Recuperatorio de Lenguajes Formales y Autómatas.

1) *Indique cuales de estas afirmaciones son falsas y justifíquelas. (25 pts.)*

a) Una gramática es ambigua si no se puede resolver mediante LR canónico.

Falso. Una gramática es ambigua si para alguna cadena de entrada se pueden encontrar 2 árboles de derivación distintos.

b) La construcción de Thompson es un método para, a partir de una expresión regular, generar un AFD de estados mínimos.

Falso. La construcción de Thompson es un método para, a partir de una ER, generar una AFND.

c) Toda gramática analizable por medio de analizadores predictivos son analizables por métodos LR.

Verdadero. Las gramáticas analizables por métodos LR son un supra conjunto de las analizables por LL.

d) Dado un lenguaje ambiguo, es imposible expresarlo a través de una gramática que no sea ambigua.

Falso. Por ejemplo el lenguaje de la calculadora es ambiguo y podemos expresarlo con una gramática no ambigua.

e) Si es posible describir el patrón de una cadena por medio de una ER, siempre es posible construir un autómata finito deterministico que reconozca dicha cadena

Verdadero. Las ER sirven para representar lenguajes regulares, los cuales son reconocidos por medio de AFD.

2) *Dada la siguiente gramática. (25 pts.)*

```
S:  A a w
  |  B x y
A:  z m
  |  a
B:  A
  |  z
```

- ¿Qué lenguaje genera?

zmaw, aaw, zmxy, axy, zxy.

- ¿Qué tipo de gramática es según la clasificación de Chomsky?

Libre de contexto.

- Verifique si es posible reconocerla mediante LL(1). En caso negativo encuentre una gramática equivalente que si pueda ser reconocida mediante esta técnica.

Regla	First	Follow	Select
S : A a w	{z, a}	{ $ }	{z, a}
S : B x y	{z, a}		{z, a}
A : z m	{z}	{a, x}	{z}
A : a	{a}		{a}
B : A	{z, a}	{x}	{z, a}
B : z	{z}		{z}

Los conjuntos marcados no son disjuntos, por la tanto **NO** es LL1.

Como el lenguaje es finito podemos simplemente encontrar una gramática equivalente enumerando cada una de las cadenas que pertenecen al lenguaje y luego sacamos factor común.

Regla
S : a a w
S : a x y
S : z m x y
S : z m a w
S : z x y

Regla
S : a A
S : z m A
S : z A
A : a w
A : x y

Regla
S : a A
S : z Z
A : a w
A : x y
Z : m A
Z : A

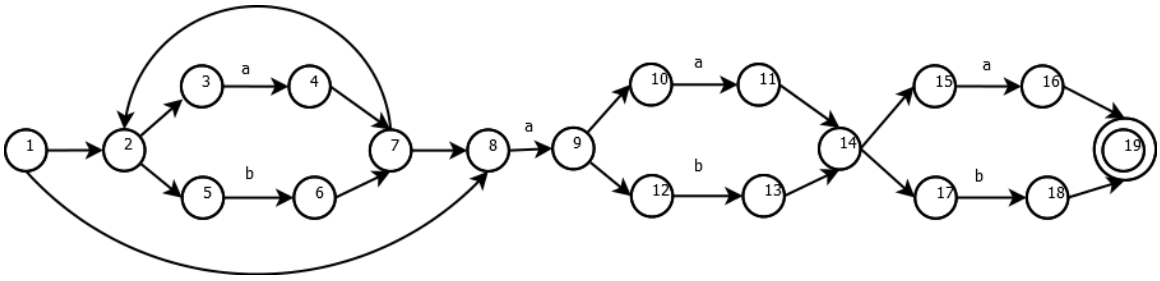
Comprobamos que es LL1.

Regla	First	Follow	Select
S : a A	{a}	{S}	{a}
S : z Z	{z}		{z}
A : a w	{a}	{$}	{a}
A : x y	{x}		{x}
Z : m A	{m}	{$}	{m}
Z : A	{a, x}		{a, x}

