1. **(3 ptos.)** Dada la siguiente gramática:

S	→	A b z | B z	B	→	b	D	→	a S | D b
A	→	C D a z | C D z	C	→	(S) | ϵ			

a) ¿Porqué no es LL(1)? Utilizando las transformaciones vistas en clase, reescribir la gramática eliminando los posibles problemas detectados.

b) Para la gramática resultante, calculad los `primeros` de todas las partes derechas de las reglas y los `siguientes` de todos sus no-terminales.

c) Construid la tabla de análisis LL(1). ¿Es una gramática LL(1) y porqué?

d) A partir de la tabla LL(1), obtened la traza LL(1) para la cadena: `a b z z b z`

---

a) No es LL(1) porque tiene recursividad a izquierdas (D) y factorización a izquierdas (A). Aplicando las transformaciones de clase, queda:

S	→	A b z | B z	A	→	C D A'	A'	→	a z | z
B	→	b	C	→	(S) | ϵ	D	→	a S D'
D'	→	b D' | ϵ						

b) Los `primeros` y `siguientes` son,

PRI(A b z)	= { (, a };	PRI(b)	= { b };	SIG(S)	= { $,), b, a, z };
PRI(B z)	= { b };	PRI((S))	= { (};	SIG(A)	= { b };
PRI(C D A')	= { (, a };	PRI(a S D')	= { a };	SIG(A')	= { b };
PRI(a z)	= { a };	PRI(b D')	= { b };	SIG(B)	= { z };
PRI(z)	= { z };	PRI(ϵ)	= { ϵ };	SIG(C)	= { a };
				SIG(D)	= { z, a };
				SIG(D')	= { z, a };

c) La tabla de análisis LL(1), solo para los no-terminales, es:

	a	**b**	**z**	**(**	**)**	**$**
S	(Abz, 1)	(Bz, 2)		(Abz, 1)		
A	(CDA', 3)			(CDA', 3)		
A'	(az, 4)		(z, 5)			
B		(b, 6)				
C	(ϵ, 8)			((S),7)		
D	(aSD', 9)					
D'	(ϵ,11)	(bD', 10)	(ϵ,11)			

d) Y la traza para `a b z z b z`

S $	a b z z b z $	–
A b z $	a b z z b z $	1
C D A' b z $	a b z z b z $	1 3
D A' b z $	a b z z b z $	1 3 8
a S D' A' b z $	a b z z b z $	1 3 8 9
S D' A' b z $	b z z b z $	1 3 8 9
B z D' A' b z $	b z z b z $	1 3 8 9 2
b z D' A' b z $	b z z b z $	1 3 8 9 2 6
D' A' b z $	z b z $	1 3 8 9 2 6
A' b z $	z b z $	1 3 8 9 2 6 11
z b z $	z b z $	1 3 8 9 2 6 11 5
$	$	1 3 8 9 2 6 11 5

4. **(2 ptos.)** Cuestiones teóricas (contestad brevemente):

a) Considerando la siguiente especificación (seudo-)FLEX,

```
aa*b*c*      { printf("1"); }
c*b*         { printf("2"); }
```

y dada la siguiente cadena de entrada, `babcaababccbcabb`,
¿Cuál será la previsible salida del Analizador Léxico?

b) Para una gramática bien formada; es decir, que no tiene símbolos inútiles y todos sus símbolos no-terminales son accesibles, demostrad que en una tabla LL(1) todas las filas, asociadas a los no-terminales, tienen al menos una acción `derivar`.

c) Dado el autómata de prefijos viables (colección canónica de conjuntos de ítems LR(0)) del apartado `2.a`, ¿cuál de las siguientes cadenas son prefijos viables de alguna de sus formas sentenciales a derechas? Justificad brevemente la respuesta. Si hay alguna que sea prefijo viable, proporcionad además el conjunto de ítems válidos para dicho prefijo viable.

- `if E + E`
- `if E I else break`

d) Suponiendo realizada la fase de declaración de los objetos (inferencia), diseñad un ETDS para la comprobación de tipos asociado a la regla:

$$I \rightarrow * id = E ;$$

---

a)

b	a b c	a a b	a b c c	b	c	a b b
2	1	1	1	2	2	1

b) Lo demostraremos por reducción al absurdo. Por construcción de la TA LL(1), para que una fila asociada a un no terminal, $A \in N$, tenga todas sus columnas con la acción `error` se debe cumplir que: $\forall k : A \rightarrow \alpha$; `primeros`$(\alpha$ `siguiente`$(A)) = \emptyset$. La única forma de que esto suceda es que $\alpha = \epsilon$ y `siguiente`$(A) = \emptyset$. Pero si `siguiente`$(A) = \emptyset$ implica que A no será accesible y por tanto la gramática no estará bien formada; invalidando el argumento inicial.

c) Ambas cadenas son prefijos viables ya que existe un camino en el autómata de prefijos viables desde el estado inicial hasta un estado que reconoce cada una de ellas.

