Universidad de Huelva

Procesadores de lenguaje

Ejercicios del Tema 3

ANÁLISIS DESCENDENTE

Ejercicio 3.1

Compruébese que la siguiente gramática es LL(1) sin modificarla.

$A \rightarrow B\ C\ D$
$B \rightarrow$ **a** C **b**
$B \rightarrow \lambda$
$C \rightarrow$ **c** A **d**
$C \rightarrow$ **e** B **f**
$C \rightarrow$ **g** D **h**
$C \rightarrow \lambda$
$D \rightarrow$ **i**

Ejercicio 3.2

¿ Es LL(1) la siguiente gramática ?

$A \rightarrow B\ C\ D$
$B \rightarrow$ **b**
$B \rightarrow \lambda$
$C \rightarrow$ **c**
$C \rightarrow \lambda$
$D \rightarrow$ **d**
$D \rightarrow \lambda$

Ejercicio 3.3

Dada la siguiente gramática, elimínese la recursividad a la izquierda y los factores comunes por la izquierda y compruébese que la gramática resultante cumple la condición LL(1).

$S \rightarrow S$ **inst**
$S \rightarrow S$ **var** D
$S \rightarrow \lambda$
$D \rightarrow D$ **ident** E
$D \rightarrow D$ **ident sep**
$D \rightarrow$ **int**
$D \rightarrow$ **float**
$E \rightarrow S$ **fproc**

Ejercicio 3.4

Dada la siguiente gramática:

$S \rightarrow A\ B$
$A \rightarrow$ **begin** S **end** B **theend**
$A \rightarrow \lambda$
$B \rightarrow$ **var** L **: tipo**
$B \rightarrow B$ **fvar**
$B \rightarrow \lambda$
$L \rightarrow L$ **,** **id**
$L \rightarrow$ **id**

a) Háganse las transformaciones necesarias para eliminar la recursividad por la izquierda.

b) Calcúlense los conjuntos de primeros y siguientes de cada no terminal.

c) Compruébese que la gramática modificada cumple la condición LL(1).

d) Constrúyase la tabla de análisis sintáctico LL(1) para esta nueva gramática.

e) Finalmente, hágase la traza del análisis de la siguiente cadena, comprobando que las derivaciones son correctas mediante la construcción del árbol de análisis sintáctico.

```
begin
  var id,id: tipo
  var id:tipo
  fvar
end
  var id: tipo
theend
```

Ejercicio 3.5

Dada la siguiente gramática:

$S \rightarrow S$ **inst**
$S \rightarrow T\ R\ V$
$T \rightarrow$ **tipo**
$T \rightarrow \lambda$
$R \rightarrow$ **blq** V **fblq**
$R \rightarrow \lambda$
$V \rightarrow$ **id** S **fin**
$V \rightarrow$ **id** **;**
$V \rightarrow \lambda$

Constrúyase un analizador descendente recursivo (ASDR) para la gramática LL(1) equivalente.

Ejercicio 3.6

Dada la siguiente gramática:

$E \rightarrow$ **[** L
$E \rightarrow$ **a**
$L \rightarrow E\ Q$
$Q \rightarrow$ **,** L
$Q \rightarrow$ **]**

a) Constrúyase un analizador descendente recursivo (ASDR) para la gramática LL(1) equivalente.

b) Escríbase la secuencia de llamadas recursivas a las funciones del ASDR que se realizarían durante la ejecución para la cadena de entrada "**[a , a]**".

Ejercicio 3.7

Dada la siguiente gramática:

$P \rightarrow D\ S$
$D \rightarrow D\ V$
$D \rightarrow \lambda$
$S \rightarrow S\ I$
$S \rightarrow \lambda$
$V \rightarrow$ **decl id ;**
$V \rightarrow$ **decl id (** P **) ;**
$V \rightarrow$ **decl [** D **] id ;**
$I \rightarrow$ **id ;**
$I \rightarrow$ **begin** P **end**

a) Háganse las transformaciones necesarias para que cumpla la condición LL(1).

b) Constrúyase la tabla de análisis sintáctico LL(1) para esa nueva gramática.

c) Hágase la traza de las cadenas:

decl id (begin id ;)
decl id (decl [decl id ;] id ;) ; id ;

