

# Teoría de Lenguajes

## Primer parcial

Primer cuatrimestre de 2022

Apagar los celulares.

Hacer cada ejercicio en hojas separadas.

Poner nombre, número de libreta y firma en cada página.

Justificar todas las respuestas.

El examen es a libro abierto.

Se aprueba con al menos 65 puntos.

1. (25 pts) Sea  $L$  el lenguaje de las cadenas sobre el alfabeto  $\{+, -, 1\}$  que representa una secuencia de cero o más operaciones aritméticas válidas con el operando 1, en que sólo se admite sumar y restar de a 1, comenzando siempre con el 1, y el resultado final debe ser  $\geq 0$  (aunque resultados intermedios pueden tener cualquier signo).  
Ejemplos de cadenas de  $L$ :  $1, 1 - 1 + 1, 1 - 1 - 1 + 1 + 1 + 1 - 1, 1 + 1 - 1 - 1, 1 + 1 + 1 + 1 - 1$ .  
Ejemplos de cadenas que no están en  $L$ :  $\lambda, 11, 1++1, 1--1, +, +1-1, -1, -1+1+1+1, 1-1-1, 1-1+1-1-1$ .  
Exhibir una expresión regular para  $L$  o probar que no existe.
2. (25 pts) La *cadena de noticias* de la BBC del Once, agradecida por el uso continuo de su sigla en nuestra materia, ha decidido vender cadenas que podrían incluir apariciones de  $bbc$  entre otros caracteres, anteponiendo a cada cadena su precio dado por palotes usando la letra  $b$ . La cadena es correcta si la cantidad de apariciones de la sigla coincide con el precio. Ejemplo:  $bbcbccbbcacb$  es correcta. Hemos aplicado un transductor a todo esto y obtuvimos el ejercicio a resolver:  
Sea  $L = \{b^n\alpha/\alpha \in \{a, b, c\}^* \wedge n = \text{número de apariciones de } bbc \text{ en } \alpha\}$ . Exhibir un autómata de algún tipo que reconozca  $L$ , indicando claramente este tipo.
3. (25 pts) Definir una gramática del tipo más bajo posible en la jerarquía de Chomsky que genere el lenguaje  $L$  de las cadenas  $\omega$  de paréntesis y corchetes balanceados tales que  $]]] \notin Sub(Ini(\{\omega\}))$ .  
Ejemplos de cadenas en  $L$ :  $\lambda, (), ([][[]]), [[()]], [[[]]], (([]))((())((([[]]))), ((([[([[]])]])))).$   
Ejemplos de cadenas que no están en  $L$ :  $()[[(())]], [[()]], (, )[[], D(), ][-]$ .
4. (25 pts) Sobre el alfabeto  $\{0, 1\}$ , determinar si el lenguaje  $FIN((L(0^{++}(1^*)^2))^c)$  admite un autómata finito. De ser así, exhibir uno. De lo contrario, probarlo.

$$4) \text{ FIN} \left( L(0^{++}(1^*)^2)^c \right) \stackrel{\text{EQUIV}}{\approx} \text{ FIN} \left( L(0^+(1^*)^2)^c \right) \stackrel{\text{EQUIV}}{\approx} \text{ FIN} \left( L(0^+1^*)^c \right)$$

$\alpha^{++} = \alpha^+$  ✓       $\alpha^*\alpha^* = \alpha^*$  ✓

ER  $\rightarrow$  AF (el complemento vale igual en AF)

ABM  $L_0 = 0^+1^*$



$$\partial_0(L_0) = \partial_0(0^+1^*) = \partial_0(0^+)1^* \mid \in (0^+) \quad \partial_0(1^*) = 0^*1^* \quad L_1^* \text{ (FINAL)}$$

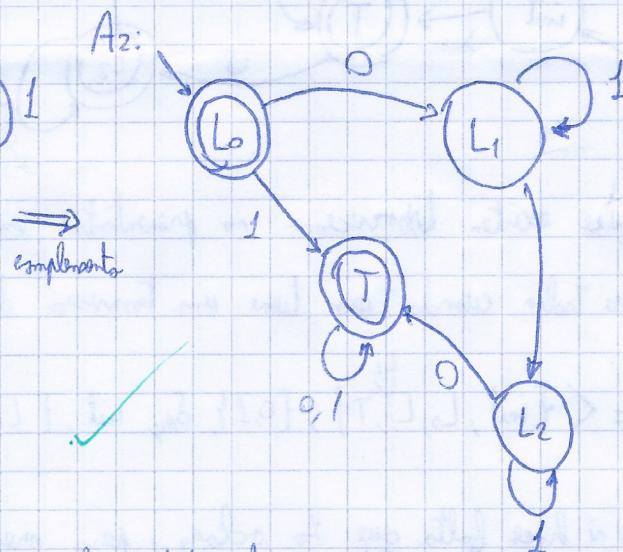
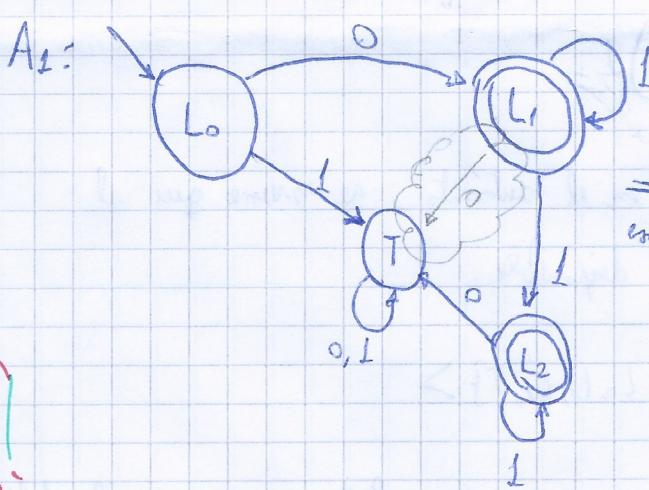
$$\partial_1(L_0) = \partial_1(0^+1^*) = \partial_1(0^+)1^* \mid \in (0^+) \quad \partial_1(1^*) = \emptyset \quad T$$

$$\partial_0(L_1) = \partial_0(0^*1^*) = 0^*1^* \mid \in (1^*) \quad \partial_0(1^*) = 0^*1^* \quad L_1$$

$$\partial_1(L_1) = \partial_1(0^*1^*) = \emptyset \mid 1^* = 1^* \quad L_2^*$$

$$\partial_0(L_2) = \partial_0(1^*) = \emptyset \quad T$$

$$\partial_1(L_2) = \partial_1(1^*) = 1^* \quad L_2$$



complemento

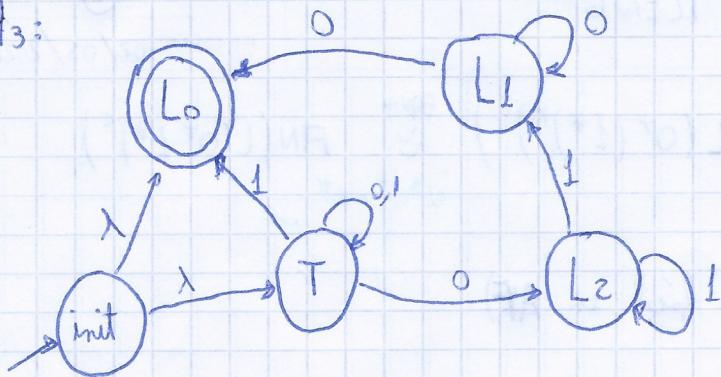


Para el complemento debemos invertir los estados de aceptación (porque es un AFD)

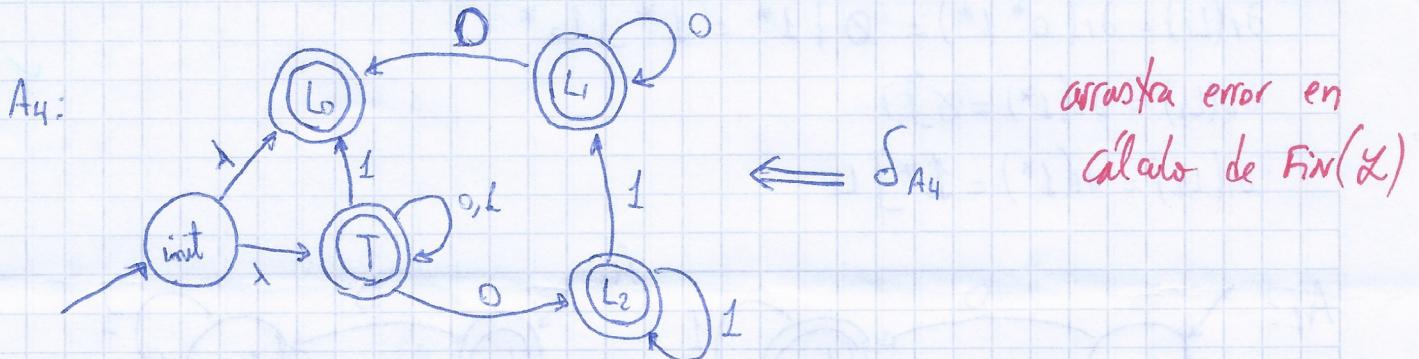
del autómata.  $\checkmark$  Hasta ahora A<sub>1</sub> es el autómata asociado a  $L(0^+1^*)$ .

Luego A<sub>2</sub> es el complemento de A<sub>1</sub>, entonces A<sub>2</sub> se traduce a  $L(0^+1^*)^c$ ,

Para conseguir  $\text{FIN} \left( L(0^+1^*)^c \right)$  primero escribimos como  $\text{INI} \left( (L(0^+1^*))^c \right)$   $\times$   
 ya que INI y reversa son más fáciles de calcular que FIN. Para conseguir el reverso  
 se toma a todos los estados finales como el estado inicial, al estado inicial como  
 estado final y se invierten todas las combinaciones, mostrando ahora A<sub>3</sub> que es A<sub>2</sub>'  
 "muy", te permites que b.  
 hija no solo  $\times$

A<sub>3</sub>:

Por último los estados iniciales se convierten en estados finales todo estados donde trabaja se pone mejor a un estado final. Así obtenemos nuestro último autómata A<sub>4</sub><sup>\*</sup> que representa  $\text{FIN}((0^+1^*)^c)$



Notar que existe transiciones no presentadas en el autómata, se presume que el mismo se trabaja cuando no tiene una transición disponible.

$$A_4 = \langle \{\text{init}, L_0, L_1, T\}, \{0, 1\}, \delta_{A_4}, \text{init}, \{L_0, L_1, L_2, T\} \rangle$$

No se hace falta que lo oculte, pero mejor que no -roba y no que falle, A<sub>4</sub> es un AFND-λ.

(25 puntos)

1) Tiene todos los puntos de no ser regular porque tiene que componer la cantidad de "+1" contra la cantidad de "-1". Tratemos de probarlo usando pumping.

1) Me dan un  $P > 0$

2) Elijo  $w \in L / |w| \geq P$ .  $w = 1(+1)^p (-1)^{p+1}$  ✓

3) Me dan una descomposición  $w = xyz$ ,  $|xy| \leq P$  y  $|y| > 0$

4) Elijo  $i \geq 0 / w' = xy^i z \notin L$

Como  $|xy| \leq P$ , y esto tiene ~~que~~ "numeradas". O sea que tienen las siguientes probabilidades

a)  $y = 1(+1)^*$

Si este es el caso  $i=0$  nos dejó una cadena inválida ya que terminaría con  $+1\dots$  (empienza con +) ni el primer uno pertenece a  $Y$  o con  $1+1\dots++1$  (repite 2 + seguidos). O sea que para cualquier cosa  $w' = xyz^0 z \notin L$ . ✓

b)  $+ (1+)^*$

Mi idea fue el a). Así con  $i=0$  tuvo a tener  $1+...11\dots1$  (repite 2 1 seguidos). O sea que  $w' = xyz^0 z \notin L$  ✓

c)  $(1+)^+ \circ (+1)^+$

Aquí si cuando  $i=0$  la cadena va a seguir siendo "correcta", salvo por un detalle, la coma final va a ser menor a 0. Esto es porque  $1(+1)^p (-1)^{p+1} = 0 \forall p$ . )

Entonces si se cumple 1 o más reglas de NFA a término siempre con un número negativo. (podemos afirmar esto porque  $|y| > 0$ )

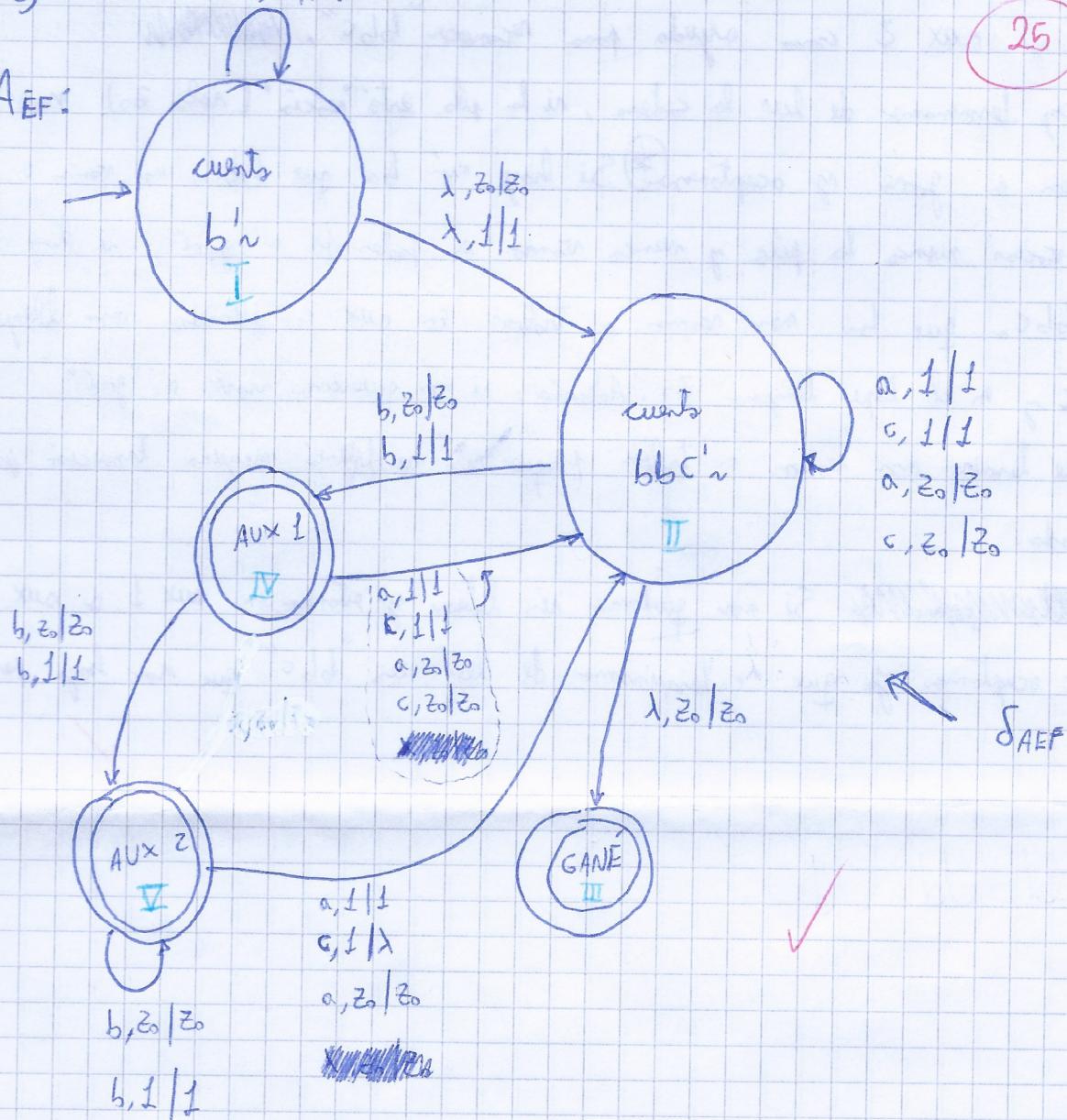
$\Rightarrow L \neq L'$

Como logré responder en  $\forall p$  y  $\forall x,y,z$  (que cumplen las hipótesis) puedes afirmar que el pumping que  $L$  es no regular. ✓

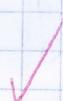
2)

$b, 1 \mid \lambda$   
 $b, z_0 \mid z_0$

A<sub>EF</sub>:



$\delta_{A_EF}$



EF an in estado final

$$A_EF = \langle \{I, II, III, IV\}, \{a, b, c\}, \{z_0, 1\}, \delta_{A_EF}, I, z_0, \{III\} \rangle$$

Idea: Vamos a crear un autómata de pilas no determinístico, ya que tenemos que componer la entrada de los iniciales con la entrada de  $bcc_n$  de  $\delta_{A_EF}$  y tenemos que hacerlo para todo  $n$ . Separamos el autómata en 2 etapas, primera contamos las "b" del principio y apilamos "1" y después contamos "bcc" y desapilamos "1". En decir, al principio esta b me apila un 1 y después cada  $bcc$  lo desapila. ✓

Al ser las determinísticas, el paso entre estados  $b_1$  y estados  $b_2$  se los podemos hacer más o menos más rápido para aceptar. En la segunda etapa usamos a aux 1 y aux 2 como ayuda para reconocer " $b_2$ ". ~~Algunas~~

Una vez terminamos de leer la cadena, si la pila está vacía (nulos  $Z_0$ ) nos movemos a  $g_{01}$  y aceptamos. (\*) Si hoy más bien que  $b_2$  no vamos a poder "vaciar" nunca la pila y nunca vamos a poder ir a  $g_{01}$ , si hoy más bien  $b_2$  nos vamos a tratar en aux 2 cuando no llegue ningun  $C$  y en el tope tengamos  $Z_0$ . Adicional, si no queremos llegar a " $g_{01}$ " antes de terminar nos vamos a tratar porque no se definió ninguna transición para ese estado.

(\*) ~~Algunas~~ Si no quedamos ni bien ni mal en aux 1 o aux 2 también aceptamos ya que no terminamos de leer un " $b_2$ " que no haya pasado. ✓

3) Para generar el lenguaje pedido necesitamos un gramática libre de contexto, en particular nivela la siguiente (ni dice quiere):

$$G = \langle \{A, B, C\}, \{(, [, ], )\}, P, A \rangle$$

P:

$$A = AA \mid (A) \mid \lambda \mid [B] \mid B$$

$$B = BB \mid (A) \mid \lambda \mid [C] \mid C$$

$$C = CC \mid (A) \mid \lambda \mid () \mid [ ]()$$

Me parece que no son necesarios  
La idea en general está bien, pero  
En B y C deberíais generar cualquier cosa válida  
a izq de ( ) o [ ]. Por ej. no generas  
[ [ [ ] ( ) ] ] ( )  
 $A \Rightarrow [B] \Rightarrow [BB]$   
redondeando con  $C \Rightarrow (A)$ ,  $A \Rightarrow \lambda$   $\Rightarrow [ [ [ ] ( ) ] ]$   
 $B \Rightarrow [B]$   
B nunca puede  
ser ], pues  
 $A \Rightarrow [B]$

Ahora me toca tratar de explicar esta magia que hace.

Los primeros que dice free: tengo un producto  $A = AA \mid (A) \mid \lambda$ .

Con eso puedes generar cualquier tipo de parentesis balanceados, pero ya también quieras poner meter corchetes, pero son más difíciles. Entonces voy a agregar  $| [B] | B$ . Esto me lleva al producto B, que básicamente es "cualquier

que sea capaz de abrir un corchete. Ahora, como ya puse abrir otro corchete sin cerrar voy a agregar  $B = [C] | \tau$  y obtendré un resultado lo mismo producto que antes  $BB \mid (A) \mid \lambda$ . (A) básicamente restan cosas, que es lo que debería poner cuando meto un parentesis.  $[C]$  es mi letra final, si estoy en C no pongo dos mas para en falso porque

son tipos tristes. Tengo un par de opciones igual:  $C = CC \mid (A) \mid \lambda \mid () \mid [ ]()$

puedo reiniciar con (A), terminar  $\lambda$ ,  $()$  o  $[ ]()$  o even ~~terminar~~ comenzar con  $CC$ . Notar que  $[ ]()$  no rompe nada porque me asegura no tener  $] ]$

para ni permitir  $[ [ [$ . La derrota no rompe nada porque no crean ].

Entonces esas reglas que algunas producciones robaban, parecen mejor que no robar

Por si te interesa querida lectora, hice los temas y posemos devolver. :)



Para arreglarlo deberíamos tener algo así como

$$B \rightarrow AB | [C] | (A) | \lambda$$

pero si B es available y  $B \rightarrow AB$ ,  $B \Rightarrow A$  y generamos [[[ ]]]  
una forma de erradicarlo es

$$B \rightarrow A(A) | A[C] | \lambda$$

Análogamente

$$C \rightarrow A(A) | \lambda$$

Eso es porque antes de un ( ) puede ir cualquier cosa, ya que  
nos molestaran los ] a dercha y es lo "reserva".

Se puede ver como que A permite 2] a der., B 1 y C ninguno.

