₂|XS|XA|a
B→YY|SC₃|YC₄|C₄Y|XC₂|XS|XA|a
X→a
Y→b
S
w
Algoritmo CYK
* Cocke-Younger-Kasami

Note

Ver **11.8.1 Problema - *Pertenencia de Palabra***

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** $\to G_1$.
2. Realizar **CYK** para w en G_1.

> **Tip**
>
> ***Video Recomendado***

11.8.2 Problema - *Lenguaje Vacío*

$L(G)$ vacío?
Para resolver este problema:

- Probar si S es **generador**.

> **Tip**
>
> ***Video Recomendado***

11.8.3 Problema - *Lenguaje Finito*

$L(G)$ finito?
Para resolver este problema:

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

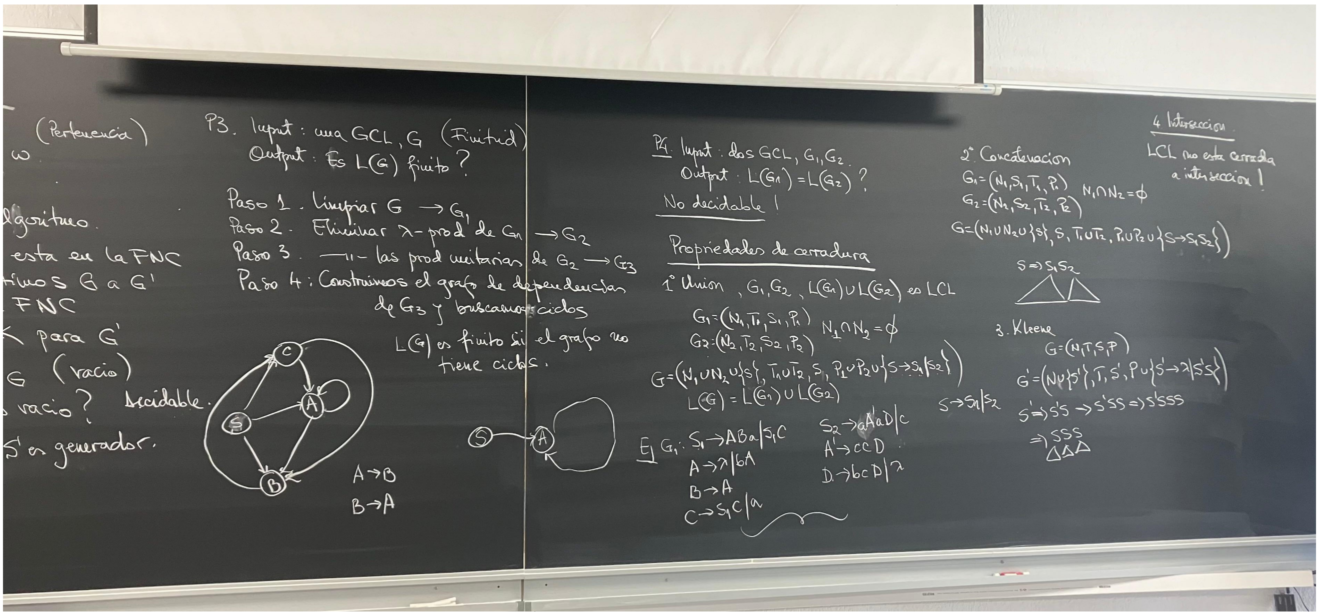
$L(G)$ será infinito cuando exista al menos un ciclo - *Ej.* $A \to AB$.

11.8.4 Problemas no Decidibles

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

> **Tip**
>
> ***Video Recomendado - GCL* ambigua**
>
> ***Video Recomendado - GLC = GLC***

11.9 Propiedades de Cerradura



Sean L_1, L_2 **LLC**'s (*Lenguajes de Libre Contexto*) y L_3 **LR** (*Lenguaje Regular*) :

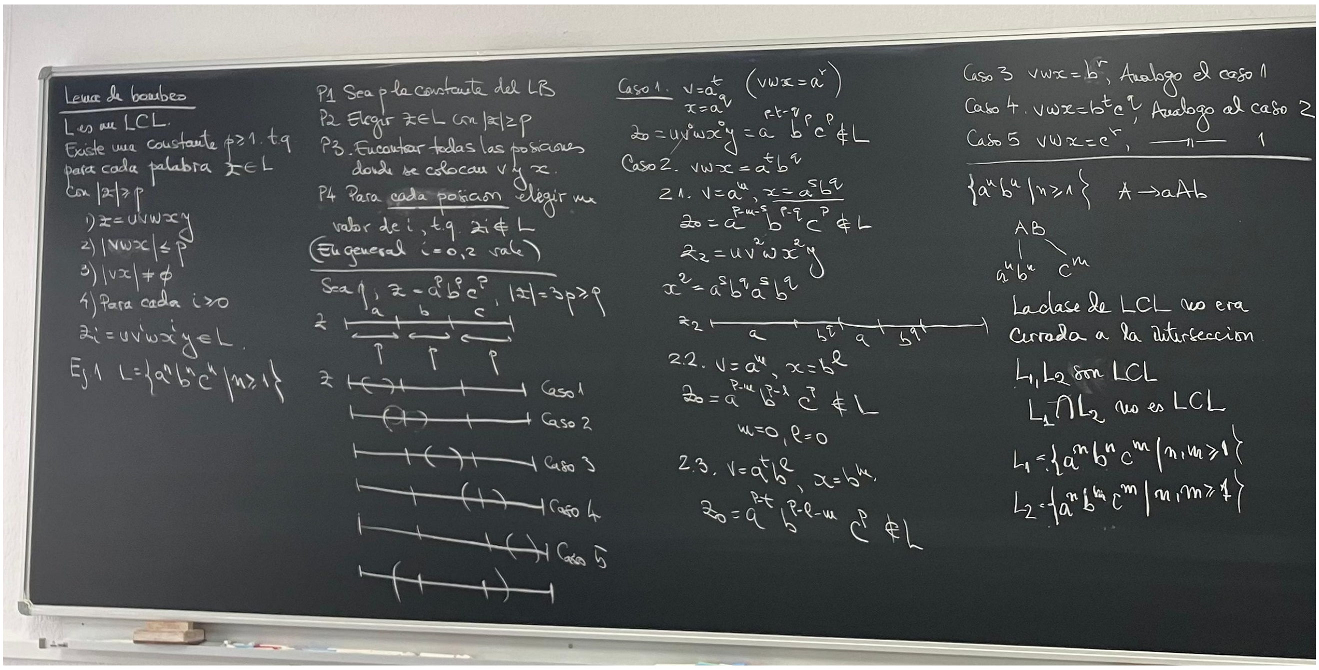
PROPIEDAD	OPERACIÓN	RESULTADO
Unión	$L = L_1 \cup L_2$	**LLC**
Intersección - **LLC**	$L = L_1 \cap L_2$	**LLC**
Intersección - **LR**	$L = L_1 \cap L_3$	**¿?**
Concatenación	$L = L_1 L_2$	**LLC**