Ejercicio 3.8

Considere la siguiente gramática descrita en notación EBNF:

$$\langle E\rangle \rightarrow \langle O\rangle(\text{barra } \langle O\rangle)^*$$
$$\langle O\rangle \rightarrow \langle O\rangle\langle I\rangle \mid \langle I\rangle$$
$$\langle I\rangle \rightarrow \text{sim} \mid \text{parab } \langle E\rangle \text{ parce } \langle C\rangle$$
$$\langle C\rangle \rightarrow (\text{ast} \mid \text{sum} \mid \text{int} \mid \lambda)$$

(a) Realizar las transformaciones necesarias para expresarla en notación BNF.

(b) Realizar las transformaciones necesarias para convertirla en una gramática LL(1).

(c) Calcular los conjuntos Primeros, Siguientes y Predicción de la gramática obtenida.

(d) Describir el analizador descendente recursivo de la gramática obtenida (en Java, C o pseudocódigo)

Ejercicio 3.9

La siguiente gramática permite describir un circuito formado por resistencias unidas en serie o en paralelo. Calcule los conjuntos Primeros, Siguientes y Predicción para las reglas y símbolos de la gramática.

Circuito → *CircuitoSerie RamaParalela*
RamaParalela → "|" *CircuitoSerie RamaParalela*
RamaParalela → λ
CircuitoSerie → *CircuitoBase ConexionSerie*
ConexionSerie → "**-**" *CircuitoBase ConexionSerie*
ConexionSerie → λ
CircuitoBase → **resistencia**
CircuitoBase → "**(**" *Circuito* "**)**"

Ejercicio 3.10

La siguiente gramática, expresada en notación EBNF, permite describir un texto formado por un único párrafo.

Párrafo → (*Frase*)+ **FinDeLínea**

Frase → *Claúsula* (**coma** *Claúsula*)* **punto**

Claúsula → *Palabra* (**espacio** *Palabra*)*

Palabra → (**letra**)+

(a) Realize las transformaciones necesarias para expresarla en notación BNF.

(b) Calcule los conjuntos Primeros, Siguientes y de Predicción para las reglas y símbolos de la gramática resultante.

Ejercicio 3.11

La siguiente gramática, expresada en notación EBNF, permite describir la estructura de una clase.

Clase → **class id llaveab** (Campo | Constructor | Metodo)* **llavece**

Campo → *Tipo* **id pyc**

Constructor → **id parab** (*Tipo* (**coma** *Tipo*)*)? **parce pyc**

Metodo → *Tipo* **id parab** (*Tipo* (**coma** *Tipo*)*)? **parce pyc**

Tipo → **int** | **float** | **char**

(a) Realice las transformaciones necesarias para expresarla en notación BNF.

(b) Realice las transformaciones necesarias para que cumpla la propiedad LL(1).

(c) Calcule los conjuntos Primeros, Siguientes y de Predicción para las reglas y símbolos de la gramática resultante.

Ejercicio 3.12

La siguiente gramática, expresada en notación EBNF, representa la sintaxis de la definición de tipos en un lenguaje funcional:

Type → **type** **id** **eq** *Decl* **semicolon**

Decl → (*BaseDecl* | *Decl* **rel** *BaseDecl*)

BaseDecl → (**id** | *List* | *Tuple*)

List → **lbra** *Decl* **rbra**

Tuple → **lpar** *Decl* (**comma** *Decl*)* **rpar**

(a) Realice las transformaciones necesarias para expresarla en notación BNF.

(b) Realice las transformaciones necesarias para que cumpla la propiedad LL(1).

(c) Calcule los conjuntos Primeros, Siguientes y de Predicción para las reglas y símbolos de la gramática resultante.

(d) Construya el Analizador Sintáctico Descente Recursivo de la gramática obtenida

Ejercicio 3.13

La siguiente gramática representa la sintaxis de la instrucción de asignación de un lenguaje basado en conjuntos:

Asig → **id** **igual** *Expr* **pyc**

Expr → *Base*

Expr → *Expr* **union** *Base*

Expr → *Expr* **interseccion** *Base*

Base → **par_ab** *Expr* **par_ce**

Base → **id**

Base → **llave_ab** *Enum* **llave_ce**

Enum → λ

Enum → *Lista*

Lista → **num**

Lista → *Lista* **coma** **num**

A continuación se muestra un ejemplo de una cadena de entrada para esta gramática:

```
A = { 1 , 2 } ∪ ( { 3 , 4, 5 } ∩ B ) ;
```

(a) Realice las transformaciones necesarias para que cumpla la propiedad LL(1).