Para `if E + E`, el conjunto de ítems válidos es: {`[E ->E + E .], [E ->E . + E ]`}

Para `if E I else break`, el conjunto de ítems válidos es: {`[I ->break .]`}

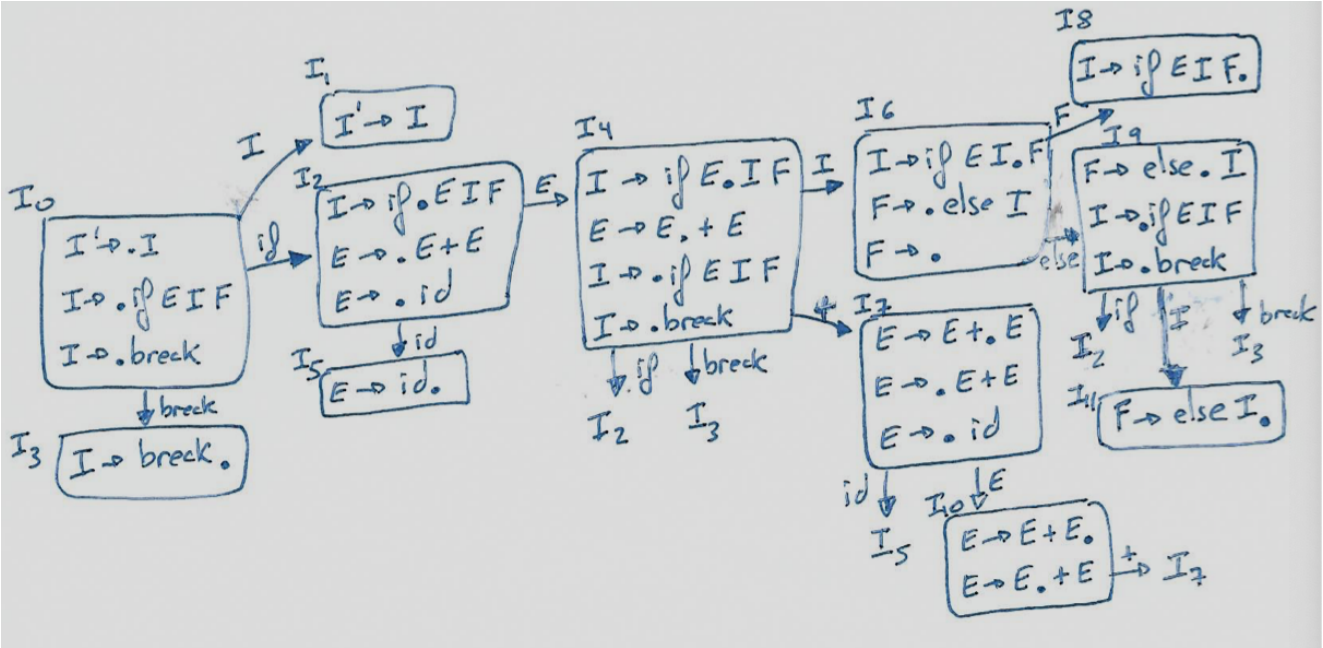
d)

I $\Rightarrow$ * id = E ;	**si** ¬[`obtTdS`(id.n, id.t) $\wedge$ (id.t = `tpuntero`(id.tap)) $\wedge$ (id.tap = E.t)] `MenError`(.);

2.- Dada la gramatica

I -> if E I F
I -> break
E -> E + E
E -> id
F -> else I
F -> ε

a) (1,5 pto) Construid la colección canónica de conjuntos de ítems LR(0).



b) (1,25 pto) A partir de dicha colección, construid la tabla de análisis SLR(1). ¿Es una gramática SLR(1)? Justificad brevemente la respuesta.

SIG(I)= { $, else}
SIG(E)= { if, break, +}
SIG(F)= { $, else}

	if	break	+	id	else	$	I	E	F
0	d2	d3					1		
1						Acept.			
2				d5				4	
3					r2	r2			
4	d2	d3	d7				6		
5	r4	r4	r4						
6					d9/r6	r6			8
7				d5				10	
8					r1	r1			
9	d2	d3					11		
10	r3	r3	d7/r3						
11					r5	r5			

c) (0,75 pto) Sabiendo que el operador + es asociativo a izquierdas y que un "else" debe asociarse al "if" más próximo, resolved adecuadamente los posibles conflictos.

I_6, con un else: Desplazar para que el else vaya asociado al if que le precede.
I_{10} con un +: Reducir porque el + es asociativo a izquierdas.

3.- Dadas las siguientes producciones correspondientes a una instrucción condicional,

I -> if id I else I
I -> break
I -> id=cte
...

a) (0.75 pto) Escribid un ETDS que deje en el atributo I.numb la cantidad de instrucciones "break" contenidas en la instrucción I
b) (0.75 pto) Proporciona un ETDS que deje en el atributo I.maxniv el nivel máximo de anidamiento en el que se encuentra una instrucción "break"

Ejemplo:
if id id=cte
else if id if id break
else id=cte
else break

El ejemplo dejaría en el símbolo inicial I.numb= 2 y I.maxniv= 3

I -> if id I_1 else I_2	{ I.numb = I_1.numb + I_2.num ; if I_1.maxniv <> 0 I*1*.maxniv=I_1.maxniv + 1 ; if I_2.maxniv <> 0 I2.maxniv=I_2.maxniv + 1 ; I.maxniv = max(I_1.maxniv, I_2.maxniv); }
I -> break	{ I.numb = 1 ; I.maxniv = 1; }
I -> id=cte ...	{ I.numb = 1 ; I.maxniv = 0; }