Universidad de Huelva

Departamento de Tecnologías de la Información

Área de Ciencias de la Computación e Inteligencia Artificial

Procesadores de lenguajes

Examen de junio

EJERCICIO 1 (2 puntos)

Considere la gestión de la memoria pila en tiempo de ejecución.

(a) Describa la estructura del registro de activación de una función.

(b) Describa el proceso de llamada a una función.

(c) Describa el proceso de retorno de una función.

EJERCICIO 2 (2 puntos)

La siguiente figura muestra una expresión regular formada por los símbolos ***b***, ***a*** y ***o***.

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

Obtenga el Autómata Finito Determinista asociado, indicando el conjunto de expresiones regulares punteadas que describen cada estado del autómata.

EJERCICIO 3 (2 puntos)

La siguiente gramática permite describir una lista de definiciones:

ListaDefiniciones → *ListaDefiniciones Definición*

ListaDefiniciones → *Definición*

Definición → **symbol eq** *ListaReglas* **semicolon**

ListaReglas → *Regla*

ListaReglas → *ListaReglas* **bar** *Regla*

Regla → λ

Regla → *Regla Elemento*

Elemento → **symbol**

Elemento → **token**

(a) Construya la tabla SLR de la gramática planteada. Utilice para ello la tabla incluida en la última página.

(b) Desarrolle la traza del analizador SLR para la siguiente cadena:

symbol eq token bar token symbol semicolon symbol eq semicolon

EJERCICIO 4 (2 puntos)

La siguiente gramática permite describir un circuito formado por resistencias unidas en serie o en paralelo en el formato de la herramienta JavaCC. Modifique esta descripción para que el símbolo *Circuito()* devuelva un objeto de tipo *Nodo*.

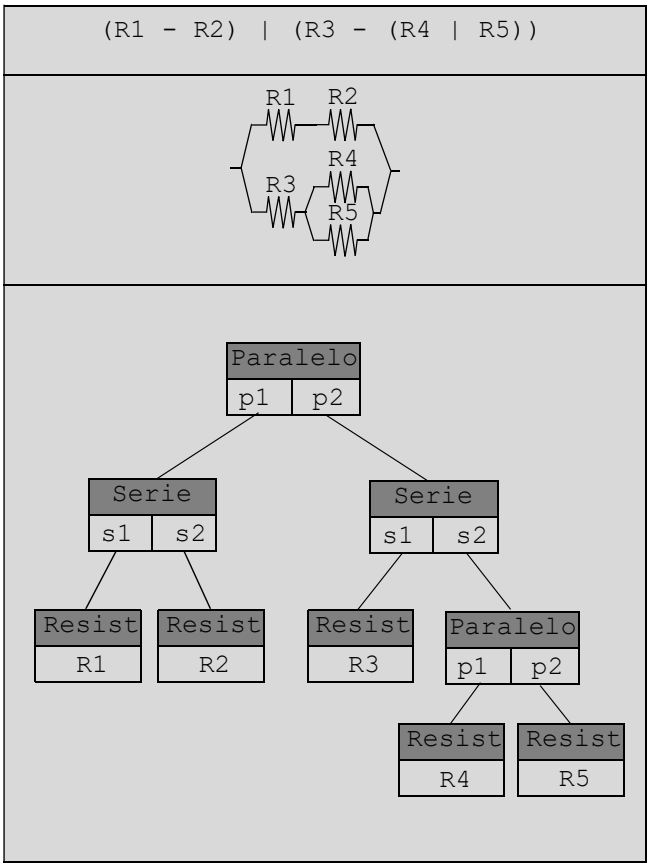
```
void Circuito() :
{}
{
  CircuitoSerie()  ( <BARRA>  CircuitoSerie() )*
}

void CircuitoSerie() :
{}
{
  CircuitoBase()  ( <GUION> CircuitoBase() )*
}

void CircuitoBase() :
{}
{
    <RESISTENCIA>
  | <PARAB> Circuito() <PARCE>
}
```