(b) Calcule los conjuntos Primeros, Siguientes y de Predicción para las reglas y símbolos de la gramática resultante.

(c) Construya la tabla de análisis sintáctico LL(1) de la gramática obtenida.

Ejercicio 3.14

Considere la siguiente gramática:

Expr → *Atom* | *List*

Atom → **number** | **identifier**

List → **lparen** *Seq* **rparen**

Seq → *Seq* *Expr* | *Expr*

(a) Realice las transformaciones necesarias para que cumpla la propiedad LL(1).

(b) Calcule los conjuntos Primeros, Siguientes y de Predicción para las reglas y símbolos de la gramática resultante.

(c) Construya la tabla de análisis sintáctico LL(1) de la gramática obtenida.

(d) Realice la traza a la siguiente entrada

```
( a ( b ( 2 ) ) ( c ) )
```

Ejercicio 3.15

La siguiente gramática representa la sintaxis simplificada del operador *new*:

NewOperator → **new id** *Constructor*

NewOperator → **new id** *Array*

NewOperator → **new type** *Array*

Constructor → **lparen** (**expr** (**comma expr**)*)? **rparen**

Array → (**lbracket expr rbracket**)+ (**lbracket rbracket**)*

(a) Realice las transformaciones necesarias para expresar la gramática en notación BNF.

(b) Realice las transformaciones necesarias para que cumpla la propiedad LL(1).

(c) Calcule los conjuntos Primeros, Siguientes y de Predicción para las reglas y símbolos de la gramática resultante.

(d) Construya la tabla de análisis sintáctico LL(1) de la gramática obtenida.

Ejercicio 3.16

La siguiente gramática representa la sintaxis de las claúsulas import en Java:

> *ImportClause* → **import id** (**dot id**)* (**dot star**)? **semicolon**

(a) Realice las transformaciones necesarias para obtener una gramática en notación BNF que cumpla la propiedad LL(1).

(b) Calcule los conjuntos Primeros, Siguientes y de Predicción para las reglas y símbolos de la gramática resultante.

(c) Construya la tabla de análisis sintáctico LL(1) de la gramática obtenida.

(d) Realice la traza para la cadena de entrada "**import id dot id dot id dot star semicolon**"

Ejercicio 3.17

La siguiente gramática representa la sintaxis de la instrucción de asignación de un lenguaje basado en conjuntos en notación EBNF:

> *Asig* → **id igual** *Expr* **pyc**
>
> *Expr* → *Comp* ((**union** | **interseccion**) *Comp*)*
>
> *Comp* → (**complemento**)? *Base*
>
> *Base* → (**llave_ab** (**num** (**coma num**)*)? **llave_ce** | **par_ab** *Expr* **par_ce** | **id**)

(a) Realice las transformaciones necesarias para expresarla en notación BNF.

(b) Realice las transformaciones necesarias para convertirla en una gramática LL(1).

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

(d) Construya la tabla de análisis sintáctico LL(1) para esa nueva gramática.

(e) Realice la traza de la siguiente cadena de entrada:

> A = { 1 , 2 } ∪ ({ 3 , 4 } ∩ ¬ B);

Ejercicio 3.18

La siguiente gramática representa la sintaxis de las expresiones formadas por productos y potencias de números, en notación EBNF:

Expresión → *Factor* (**prod** *Factor*)*

Factor → *Base* (**elev** *Base*)*

Base → (**num** | **lparen** *Expresión* **rparen**)

(a) Realice las transformaciones necesarias para expresarla en notación BNF.

(b) Realice las transformaciones necesarias para convertirla en una gramática LL(1).

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

(d) Construya la tabla de análisis sintáctico LL(1) para esa nueva gramática.

(e) Realice la traza de la siguiente cadena de entrada:

2 ^ 4 * 3 ^ (5 * 6) ^ 3

Ejercicio 3.19

La siguiente gramática, expresada en notación EBNF, permite describir una versión reducida de la sintaxis de Prolog.

