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 en tiempo de ejecución.

(a) ¿En que consiste la memoria de pila?

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

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

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

EJERCICIO 2 (2 puntos)

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

l l ((g | λ) (o | l))* g g

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

EJERCICIO 3 (2 puntos)

Considere la siguiente gramática, que describe las expresiones lógicas basadas en los operadores **and**, **or** y **not**:

Expr → *Expr* **or** *Term*

Expr → *Term*

Term → *Term* **and** *Comp*

Term → *Comp*

Comp → **not** *Base*

Comp → *Base*

Base → **id**

Base → **(** *Expr* **)**

Construya el autómata reconocedor de prefijos viables y la tabla de análisis SLR de la gramática planteada. Utilice para ello la tabla incluida en la última página.

EJERCICIO 4 (2 puntos)

La siguiente gramática describe en el formato de JavaCC expresiones formadas con las operaciones producto y potencia de números:

```
void Expresion() :
{}
{
  Factor() ( <PROD> Factor() )*
}

void Factor() :
{}
{
  Base() ( <ELEV> Base() )*
}

void Base() :
{}
{
  <NUM>
  | <PARAB> Expresion() <PARCE>
}
```

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;  }
}
```

Modifique la descripción JavaCC para que el símbolo *Expresion()* devuelva la representación de una expresión formada por los operadores producto y potencia. Es importante tener en cuenta que la potencia es un operador asociativo a la derecha, mientras que el producto es un operador asociativo a la izquierda.

EJERCICIO 5 (2 puntos)

Considere la siguiente sintaxis LL(1) para las expresiones condicionales:

Condición → *CondiciónAnd Disyunción*

Disyuncion → "||" *CondiciónAnd Disyunción*

Disyunción → λ

CondiciónAnd → *CondiciónBase Conjunción*

Conjunción → "**&&**" *CondiciónBase Conjunción*

Conjunción → λ

CondiciónBase → *Expresión Operador Expresión*

CondiciónBase → "(" *Condición* ")"

Operador → "= ="

Operador → "**!**="

Operador → ">"

Operador → "<"

Operador → ">="

Operador → "<="

Genere un ETDS que permita obtener el código intermedio asociado a una condición teniendo en cuenta las siguientes consideraciones:

(a) Se dispone del método *getNewLabel()* para generar nuevas etiquetas.

(b) El símbolo *Expresión* tiene asociados dos atributos sintetizados: *code*, con el código intermedio que genera el valor de la expresión; y *temp*, con la referencia a la variable donde se almacena el valor de la expresión.

(c) El símbolo *Condición* debe tener asociados los siguientes atributos: *code*, atributo sintetizado con el código intermedio asociado a la condición; *label_true*, atributo sintetizado con el nombre de la etiqueta a la que debe saltar el código de la condición en caso de que ésta sea cierta; *label_false*, atributo sintetizado con el nombre de la etiqueta a la que debe saltar el código de la condición en caso de que ésta sea falsa.

Estado	or	and	not	id	(	)	$	Expr	Term	Comp	Base