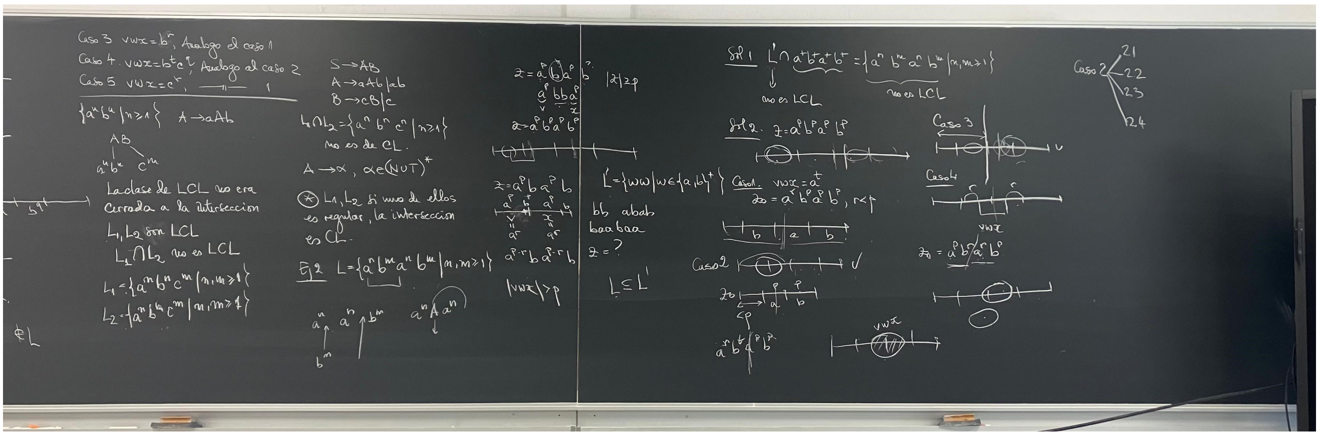
PROPIEDAD	OPERACIÓN	RESULTADO
Kleene	$L = L_1^*$	**LLC**
Inversión	$L = L_1^R$	**LLC**
Complementario	$L = L_1^C$	**¿?**
Diferencia	$L = L_1 - L_2$	**¿?**

Tip

Video Recomendado

11.10 Lema de Bombeo - *GCL*



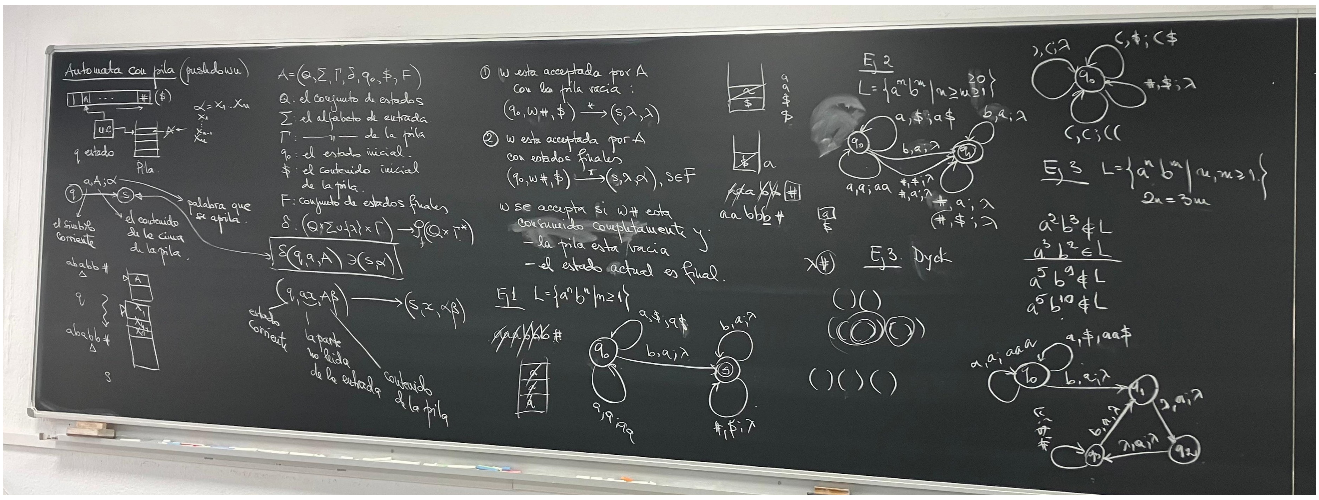


Tip

Video Recomendado - Teoría

Video Recomendado - Ejemplos

11.11 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 - $(q_i, a, A) \to (q_j, BB)$.
    - 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** : $\# \to]$
> - Centinela de **Pila** : $\$ \to \}$

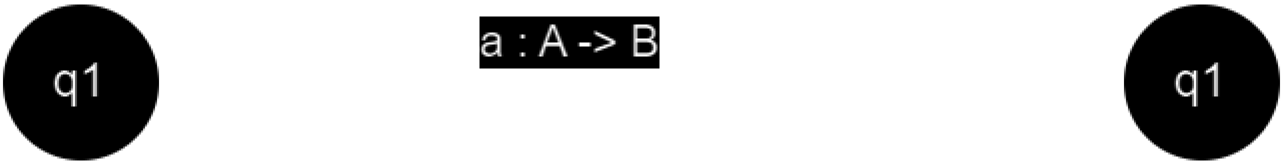
11.11.1 Representación Gráfica - *Grafos*

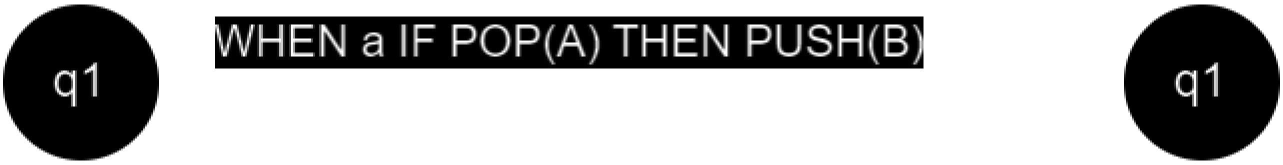
Se hace igual a que en grafos de expresiones regulares, excepto:

- 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:





11.11.2 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**.

11.11.3 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**.

> **Note**
>
> O en forma de código:

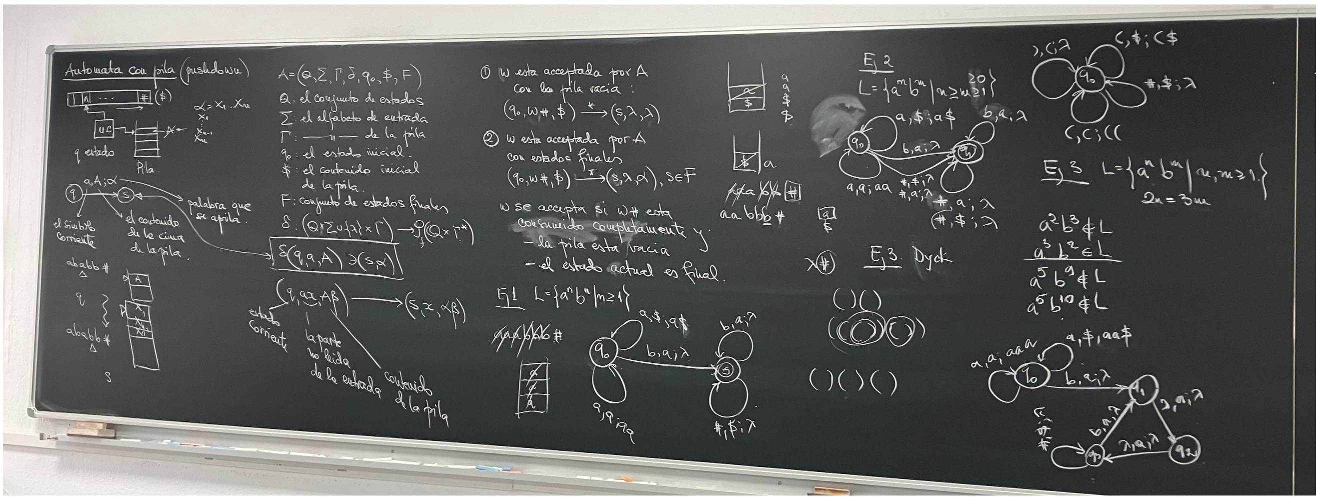
```
stack;
current_state;

FUNCTION consume (entry)                                        // entry es una letra
        transitions = current_state.transitions_with(entry)
        FOR transition IN transitions DO
                IF transition.stack_top_value == stack.peek() THEN  // peek devuelve top
                        stack.pop()                                     // pop quita top
                        stack.push(transition.stack_new_top_values)     // push añade a top
                        current_state = transition.next_state
                END
        END
END
```

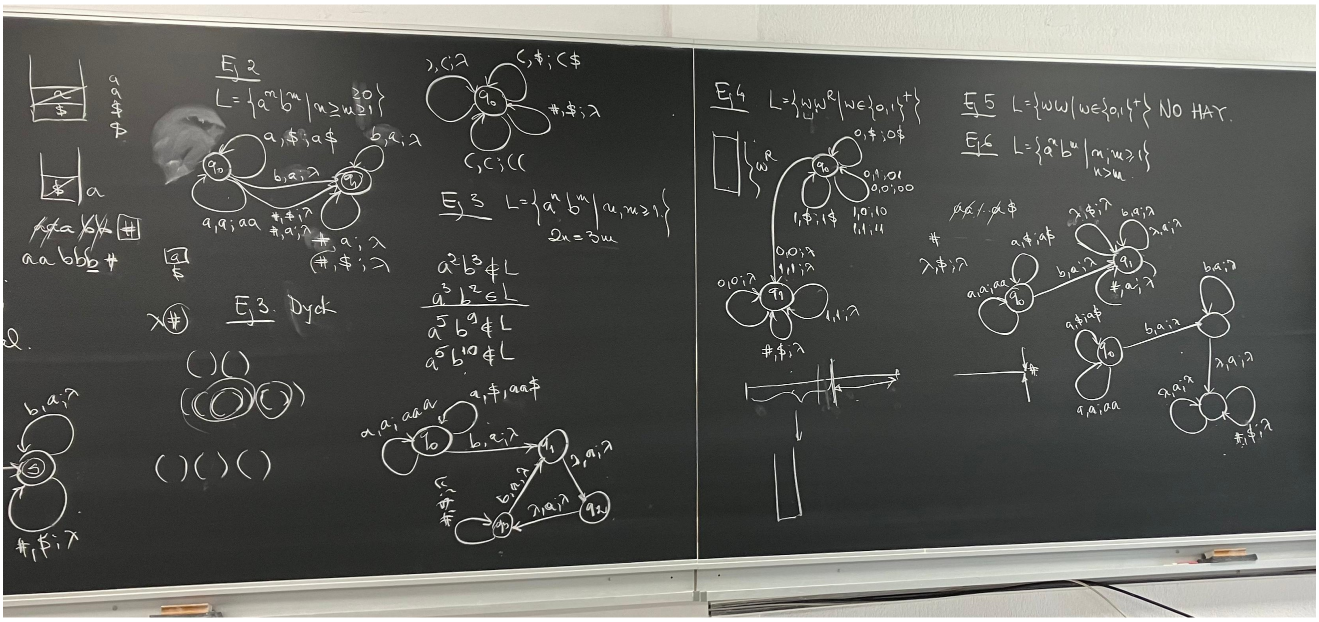
11.11.4 Aceptación

*Un **Pushdown Automaton** acepta palabras - w - cuando:*

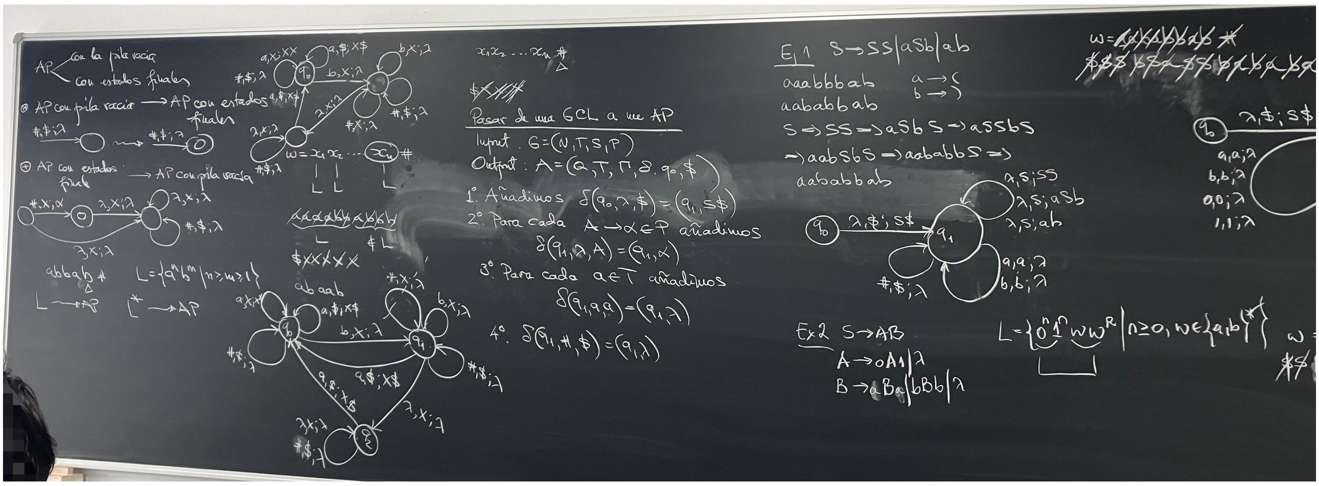
- Al consumir totalmente w, tiene la **Pila Vacía**.
- Al consumir totalmente w, está en un **Estado Final**.

11.12 Ejemplos: AP





11.13 Transformación: GCL a AP



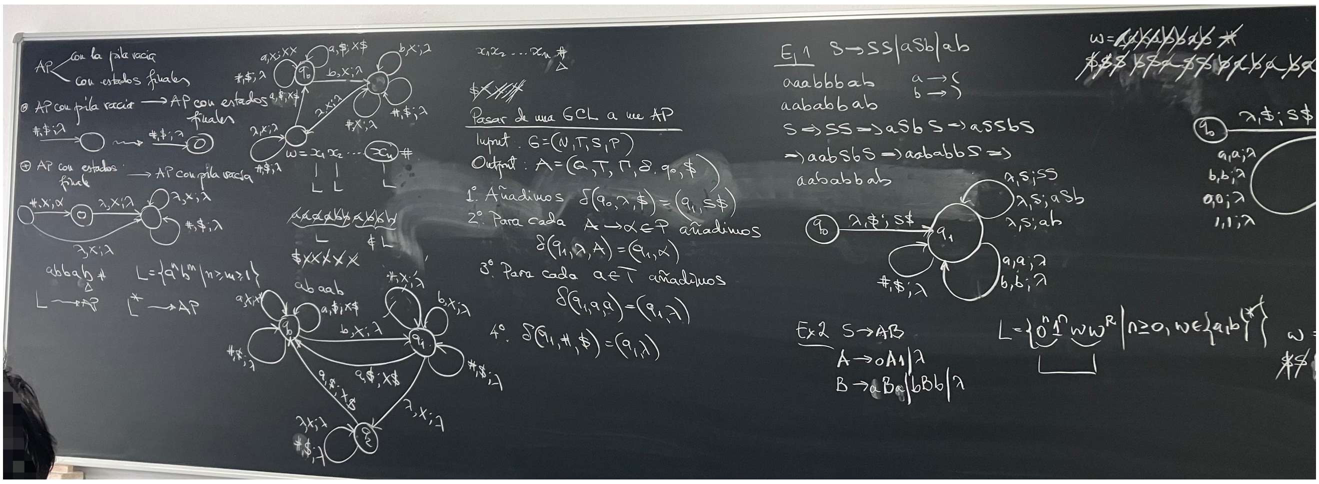
Pasos:

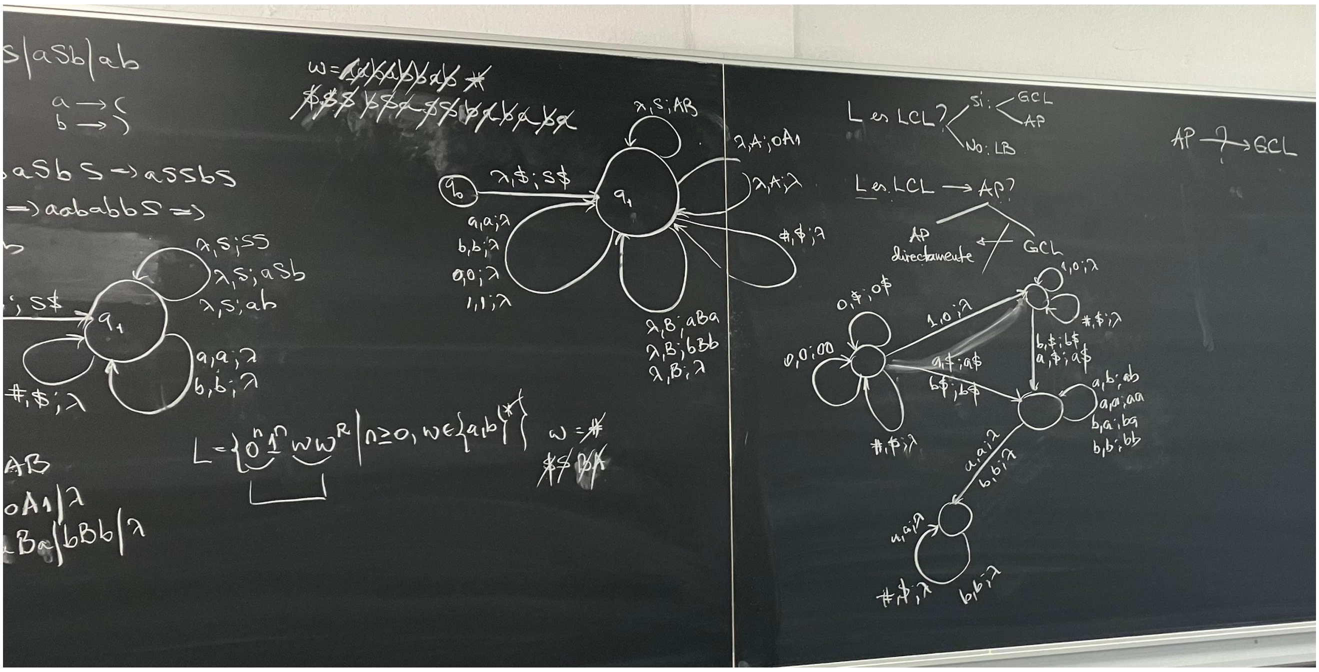
- **Añadir Estado Inicial** de inicialización.
- **Añadir Estado** para consumición.

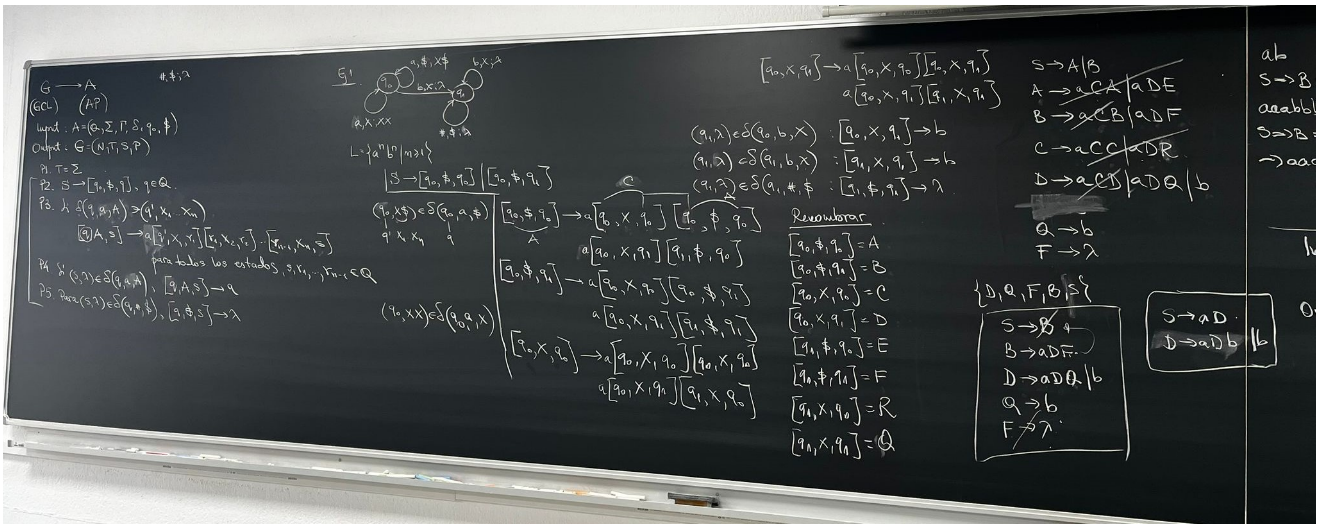
1. **Pushear** el **Centinela de Pila** con transición (λ, }, S}).
2. $\forall A \to X$: X es un implicado cualquiera, **creamos transición** (λ, A, X).
3. $\forall a \in \Sigma$, **creamos transición** $(a,\ a,\ \lambda)$.
4. **Creamos transición** para **Vaciado de Pila** ( ], }, λ).

> **Note**
>
> Solo con dos estados ya nos sirve, ya que definiremos todas entradas como lambda y solo se aceptará la palabra si al consumirla la pila está vacía. De esta forma construimos un AP lo suficientemente general como para aceptar cualquier gramática.
>
> El inconveniente es tener varias transiciones lambda y generar un AP No Determinista.

11.14 Ejemplos: GLC a AP







11.Extra Lenguaje a GCL

12. Sintaxis

*Las características sintácticas de un **LP** se pueden especificar mediante una **GCL**.*

*Esa especificación puede ser en formato **BNF** o **BNFA**.*

> **Note**
>
> - **LP**: Lenguaje de Programación.
> - **EBNF**: **BNFA**.
> - **AS**: Analizador Sintáctico.

12.1 Gramáticas en BNF y BNFA

12.1.1 BNF

Notación:

- **Símbolos No Terminales (A)**: `<no_term>`
- **Símbolos Terminales (a)**: `term`
- **Producción ($\rightarrow$)**: `::=`

12.1.1 BNFA

Facilita el entendimiento de gramáticas BNF, sobre todo la opcionalidad y repetición de elementos.

Notación (BNF + Nuevas):

- **Subpalabra Repetida**: `{subpalabra}`

  > *Subpalabra aparece **0 o más** veces.*

- **Subpalabra Opcional**: `[subpalabra]`

  > *Subpalabra aparece **0 o 1** vez.*

- **Subpalabra Variante**: `(subpalabra_1, subpalabra_2)`

  > *Aparece **subpalabra_1 o subpalabra_2**, etc.*

12.1.2 Ejemplo: BNF

Gramática en notación BNF que define la sintaxis de las declaraciones de variables en C.

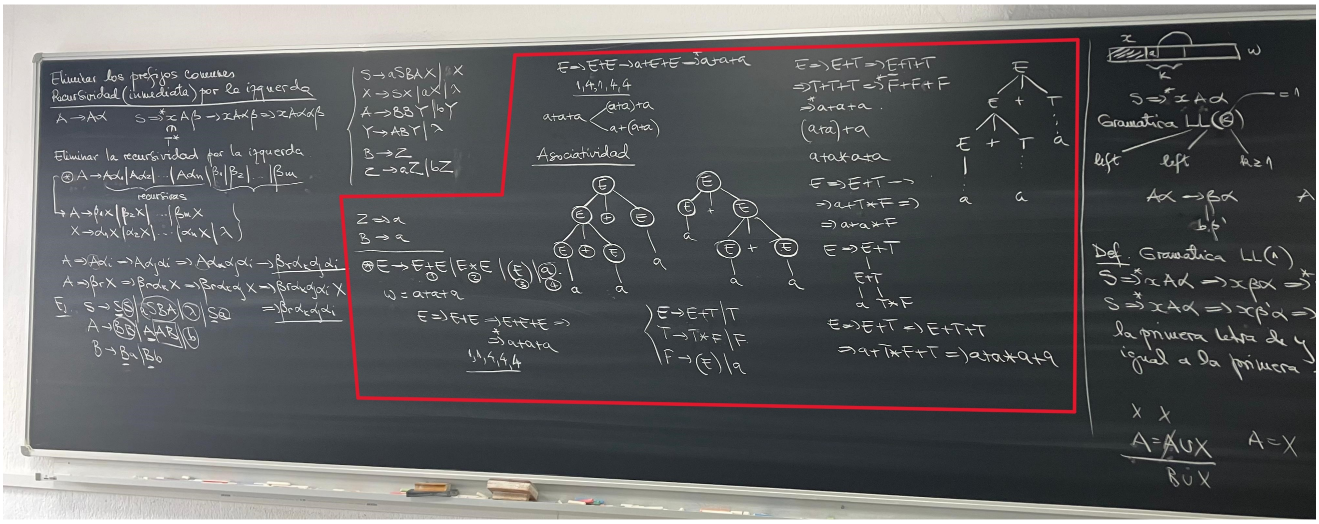
```
<declaraciones> ::= <una_declaracion> <declaraciones> | e
<una_declaracion> ::= <tipo> <lista_id_var> ;
<lista_id_var> ::= id <mas_id>
<mas_id> ::= , id <mas_id> | e
<tipo> ::= int | float
```

12.1.3 Ejemplo: BNFA

Gramática en notación BNF que define la sintaxis de las declaraciones de variables en C.

```
<declaraciones> ::= {<una_declaracion> ;}
<una_declaracion> ::= <tipo> id {, id}
<tipo> ::= int | float
```

12.2 Expresiones Aritméticas



Pg. 15 - 19 : Tema 6

Partimos de una gramática en BNF:

```
<expr> ::= <expr> + <expr> | <expr> * <expr>| (<expr>) | a
```

Su equivalente:

- $E \to E + E | E * E | (E) | a$

*Tenemos que resolver el problema de la **Precedencia entre Operadores**, para ello:*

- $\forall p, A_p \to O_p$

> Es decir, para cada nivel de precedencia, creamos un **Símbolo No Terminal** para sus operadores.

> ⚠ **Warning**
>
> **Menos Precedencia:** Más cerca al **Símbolo Inicial**.
>
> **Más Precedencia**: Más cerca a los **Operandos**.

Quedamos con la siguiente gramática en BNF:

```
<expr> ::= <expr> + <expr> | <term>
<term> ::= <term> * <func> | <func>
<func> ::= (<expr>) | a
```

> **Note**
>
> **Paréntesis**: Se trata como un Operando.
>
> **Función** Se trata también como un Operando.
>
> **Unarios (-)**: Tienen mayor precedencia.

La gramática es ambigua por lo tanto tenemos que quitarle la ambiguedad, quedando:

```
<expr> ::= <expr> + <term> | <term>
<term> ::= <term> * <func> | <func>
<func> ::= (<expr>) | a
```

La gramática en BNF es equivalente a:

- $E \rightarrow E + T|T$
- $T \rightarrow T * F|F$
- $F \rightarrow (E)|a$

12.3 Analizadores Sintácticos

Pag. 24 - : Tema 6

*Tiene como fin **reconstruir** y **comprobar** si los **tokens** proporcionados por el analizados léxico **pueden ser generados por la gramática** sintáctica que define el lenguaje.*

12.3.1 Símbolos por Adelantado

*Usados para **resolver** la **indeterminación** que se presente **en el proceso de reconstrucción** del árbol de derivación.*

> **Note**
>
> Es el numero de símbolos de la entrada que leemos a la vez.

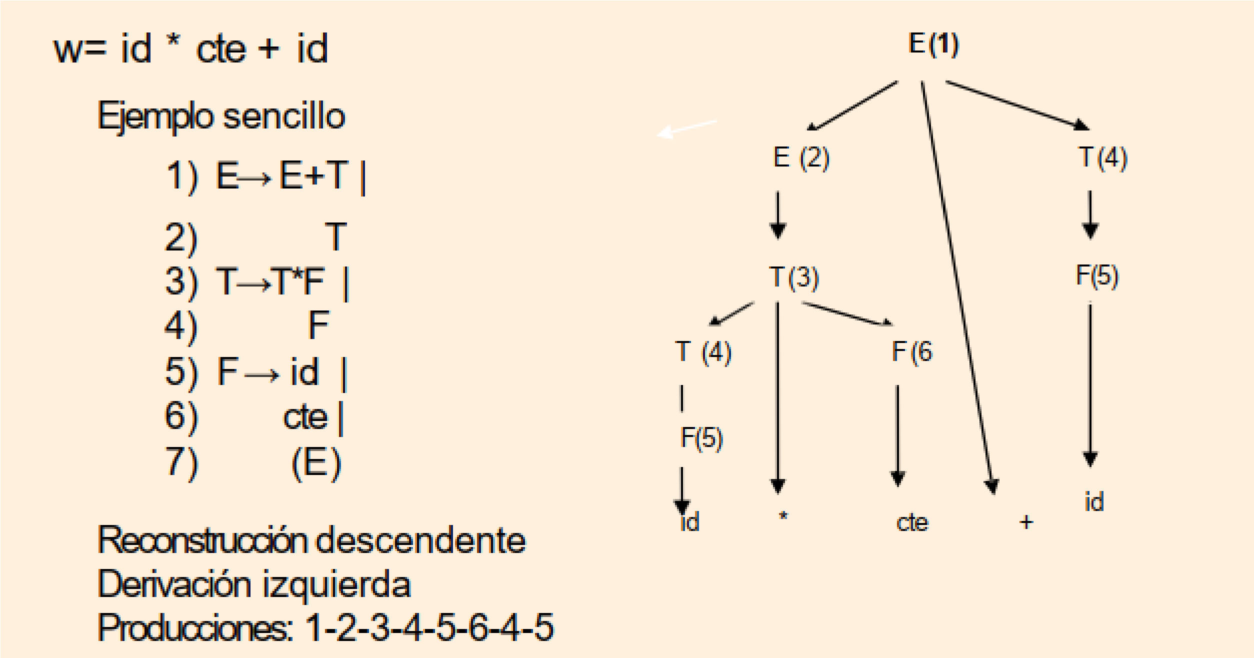
12.3.2 Reconocimiento Descendente - *LL*

Forma de los **Analizadores Sintácticos** de reconstruir el árbol **desde la raíz a las hojas**.

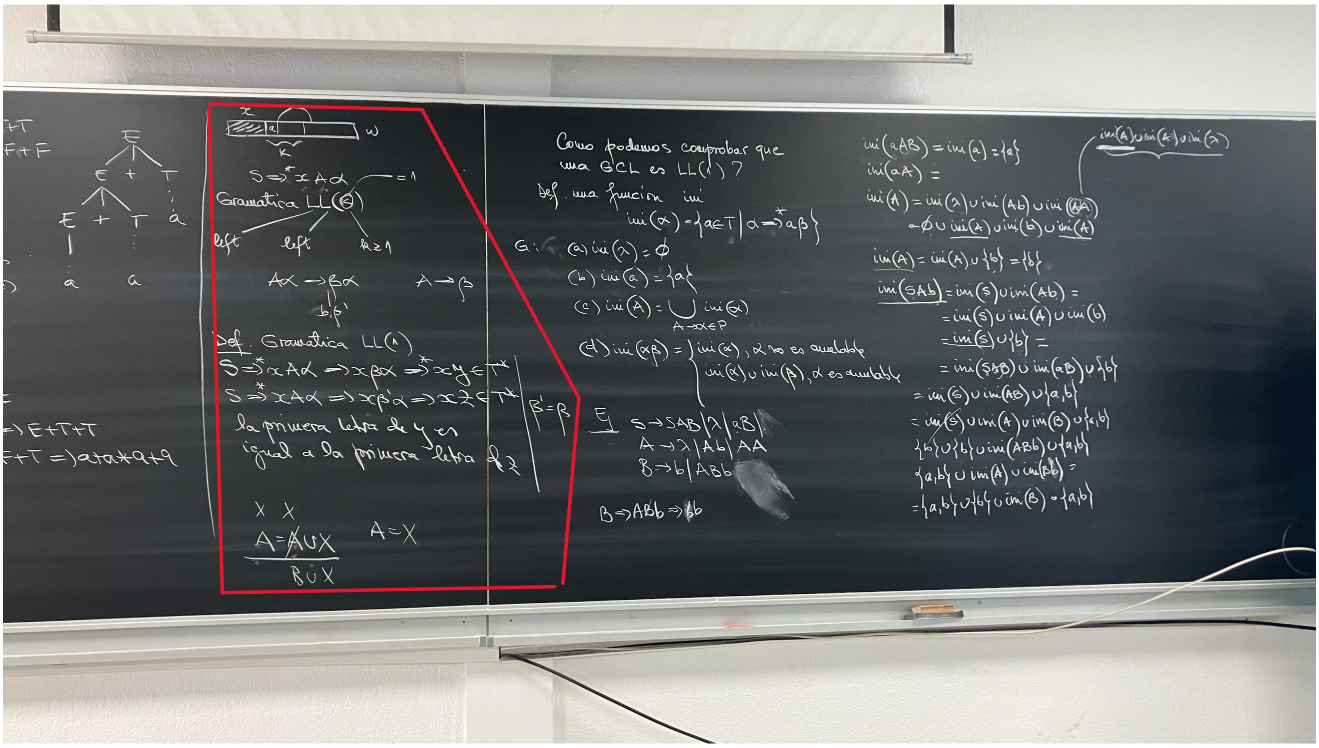
> **Note**

LL(K) : **L**eft Reading, **L**eft Derivation, con **K Símbolos por Adelantado.**

Ejemplo: Pag 26 : Tema 6



12.3.2.1 Definición LL(1)

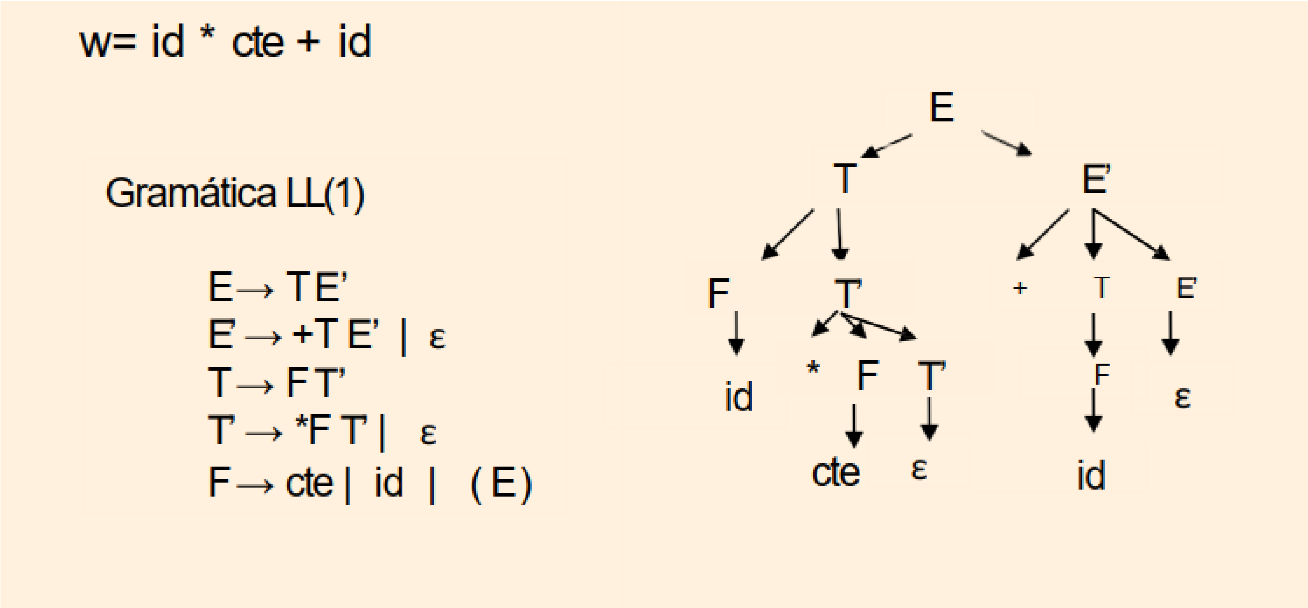


Note

En mis palabras: "*Una **GCL** está en **LL(1)** si puedes elegir la siguiente producción con solo leer 1 entrada*".

Sobre esta idea podemos asegurar que **si GCL es No Determinista, no es LL(1)**. Pero no podemos asegurar que un GCL Determinista sea siempre LL(1).

Ejemplo LL(1): Pag 27 : Tema 6



> **Note**
>
> Los pasos para saber si una **Sintaxis** definida por una **GCL** - *o por un **Lenguaje** definido por una **GCL*** - está en **LL(1)** se explican en el apartado [12.5 Sintaxis en LL(1)?]

12.4 Calculo de Símbolos Directores

12.4.1 ¿Qué realmente Representan?

12.4.2 Palabras Anulables

Pueden o no tomar un valor en una producción.

> **Note**
>
> Será **Anulable** siempre y cuando no exista ninguna **Subpalabra** en α que lleve a un **Símbolo Terminal** a .

Ejemplo:

- $A \rightarrow a|Bb$
- $B \rightarrow aC|\lambda$
- $C \rightarrow B|BA$

> **Note**

Es la palabra a de la producción $A \to a$ anulable?
*Obviamente no, ya que a ya es en sí **Símbolo Terminal**.*

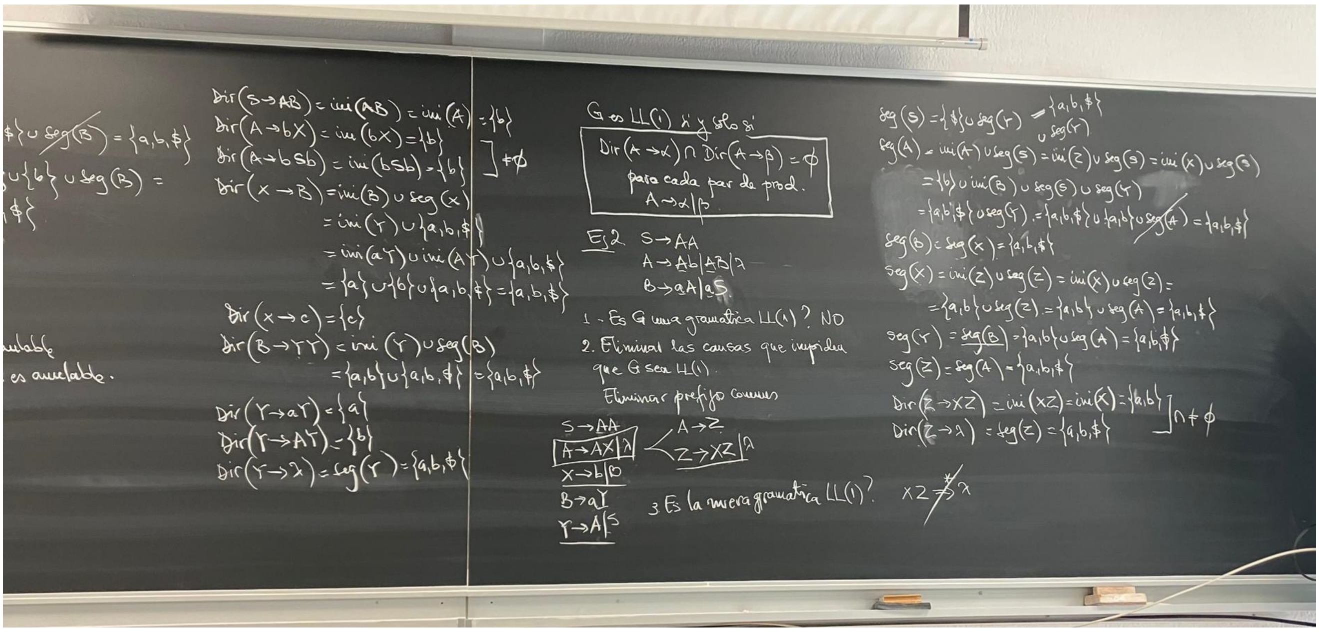
Es la palabra Bb de la producción $A \to Bb$ anulable?
No, ya que contiene un terminal - b.

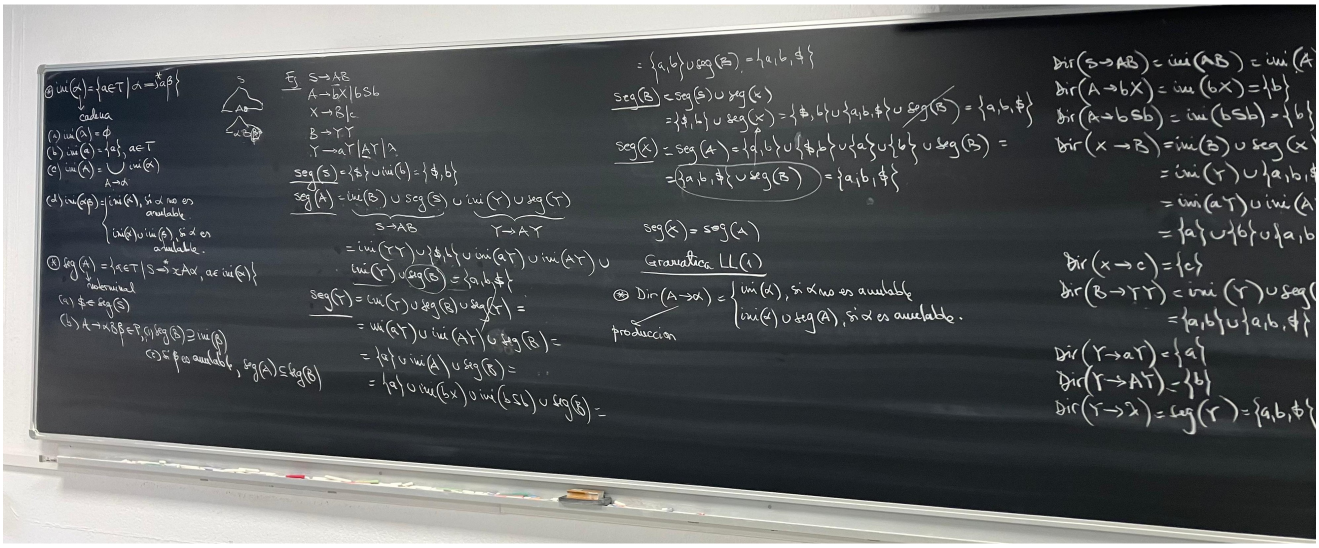
Es la palabra aC de la producción $B \to aC$ anulable?
No, ya que contiene un terminal - a.

Es la palabra λ de la producción $B \to \lambda$ anulable?
Obviamente sí, ya que λ es en sí la definición de nulo.

Es la palabra B de la producción $C \to B$ anulable?
Sí, ya que B podría ser nulo.

Es la palabra BA de la producción $C \to BA$ anulable?
No, ya que A no es anulable.





12.4.2 Calcular: Iniciales

Representación: **Inic(α)**

En los apuntes: $I(\alpha)$

> a : ***Subpalabra***.

$$I(\alpha) = \begin{cases} a & \text{si el primer Símbolo de } \alpha \text{ es Terminal} \\ I(\beta) \cup I(\alpha - \beta) & \text{en el caso contrario} \end{cases}$$

Note

En código:

```
FUNCTION init(alpha)
        x = first_symbol_of(alpha)
        IF x IS TERMINAL THEN
                x
        ELSE
                init(x) UNION init(alpha - x)
        END
END
```

12.4.3 Calcular: Seguidores

Representación: **Seg(X)**

En los apuntes: $S(X)$

> X : ***No Terminal***.

$$S(X) = \forall \beta \text{ que procede X } \begin{cases} I(\beta) & \text{si } \beta \text{ es No Anulable} \\ I(\beta) \cup S(A) & \text{en el caso contrario} \end{cases}$$

Note

En código:

```
FUNCTION seg(X)
        FOR beta EN LA DERECHA DE X DO
                IF beta IS NOT NULLABLE THEN
                        inic(beta)
                ELSE
                        inic(beta) UNION seg(A)
                END
        END
END
```

12.4.4 Calcular: Directores

Representación: **Dir($A \to \alpha$)**

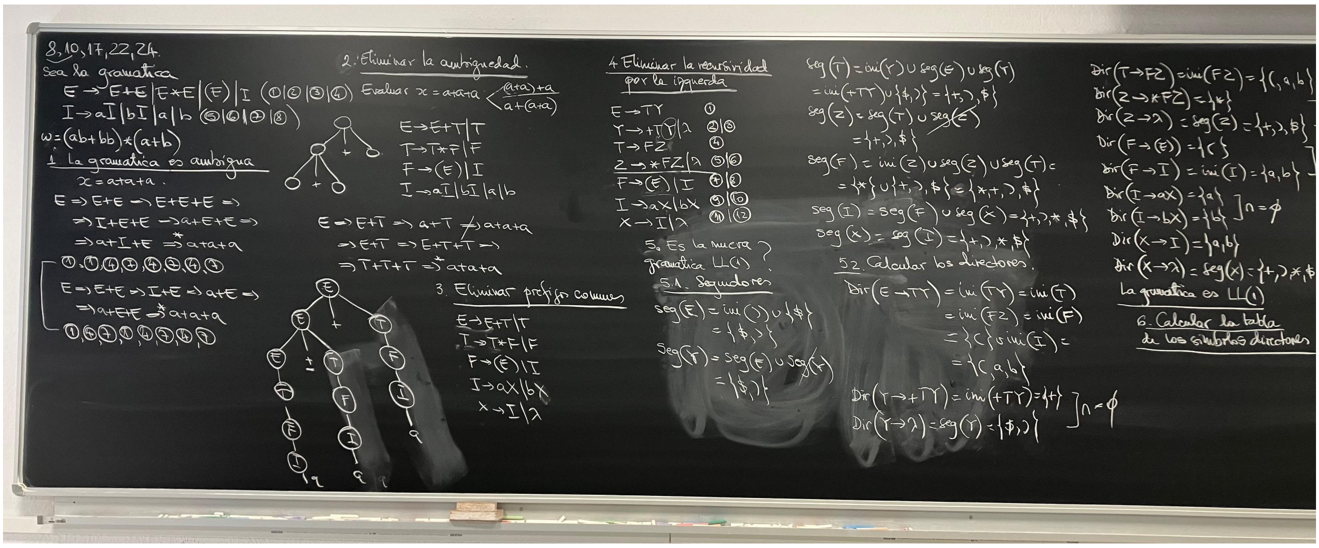
En los apuntes: $D(A \to \alpha)$

> A : ***No Terminal***.
> α : ***Subpalabra***.

$$D(A \to \alpha) = \begin{cases} I(\alpha) & \text{si } \alpha \text{ es No Anulable} \\ I(\alpha) \cup S(A) & \text{en el caso contrario} \end{cases}$$

> **Note**
>
> *En código:*
>
> ```
> FUNCTION dir(A, alpha)
>         IF alpha IS NOT NULLABLE THEN
>                 inic(alpha)
>         ELSE
>                 inic(alpha) UNION seg(A)
>         END
> END
> ```

12.5 Comprobar si Sintaxis está en LL(1)



Seguimos estos pasos:

1. **Eliminar Ambiguedad** [12] .
2. **Eliminar Prefijos Comunes** [12].
3. **Eliminar Recursividad por la Izquierda** [12].
4. **Calcular Todos Directores** [12].

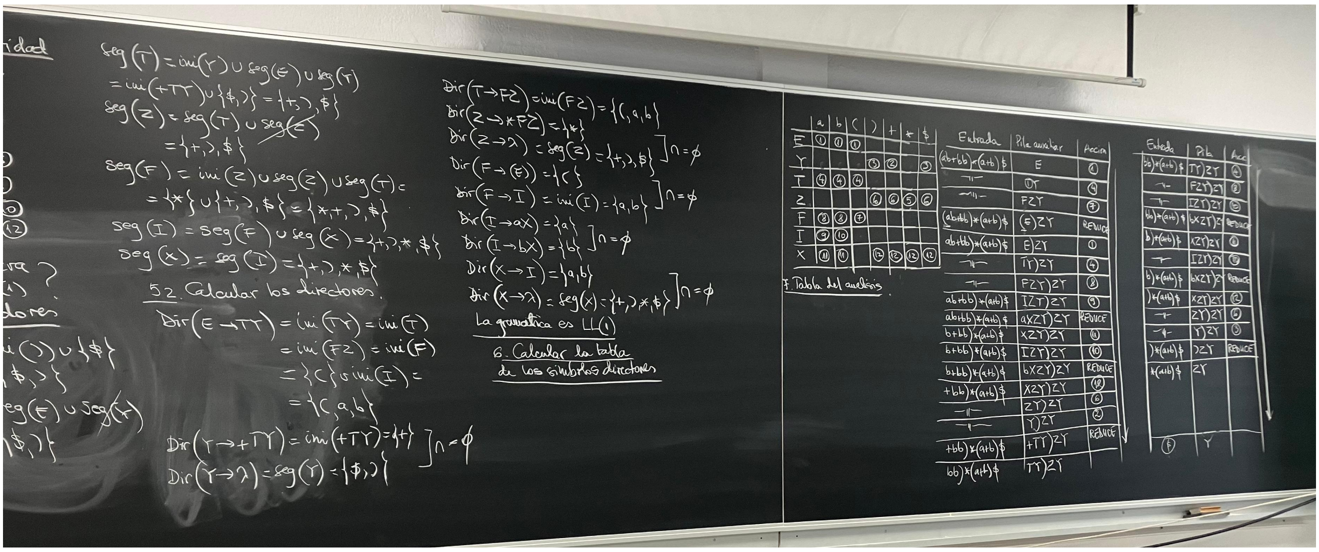
12.5.1 ¿Qué realmente Significa estar en LL(1)?

12.5.2 ¿Qué tiene que ver los Símbolos Directores con LL(1)?

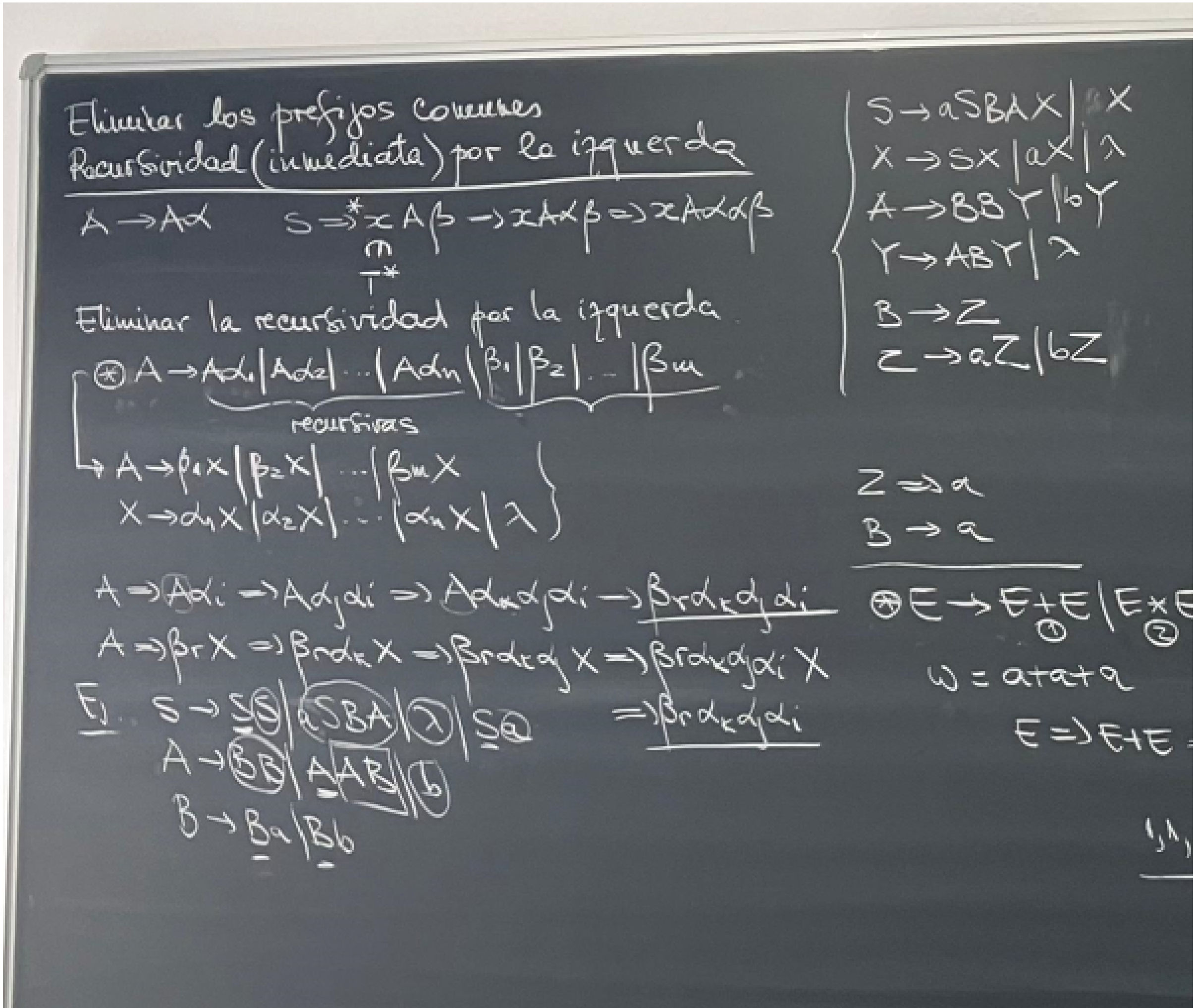
12.6 Analizadores Descendentes Predictivos Recursivos

12.7 Actividad - Analizadores Léxico-Sintácticos Automáticos

[POR HACER]



[DUDAS]



Como S ya nos lleva a aS y de S podríamos seguir por X, por transitividad se podría eliminar la producción X -> aX ?