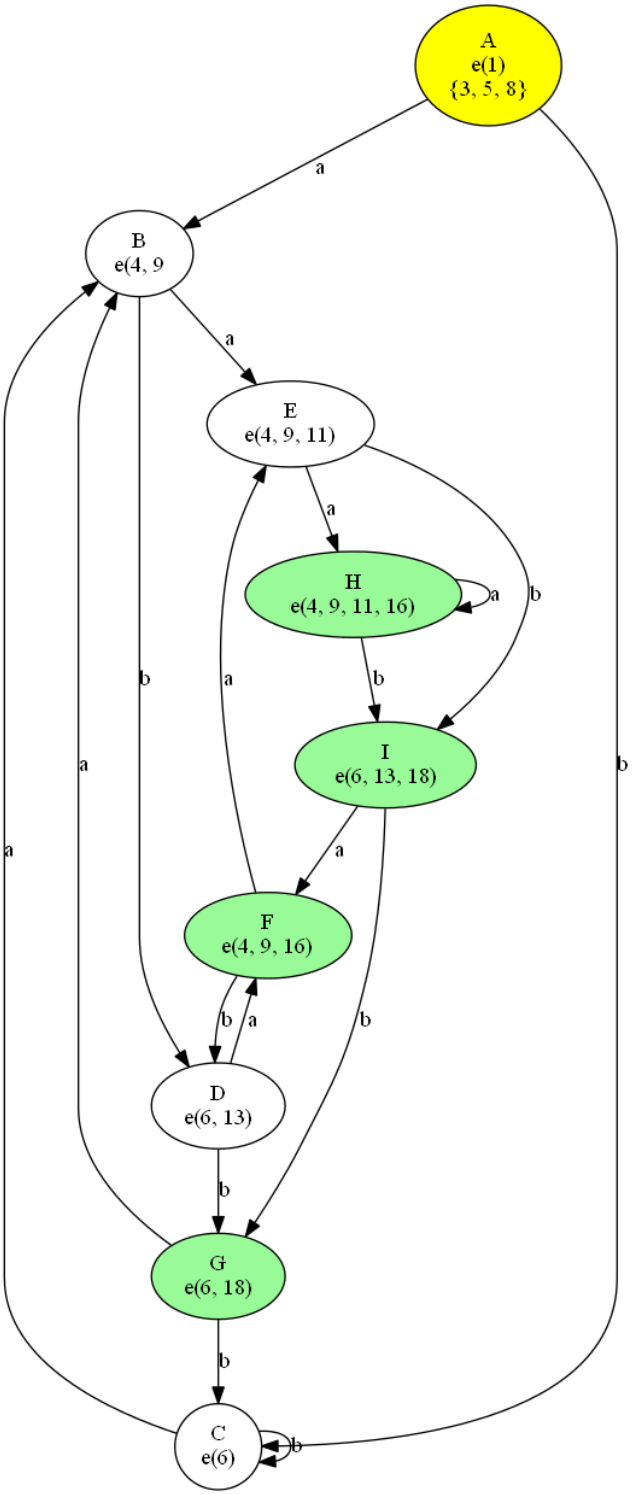
3) Dada la siguiente ER: (25 pts.)

(a | b)* a (a|b) (a|b)

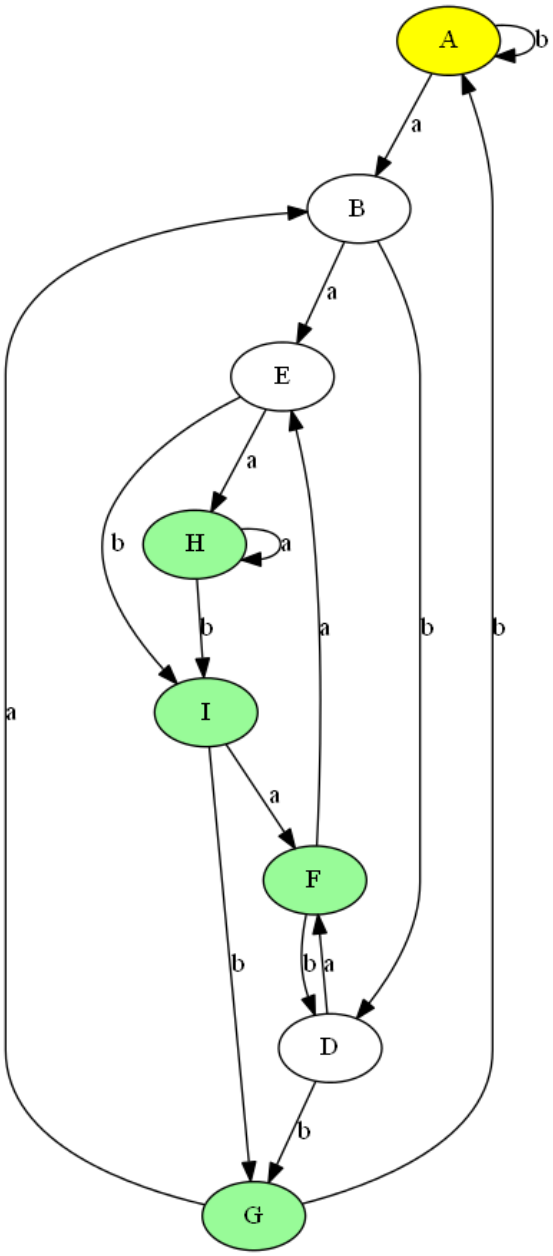
a. Describa el lenguaje que genera
b. Construya los siguientes autómatas:
   i. Thompson



ii. Determinístico



iii. Determinístico de estados mínimos



4) *Construya la tabla del analizador sintáctico ascendente de menor potencia para la siguiente gramática: (25 pts.)*