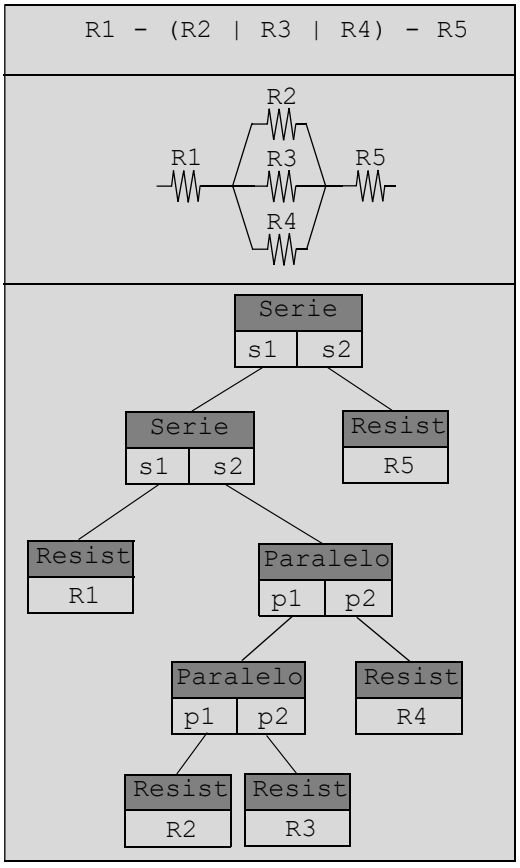
Considere que las siguientes clases están definidas:

```
public class Nodo {
}

public class Resistencia extends Nodo {
  public String R;
}

public class Serie extends Nodo {
  public Nodo s1;
  public Nodo s2;
}

public class Paralelo extends Nodo {
  public Nodo p1;
  public Nodo p2;
}
```

Desarrolle un ETDS que genere un árbol basado en estas estructuras a partir de una descripción en modo texto de un circuito.

A continuación se muestran dos ejemplos de circuitos, su representación textual y el árbol que debe generar el ETDS.





EJERCICIO 5 (2 puntos)

Considere la siguiente sintaxis LL(1) para la instrucción de asignación:

Asignación → **id** "=" *Expresion* "**;**"

Expresión → *Término ListaTérminos*

ListaTérminos → "+" *Término ListaTérminos*

ListaTérminos → "**-**" *Término ListaTérminos*

ListaTérminos → λ

Término → *Factor ListaFactores*

ListaFactores → "*****" *Factor ListaFactores*

ListaFactores → "/" *Factor ListaFactores*

ListaFactores → "**%**" *Factor ListaFactores*

ListaFactores → λ

Factor → **num**

Factor → **id**

Factor → "**(**" *Expresión* "**)**"

(a) Calcule los conjuntos Primeros, Siguientes y de Predicción de la gramática.

(b) Escriba un ETDS que genere el código intermedio asociado a las instrucciones de asignación.

NOTA: Considere que se dispone de la función *getNewTemp()*, que devuelve la referencia a una nueva variable temporal.

NOTA: El código intermedio a generar estará formado por las siguientes instrucciones de código de 3 direcciones:

```
var1 = constante
var1 = var2
var1 = - var2
var1 = var2 + var3
var1 = var2 - var3
var1 = var2 * var3
var1 = var2 / var3
var1 = var2 % var3
var1 = MEM[var2 + var3]
if var1 == var2 goto etiqueta
if var1 != var2 goto etiqueta
if var1 > var2 goto etiqueta
if var1 < var2 goto etiqueta
if var1 >= var2 goto etiqueta
if var1 <= var2 goto etiqueta
goto etiqueta
```

Estado	symbol	token	eq	bar	semicolon	$	LiDef	Def	LiReg	Reg	Elem