Departamento de Tecnologías de la Información

Área de Ciencias de la Computación e Inteligencia Artificial

Procesadores de lenguajes

Examen de septiembre

EJERCICIO 1 (2 puntos)

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

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

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

EJERCICIO 2 (2.5 puntos)

La siguiente gramática permite describir una instrucción de asignación:

Asignación → **id eq** *Expresion* **semicolon**

Expresión → *Término*

Expresión → *Expresión* **plus** *Término*

Expresión → *Expresión* **minus** *Término*

Término → *Factor*

Término → *Término* **prod** *Factor*

Término → *Término* **div** *Factor*

Término → *Término* **mod** *Factor*

Factor → **num**

Factor → **id**

Factor → **lparen** *Expresión* **rparen**

(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:

id eq lparen id plus id rparen div num semicolon

EJERCICIO 3 (3 puntos)

La siguiente gramática describe expresiones formadas con las operaciones producto y potencia de números:

Expr → *Factor Producto*

Producto → **prod** *Factor Producto*

Producto → **lambda**

Factor → *Base Potencia*

Potencia → **power** *Base Potencia*

Potencia → **lambda**

Base → **num**

Base → **lparen** *Expr* **rparen**

Considere las siguientes clases que permiten definir expresiones con estos operadores:

```
// Clase abstracta que describe una expresión aritmética
public abstract class  Expression {
}

// Clase que describe un número constante
public class Number extends Expression {
  private double value;

  public Number(String val) {  this.value = Double.parseDouble(val); }
}

// Clase que describe la potencia entre dos expresiones
public class Power extends Expression {
  public Expression base;
  public Expression pow;

  public Power(Expression a, Expression b) {  this.base = a;   this.pow = b;  }
}

// Clase que describe el producto entre dos expresiones
public class Product extends Expression {
  public Expression left;
  public Expression right;

  public Product(Expression a, Expression b) {  this.left = a; this.right = b;  }
}
```

Desarrolle un ETDS que genere el árbol de sintaxis abstracta asociado a las expresiones formadas por productos y potencias de números.

IMPORTANTE: El operador potencia es asociativo a la derecha, mientras que el operador producto es asociativo a la izquierda.

EJERCICIO 4 (2.5 puntos)

Se pretende construir un ETDS que permita generar el código asociado a las expresiones condicionales por medio de la herramienta JavaCC. A continuación se muestra la sintaxis de las expresiones condicionales en el formato de esta herramienta:

```
void Condicion() :
{}
{
  CondicionAnd()  (  <OR>  CondicionAnd()  ) *
}

void CondicionAnd() :
{}
{
  CondicionSimple()  ( <AND>  CondicionSimple()  )*
}

void CondicionSimple() :
{}
{
  Expresion()  (  <EQ>  |  <NE>  |  <GT>  |  <GE> |  <LT>  | <LE> )  Expresion()
 |  <PARAB>  Condicion()  <PARCE>
}
```

A tal fin, la definición de los símbolos anteriores debe modificarse para devolver objetos de la clase *Condition* y para aceptar como atributos heredados los parámetros *label_true* y *label_false* (ambos de tipo *String*). Se asume que el símbolo *Expresion* devuelve un objeto de tipo *Expression* y que se dispone de los métodos *getNewTemp()* y *getNewLabel()*. La descripción de las clases *Condition* y *Expression* es la siguiente:

```
public class Condition {
  public String code;
  public String label_true;
  public String label_false;

  public Condition() { }
}

public class Expression {
  public String code;
  public String temp;

  public Expression() { }
}
```

Estado	id	num	eq	plus	minus	prod	div	mod	lparen	rparen	semicolon	$	Asig	Expr	Term	Factor