Programa → (*Claúsula*)*

Claúsula → (*Hecho* | *Regla*)

Hecho → *Predicado* **dot**

Regla → *Predicado* **imp** *Literales* **dot**

Predicado → **atom** (**lparen** *Literales* **rparen**)?

Literales → *Literal* (**comma** *Literal*)*

Literal → **var** | **const** | *Predicado* | *Lista*

Lista → **lbracket** *Literales* (**tail** *Literal*)? **rbracket**

(a) Realice las transformaciones necesarias para expresarla en notación BNF.

(b) Realice las transformaciones necesarias para convertirla en una gramática LL(1).

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

(d) Construya la tabla de análisis sintáctico LL(1) para esa nueva gramática.

Ejercicio 3.20

La siguiente gramática, expresada en notación EBNF, permite describir un circuito formado por resistencias unidas en serie o en paralelo:

Circuito → *CSerie* (**paralelo** *CSerie*)*

CSerie → *CBase* (**serie** *CBase*)*

CBase → **resistencia** | **parab** *Circuito* **parce**

(e) Realice las transformaciones necesarias para obtener una gramática equivalente expresada en notación BNF y que cumpla la propiedad LL(1).

(f) Calcule los conjuntos Primeros, Siguientes y de Predicción para las reglas y símbolos de la gramática resultante.

Construya la tabla de análisis sintáctico LL(1) para la gramática resultante.

Ejercicio 3.21

A continuación se presenta la gramática de un lenguaje de representación de árboles:

Arbol → **tree id lbrace** *ListaDeNodos* **rbrace**

ListaDeNodos → *Nodo* (**comma** *Nodo*)*

Nodo → *NodoHoja* | *NodoInterno*

NodoHoja → **id**

NodoInterno → **id lbrace** *ListaDeNodos* **rbrace**

(a) Realice las transformaciones necesarias para expresarla en notación BNF.

(b) Realice las transformaciones necesarias para convertirla en una gramática LL(1).

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

(d) Construya la tabla de análisis sintáctico LL(1) para esa nueva gramática.

DISEÑO DE GRAMÁTICAS LL(1)

Ejercicio 3.22

Los ficheros fuente en Java comienzan con la declaración del paquete al que pertenecen las clases o interfaces definidas en el fichero, seguido de un conjunto de claúsulas de importación. Las claúsulas *import* indican al compilador que el fichero fuente puede contener referencias a una determinada clase de un paquete diferente.

Las claúsulas de importación comienzan con la palabra clave *import*, seguida del nombre completo de la clase, terminando en punto y coma. El nombre completo de la clase se forma con el nombre del paquete, seguido de un punto y el nombre de la clase. Si la clase pertenece a un subpaquete dentro de una jerarquía, entonces el nombre del paquete se forma con los nombres de los subpaquetes separados por puntos. El nombre de la clase se pude sustituir por un asterisco, indicando en ese caso que se están importando todas las clases del paquete indicado.

A continuación se muestran algunos ejemplos de claúsulas import:

```
import java.util.Vector;
import java.util.Stack;
import java.sql.*;
import javax.swing.border.*;
```

Desarrolle una gramática LL(1) que reconozca el formato de las claúsulas *import* de Java.

Ejercicio 3.23

En Java, el operador **new** permite crear tanto objetos de una cierta clase como matrices de un cierto tipo de datos. En el primer caso, la palabra clave **new** va seguida del constructor de la clase. En el caso de la creación de matrices, la palabra clave **new** va seguida del tipo de dato y de la dimensión de la matriz. El tipo de datos de la matriz puede ser tanto un tipo simple (**char**, **int**, **long**, **float**, **double**, ...) como objetos de una determinada clase. Es importante señalar que las matrices pueden ser multidimensionales y que se puede dejar sin especificar el tamaño de las últimas dimensiones de la matriz, pero nunca el tamaño de la primera dimensión.

A continuación se muestran algunos ejemplos de la instrucción **new**:

```
new Label("Titulo")
new Label[5][2]
new Label[3][][]
new int[2+3][j][]
```

Desarrolle una gramática LL(1) que reconozca el formato de la instrucción **new** de Java. