PREGUNTAS TEST COMPILADORES - Gramáticas LL (2° Grado en Informática)

FACULTAD DE INFORMÁTICA

1. El análisis sintáctico LL(1) consiste...
   a) ...en una técnica ascendente dónde se realiza la derivación más a la izquierda.
   b) ...en una técnica descendente dónde se realiza la derivación más a la derecha.
   c) ...en una técnica descendente dónde se realiza la derivación más a la izquierda.

2. Elige la opción correcta:
   a) Una *gramática LL(1)* es *no ambigua*, *λ-libre* y está *factorizada*.
   b) Una *gramática LL(1)* es *no ambigua*, *no recursiva por la izquierda* y está *factorizada*.
   c) Una gramática *no ambigua*, *no recursiva por la izquierda* y *factorizada* es una *gramática LL(1)*.

3. En la gestión de error en modo pánico de un analizador LL(1),
   a) cuando ocurre un error en el cual $M[A, b] = \emptyset$, siendo A el símbolo de la cima de la pila y b el símbolo de la entrada, si $b \in SIGUIENTE(A)$, se desapila A.
   b) cuando ocurre un error en el cual $M[A, b] = \emptyset$, siendo A el símbolo de la cima de la pila y b el símbolo de la entrada, se descarta el símbolo b de la entrada.
   c) cuando ocurre un error en el cual $M[A, b] = \emptyset$, siendo A el símbolo de la cima de la pila y b el símbolo de la entrada, se continúa con el análisis.

4. Podemos afirmar que una gramática es LL(1)
   a) si no es recursiva por la izquierda, ni ambigua ni está factorizada.
   b) si es LR.
   c) sii $\forall A \in V_N$, si existen producciones $A \to \alpha$ y $A \to \gamma$, entonces $predict(A \to \alpha) \cap predict(A \to \gamma) = \emptyset$.

5. Dada la gramática con el siguiente conjunto P de reglas:

$$\begin{array}{lcl} (1) E & \to & id\ E' \\ (2) E' & \to & \lambda \\ (3) & | & \hat{}\ E' \\ (4) & | & .\ id\ E' \\ (5) & | & [\ E\ ]\ E' \end{array}$$

   a) no puede ser LL puesto que $predict(2) \cap predict(4) \neq \emptyset$.
   b) es LL, puesto que $predict(i) \cap predict(j) = \emptyset\ \forall i, j \in P$.
   c) no puede ser LL puesto que no es λ-libre.

6. Indica la respuesta incorrecta:
   a) Dado un lenguaje formal cualquiera, no es posible determinar de forma automática si existe una gramática libre de contexto de tipo LL(1) que lo genere.
   b) Dada una gramática libre de contexto ambigua cualquiera, es posible obtener de forma automática una gramática equivalente que no sea ambigua.
   c) Dada una gramática libre de contexto propia con recursividad izquierda, es posible obtener de forma automática una gramática equivalente que no sea recursiva por la izquierda.

7. Indica la respuesta incorrecta:
   a) No existen métodos generales de análisis sintáctico que puedan trabajar con cualquier tipo de gramática libre de contexto no ambigua.
   b) Si una gramática es LL(1), es posible construir un analizador de una sola pasada.
   c) El orden de complejidad de un analizador LL(1) es O(n).

8. Elige de entre las tres alternativas, una gramática equivalente a G que no sea recursiva por la izquierda, siendo G:

$$\begin{array}{lcl} Lista & \rightarrow & [\,] \mid [\,Termino\,] \\ Termino & \rightarrow & Termino\ ,\ Termino \mid ID \mid Lista \end{array}$$

a)

$$\begin{array}{lcl} Lista & \rightarrow & [\,] \mid [\,Termino\,] \\ Termino & \rightarrow & ID \mid Lista \mid ID\ TerminoF \mid Lista\ TerminoF \\ TerminoF & \rightarrow & ,\ Termino \mid ,\ Termino\ TerminoF \end{array}$$

b)

$$\begin{array}{lcl} Lista & \rightarrow & [\,] \mid [\,Termino\,] \\ Termino & \rightarrow & ID\ ,\ Termino \mid ID \mid Lista \end{array}$$

c)

$$\begin{array}{lcl} Lista & \rightarrow & [\,] \mid [\,Termino\,] \\ Termino & \rightarrow & Termino2\ ,\ Termino \mid ID \mid Lista \\ Termino2 & \rightarrow & ID \mid Lista \mid Termino \end{array}$$

9. El análisis descendente predictivo recursivo:

   a) se realiza construyendo un procedimiento para cada símbolo no terminal de la gramática, encargado de reconocer la cadena derivada a partir de él.

   b) se realiza construyendo un procedimiento para cada símbolo terminal de la gramática, encargado de reconocer la porción de cadena que empareja con él.

   c) se realiza construyendo un procedimiento para cada regla de la gramática, encargado de reconocer la cadena derivada a partir de esa regla.

10. Considérese la siguiente gramática, recursiva por la izquierda:
    $S \rightarrow A\alpha \mid \delta$
    $A \rightarrow S\beta$
    ¿Cuál de las siguientes gramáticas es el resultado de eliminar correctamente la recursividad?

    a) $S \rightarrow A\alpha \mid \delta$
    $A \rightarrow \delta\beta \mid A\alpha\beta$

    b) $S \rightarrow A\alpha \mid \delta$
    $A \rightarrow \delta\beta \mid \delta\beta A'$
    $A' \rightarrow \alpha\beta \mid \alpha\beta A'$

    c) $S \rightarrow \alpha A \mid \delta$
    $A \rightarrow \delta\beta \mid \delta\beta A'$
    $A' \rightarrow \alpha\beta \mid \lambda$

11. Una gramática LL:

    a) Tiene que ser propia, puesto que si no, sería imposible eliminar la recursividad por la izquierda.

    b) Puede no ser propia.

    c) Tiene que ser no recursiva por la izquierda, factorizada y λ-libre.

12. ¿En qué condiciones se puede aplicar el método de análisis descendente recursivo predictivo a una gramática?

    a) En las mismas condiciones que el método LL.

    b) La gramática no debe ser recursiva por la izquierda, pero puede no estar factorizada.

    c) Es suficiente que la gramática no sea ambigua.

13. Si partimos de una gramática libre de contexto y no LL, con un analizador descendente recursivo:

    a) No podría analizarse ninguna sentencia.

    b) Podría analizarse sin problemas cualquier sentencia, aunque sería más ineficiente que si tuviéramos una gramática LL.

    c) Podría entrar en bucles infinitos con algunas sentencias.

14. Sea G la gramática con las producciones:

$S \to 0\ S\ 1 \mid 0\ 1$

Decir cuál de las siguientes es una gramática equivalente a G y LL(1).

a) $S \to 0\ S\ S' \mid 1$
$S' \to 1 \mid \lambda$

b) $S \to 0\ S'\ 1$
$S' \to 0\ S'\ 1 \mid S''$
$S'' \to 0\ 1 \mid \lambda$

c) $S \to 0\ S'$
$S' \to S\ 1 \mid 1$

15. Dada la siguiente gramática:

$E \to TE'$
$E' \to +TE' \mid -TE' \mid \lambda$
$T \to FT'$
$T' \to *FT' \mid /FT' \mid \lambda$
$F \to (E) \mid num$

Si construimos un analizador descendente predictivo recursivo, ¿cuántas veces entrará en el método del no terminal E' con la entrada `2+3*4`?

a) 2

b) 3

c) 0

16. Elige, de entre las opciones, una gramática no recursiva por la izquierda que NO sea equivalente a la siguiente:

$$\begin{array}{lcl} A & \to & (\ L\ ) := (\ R\ ) \\ L & \to & \textbf{id} \\ & \mid & L\ ,\ \textbf{id} \\ R & \to & \textbf{num} \\ & \mid & R\ ,\ \textbf{num} \end{array}$$

a)

$$\begin{array}{lcl} A & \to & (\ L\ ) := (\ R\ ) \\ L & \to & \textbf{id} \\ & \mid & \textbf{id}\ ,\ L \\ R & \to & \textbf{num} \\ & \mid & \textbf{num}\ ,\ R \end{array}$$

b)

$$\begin{array}{lcl} A & \to & (\ L\ ) := (\ R\ ) \\ L & \to & \lambda \\ & \mid & \textbf{id}\ ,\ L \\ R & \to & \lambda \\ & \mid & \textbf{num}\ ,\ R \end{array}$$

c)

$$\begin{array}{l} A \to (\ L\ ) := (\ R\ ) \\ L \to \textbf{id} \mid \textbf{id}\ L' \\ L' \to , \textbf{id} \mid , \textbf{id}\ L' \\ R \to \textbf{num} \mid \textbf{num}\ R' \\ R' \to , \textbf{num} \mid , \textbf{num}\ R' \end{array}$$

17. En la siguiente gramática:

$$
\begin{array}{lcl}
MATRIZ & \rightarrow & (\ FILA\ FILAS\ ) \\
FILA & \rightarrow & num\ NUMEROS\ ; \\
FILAS & \rightarrow & FILA\ FILAS \\
 & | & \lambda \\
NUMEROS & \rightarrow & num\ NUMEROS \\
 & | & \lambda
\end{array}
$$

a) $PRIMERO(FILAS) = \{num, \lambda\}$, $SIGUIENTE(FILAS) = \{), \$\}$ y $predict(4) = \{), \$\}$

b) $PRIMERO(FILAS) = \{num\}$, $SIGUIENTE(FILAS) = \{)\}$ y $predict(4) = \{num\}$

c) $PRIMERO(FILAS) = \{num, \lambda\}$, $SIGUIENTE(FILAS) = \{)\}$ y $predict(4) = \{)\}$

Parte II: PREGUNTAS CORTAS.

1. La siguiente gramática describe sentencias con asignaciones múltiples. ¿Qué modificaciones harías en la gramática para llegar a una equivalente que pudiese ser reconocida con un analizador LL(1)?

$$\begin{array}{lcl} S & \rightarrow & SA \mid A \\ A & \rightarrow & id = L; \\ L & \rightarrow & num \mid id = L \end{array}$$

2. Dado el siguiente programa en C que implementa un analizador, dar la gramática que genera el lenguaje reconocido por dicho analizador:

```
/* Análisis sintáctico */
#include <stdio.h>
char token, cadena[80];
int i=0;
void main(void)
{
    printf("Introduce la cadena a reconocer \n");
    printf("=>");
    scanf("%s",cadena);
    token= cadena[0];
    if (a()) printf("\nCADENA RECONOCIDA");
        else printf("\nCADENA NO RECONOCIDA");
}
/*******************************/
int a(void)
{
if (token== 'x') { i+=1; token= cadena[i]; return(1); }
else if (token== '(')
        if (b())
            {
                if (token== ')') return(1);
                else return(0);
            }
        else return(0);
      else return(0);
}
/*******************************/
int b(void)
{
    i+=1;
    token= cadena[i];
    if (a())
        if (c()) return(1);
        else return(0);
    else return(0);
}
/*******************************/
int c(void)
{
    while (token== '+')
    {
        i+=1;
        token= cadena[i];
        if (!a()) return(0);
    }
    return(1);
}
```

Dada cualquier gramática G, ¿existe alguna propiedad que pueda cumplir dicha gramática que haga imposible la implementación de un analizador descendente recursivo para reconocer L(G)? Justifica la respuesta.

3. Dada la siguiente tabla LL(1):

	id	^	.	[	]	$
E	$E \rightarrow id\ E'$					
E'		$E' \rightarrow \hat{}\ E'$	$E' \rightarrow .\ id\ E'$	$E' \rightarrow [\ E\ ]\ E'$	$E' \rightarrow \lambda$	$E' \rightarrow \lambda$

asociada a la gramática:

$$\begin{array}{lcl} E & \rightarrow & id\ E' \\ E' & \rightarrow & \lambda \\ & \mid & \hat{}\ E' \\ & \mid & .\ id\ E' \\ & \mid & [\ E\ ]\ E' \end{array}$$

a) Explicar por qué aparece la regla $E' \rightarrow \lambda$ en las casillas $\{E',]\}$ y $\{E', \$\}$

b) Rellenar la casilla $\{E,]\}$ con una llamada a una rutina de recuperación de errores *a nivel de frase*.

c) Proceder al reconocimiento o rechazo de la entrada $id\hat{}.id[\ ]\$$ usando recuperación de errores a nivel de frase si fuese necesario. En caso de no haber completado el apartado anterior, resolver el apartado empleando recuperación de errores en modo pánico.

Parte III: PROBLEMAS.

1. Supongamos que se desea formalizar un lenguaje de matrices numéricas en el que, por ejemplo, una matriz de dos filas y tres columnas como

$$\begin{pmatrix} -1 & 3 & 7 \\ 4 & 6 & 0 \end{pmatrix}$$

quedaría representada de la siguiente forma:

$$(\ -1\ 3\ 7\ ;\ 4\ 6\ 0\ ;\ )$$

La siguiente gramática, G, con $V_T = \{(,), num, ;\}$, $V_N = \{MATRIZ, FILA, FILAS, NUMEROS\}$, símbolo inicial $MATRIZ$ y el siguiente conjunto P de producciones:

$$\begin{array}{lcl} MATRIZ & \rightarrow & (\ FILA\ FILAS\ ) \\ FILA & \rightarrow & num\ NUMEROS\ ; \\ FILAS & \rightarrow & FILA\ FILAS \\ & | & \lambda \\ NUMEROS & \rightarrow & num\ NUMEROS \\ & | & \lambda \end{array}$$

sirve para generar matrices con el formato anterior.

Realiza los siguientes ejercicios:

a) Comprueba si se trata de una gramática LL(1) calculando los conjuntos PRIMERO y SIGUIENTE para cada símbolo no terminal, los conjuntos *predict* para cada regla y la tabla de análisis.

b) Simular el comportamiento del algoritmo de análisis LL para la cadena w ≡ (num num ; ;), realizando la recuperación de errores en modo pánico en caso de error.

2. Dada la gramática G, con $V_T = \{id, =, ;\}$, $V_N = \{S, A, L\}$, símbolo inicial S y el siguiente conjunto P de producciones:

$$\begin{array}{lcl} S & \rightarrow & S\ A \\ & | & A \\ A & \rightarrow & id\ =\ L\ ; \\ L & \rightarrow & id \\ & | & L\ =\ L \end{array}$$

responder a las siguientes cuestiones:

a) Decir, justificando las respuestas, y sin construir ninguna tabla de análisis, si G es:

1) Propia.
2) Ambigua.
3) LL(1).

En caso de que G no cumpla alguna de estas propiedades, dar todas las razones que se conozcan para justificarlo.

b) Construir los conjuntos *PRIMERO* y *SIGUIENTE* para cada no terminal, y *predict* para cada regla. Construir la tabla LL.

3. La siguiente gramática G, con $V_T = \{(,), :=, \mathbf{id}, \mathbf{num}, ,\}$, $V_N = \{, L, R\}$, símbolo inicial A y el siguiente conjunto P de producciones:

$$\begin{array}{lcl} A & \rightarrow & (\ L\ ) := (\ R\ ) \\ L & \rightarrow & \mathbf{id} \\ & | & L\ ,\ \mathbf{id} \\ R & \rightarrow & \mathbf{num} \\ & | & R\ ,\ \mathbf{num} \end{array}$$

permite definir asignaciones múltiples (es una característica de algunos lenguajes de programación, como Perl). Por ejemplo, la siguiente sentencia:

```
(a, b) := (1, 2)
```

asigna simultáneamente a la variable *a* el valor *1*, y a la variable *b* el valor *2*. Responder a las siguientes cuestiones:

a) Decir, justificando la respuesta, y sin construir ninguna tabla de análisis, si G puede ser LL(1). En caso de que no lo sean, realizar las transformaciones necesarias para llegar a una gramática equivalente que pueda serlo.

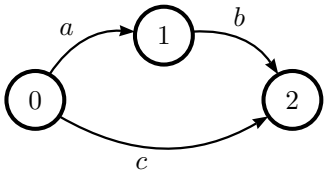
b) Verificar si G o la gramática equivalente obtenida en el apartado anterior es LL(1), calculando los conjuntos PRIMERO y SIGUIENTE, los conjuntos *predict*, y la tabla de análisis.

c) Simular el comportamiento del algoritmo de análisis LL(1), usando la tabla obtenida en el apartado anterior, para la cadena $w \equiv (a, b)(1, 2)$, realizando la recuperación de errores en modo pánico en caso de error.

4. La siguiente gramática G, con $V_T = \{\textbf{id}\ ,\ \textbf{num}\ ,\ \&\ ,\ ;\ ,\ :\ ,\ >\}$, $V_N = \{A, S, D, L, B\}$, símbolo inicial A y el siguiente conjunto P de producciones:

$$\begin{array}{lcl} A & \rightarrow & A\ S \\ & | & S \\ S & \rightarrow & \textbf{num}\ :\ D\ ; \\ D & \rightarrow & L \\ & | & \lambda \\ L & \rightarrow & L\ \&\ B \\ & | & B \\ B & \rightarrow & \textbf{id} > \textbf{num} \end{array}$$

permite representar autómatas finitos textualmente. Por ejemplo, el siguiente autómata:



se representa de la siguiente forma:

```
0 : a > 1 & c > 2 ;
1 : b > 2 ;
2 : ;
```

Responder a las siguientes cuestiones:

a) Decir, justificando la respuesta, y sin construir ninguna tabla de análisis, si G es LL(1). En caso de que no lo sea, realizar las transformaciones necesarias en la gramática que puedan conducir a que lo sea.

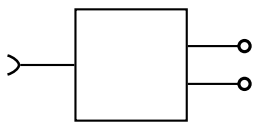
5. La siguiente gramática G, con $V_T = \{\&\ ,\ <\ ,\ >\ ,\ ;\ ,\ (\ ,\ )\ ,\ \textbf{int}\}$, $V_N = \{P, S, E, I, O\}$, símbolo inicial P y el siguiente conjunto de producciones:

$$\begin{array}{lcl} P & \rightarrow & S \\ & | & S\ \&\ P \\ S & \rightarrow & E \\ & | & E\ S \\ E & \rightarrow & <\ I\ ;\ O\ > \\ & | & (\ P\ ) \\ I & \rightarrow & \textbf{int} \\ O & \rightarrow & \textbf{int} \end{array}$$

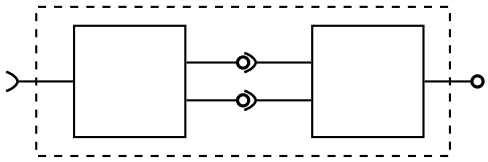
permite representar grafos de componentes agrupados de forma serie o paralela. Un componente se describe mediante un bloque que posee interfaces de entrada y salida. Textualmente, un componente básico se representa mediante una cadena de la forma indicada por la primera alternativa de E:

```
< n  interfaces entrada ; n  interfaces de salida >
```

Por ejemplo, un componente `<1;2>` representa al bloque de la siguiente figura:

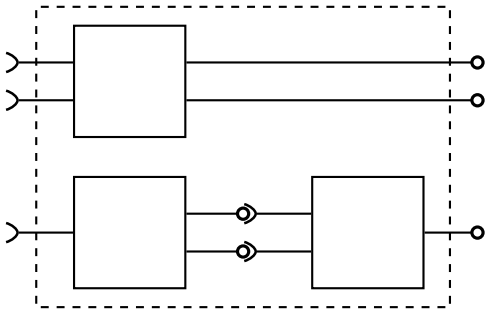


Se pueden conectar en serie dos o más componentes. Para ello, basta con concatenarlos uno seguido de otro, empleando las opciones del no terminal S. Por ejemplo, el componente anterior se puede conectar con otro que tenga dos interfaces de entrada. La cadena `<1;2><2;1>` representa el siguiente grafo:



La agrupación serie de dos componentes da lugar a un nuevo componente con tantos interfaces de entrada como el primer componente, y tantos interfaces de salida como el último componente.

También es posible agrupar en paralelo dos o más componentes. Para ello es necesario concatenarlos empleando las opciones del no terminal P. Por ejemplo, la agrupación serie anterior se puede disponer en paralelo con un nuevo componente. La cadena `<2;2>&<1;2><2;1>` representa el siguiente grafo:



La agrupación en paralelo de dos componentes da lugar a un nuevo componente con tantos interfaces de entrada como la suma de interfaces de entrada de los componentes agrupados, y tantos interfaces de salida como la suma de interfaces de salida de los componentes agrupados.

Responder a las siguientes cuestiones:

a) Decir, justificando la respuesta, y sin construir ninguna tabla de análisis, si G es LL(1). En caso de que no lo sea, realizar las transformaciones necesarias en la gramática que puedan conducir a que lo sea.

b) A partir de la gramática obtenida en el apartado anterior, construir la tabla LL(1). Indicar si G es una gramática LL(1) justificando la respuesta.

c) Simular el reconocimiento descendente de la cadena errónea `<1;>` empleando el modo pánico cuando se detecte el error.

d) Analizando la gramática de partida, indicar justificadamente qué operación tiene mayor prioridad, la conexión en serie o la conexión en paralelo. Indicar igualmente el tipo de asociatividad de ambas operaciones.

6. Considerar la siguiente gramática:

$$\begin{array}{lcl} S & \rightarrow & LP\ ;\ E \\ LP & \rightarrow & LP\ ;\ num \\ & | & num \\ E & \rightarrow & num \\ & | & var \\ & | & fun\ (\ E\ ) \end{array}$$

que genera frases consistentes en una lista de puntos (LP) y una expresión matemática sencilla en la que se emplean únicamente funciones `seno` y `coseno`. Por ejemplo, la gramática podría generar la siguiente cadena:

$$0{,}1; 0{,}2; 0{,}3; seno(seno(x))$$

a) Decir si se trata de una gramática LL(1), justificando la respuesta.

b) Calcular los conjuntos PRIMERO y SIGUIENTE.

7. Considerar la siguiente gramática:

$$\begin{array}{lcl} P & \rightarrow & [\ L\ ]\ (\ num\ ) \\ L & \rightarrow & E \\ & | & E\ ,\ L \\ E & \rightarrow & num \\ & | & P \end{array}$$

que genera un lenguaje para evaluación de polinomios. Por ejemplo, la gramática podría generar la siguiente cadena:

$$[1, 3, [2, 1](2)](1)$$

donde los números de la lista entre corchetes representarían los coeficientes de un polinomio de una variable, y el número entre paréntesis, el valor que se quiere asignar a la variable para evaluar el polinomio. Por ejemplo, la frase $[1, 2, 6](5)$ representaría al polinomio $x^2 + 2x + 6$, que deberá evaluarse asignando a x el valor 5.

Decir si se trata de una gramática LL(1), justificando la respuesta. Si no se tratara de una gramática LL(1), intentar transformarla para conseguir que lo sea. Calcular los conjuntos PRIMERO y SIGUIENTE para cada no terminal y *predict* para cada regla. Razonar con estos últimos conjuntos si la gramática es LL(1), sin construir la tabla de análisis.

8. Considerar la siguiente gramática G:

$$\begin{array}{lcl} P & \rightarrow & [\ L\ ]\ (\ num\ ) \\ L & \rightarrow & E \\ & | & L\ ,\ E \\ E & \rightarrow & num \\ & | & P \end{array}$$

que genera un lenguaje para evaluación de polinomios. Por ejemplo, la gramática podría generar la siguiente cadena:

$$[1, 3, [2, 1](2)](1)$$

donde los números de la lista entre corchetes representarían los coeficientes de un polinomio de una variable, y el número entre paréntesis, el valor que se quiere asignar a la variable para evaluar el polinomio. Por ejemplo, la frase $[1, 2, 6](5)$ representaría al polinomio $x^2 + 2x + 6$, que deberá evaluarse asignando a x el valor 5.

Decir si se trata de una gramática LL(1), justificando la respuesta. Si no se tratara de una gramática LL(1), intentar transformarla para conseguir que lo sea. Calcular los conjuntos PRIMERO y SIGUIENTE para cada no terminal y *predict* para cada regla. Construir la tabla de análisis. Decir si la nueva gramática es LL(1).

9. En el sistema formal denominado λ-*Cálculo* se definen los *números de Church* de la siguiente forma:

$$\begin{array}{l} \underline{0} \equiv \lambda f.(\lambda x.x) \\ \underline{1} \equiv \lambda f.(\lambda x.(fx)) \\ \underline{2} \equiv \lambda f.(\lambda x.(f(fx))) \\ \underline{3} \equiv \lambda f.(\lambda x.(f(f(fx)))) \\ \ldots \end{array}$$

Considerar la siguiente gramática G para generar *números de Church*, con $V_T = \{\lambda, ., (,), f, x\}$, $V_N = \{N, F, X, C\}$, N el símbolo inicial, y P el siguiente conjunto de producciones[1]:

$$\begin{array}{lcl} N & \rightarrow & F\ .\ (\ X\ .\ C\ ) \\ F & \rightarrow & \lambda\ f \\ X & \rightarrow & \lambda\ x \\ C & \rightarrow & (\ f\ C\ ) \\ & | & x \end{array}$$

a) Decir si se trata de una gramática LL(1), justificando la respuesta. Si no se tratara de una gramática LL(1), intentar transformarla para conseguir que lo sea. Calcular los conjuntos PRIMERO y SIGUIENTE para cada no terminal y *predict* para cada regla. Construir la tabla de análisis.

b) Simular el algoritmo DPNR usando la tabla anterior, para la cadena de entrada $w \equiv \lambda f(x.x)$, aplicando el método de *recuperación de errores en modo pánico*, en caso de error.

[1] Observar que en esta gramática, tanto el símbolo λ como el símbolo . forman parte de los tokens del lenguaje. Por tanto, no debéis confundir en este caso el símbolo λ con la cadena vacía.