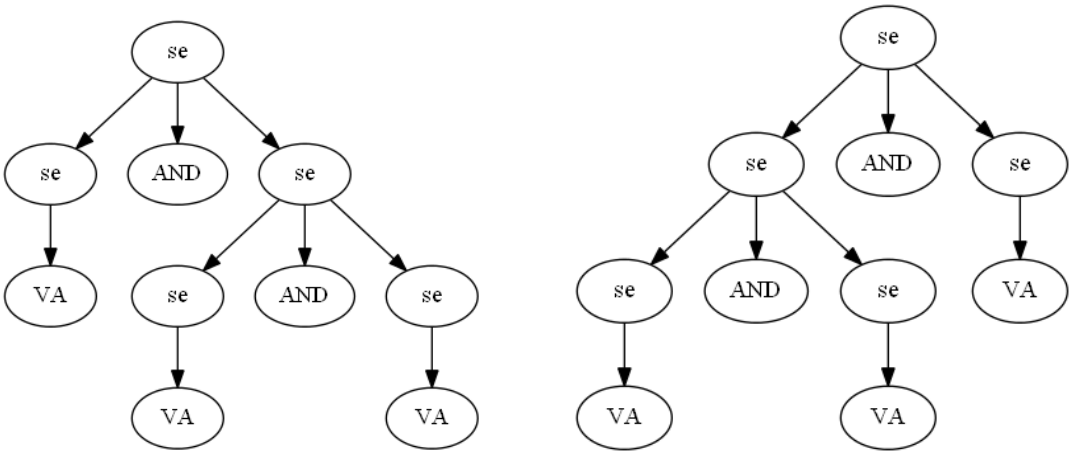
```
se:   VA
    | ID
    | se AND se
    | se OR se
    | NOT se
    | (se)
```

Si la gramática no se alcanza a resolver mediante LR(1) debe especificar qué acciones toma para solventar los problemas y mantener el orden de precedencia de los operadores.

El orden de precedencia es NOT > OR > AND.

La gramática es ambigua, o sea que demostrando esto, demostramos que no hay método LR capaz de resolverla.

Cadena de ejemplo: VA AND VA AND VA



Demostrado esto optamos por hacer SLR como parser de menor potencia, pero vamos a tener que tomar decisiones para resolver los conflictos que aparezcan.

	VA	ID	AND	OR	NOT	(	)	$	se
0	D2	D3			D4	D5			1
1			D6	D7				OK	
2			R1	R1			R1	R1	
3			R2	R2			R2	R2	
4	D2	D3			D4	D5			8
5	D2	D3			D4	D5			9
6	D2	D3			D4	D5			10
7	D2	D3			D4	D5			11
8			R5/D6	R5/D7			R5	R5	
9			D6	D7			D12		
10			R3/D6	R3/D7			R3	R3	
11			R4/D6	R4/D7			R4	R4	
12			R6	R6			R6	R6	

En la tabla se resalta en amarillo las decisiones que necesariamente tenían que hacer para cumplir con la precedencia pedida. En verde se resaltan las 2 opciones faltantes con la opción más eficiente desde el punto de vista del parser.

Pasamos a explicar el porqué de las decisiones.

R3: se -> se AND se

R4: se -> se OR se
R5: se -> NOT se

En el estado 8 tenemos la posibilidad de hacer R5 y como lookahead tenemos AND u OR. Esto quiere decir que en la pila tenemos NOT se. Como NOT tiene el mayor nivel de precedencia tenemos que reducir.

Por ejemplo si la cadena es **NOT se AND se**; la precedencia indica que hay que hacer **(NOT se) AND se**.

En el estado 10 tenemos la posibilidad de reducir por la 3 y como look un OR. Para eso la cadena de entrada sería algo así: **se AND se OR se** y se nos indica que la precedencia es OR > AND; por lo tanto, hay que resolverlo así **se AND (se OR se)**. O sea, tenemos que desplazar para agregar el OR a la pila y que luego se reduzca primero por la 4 antes que por la 3.

En este mismo estado, si el look es AND conviene reducir para achicar la pila.

En el estado 11 tenemos la posibilidad de reducir por la 4 y como look un AND. Para eso la cadena de entrada sería algo así: **se OR se AND se** y se nos indica que la precedencia es OR > AND; por lo tanto, hay que resolverlo así **(se OR se) AND se**. O sea, tenemos que reducir. En este mismo estado, si el look es OR conviene reducir para achicar la pila.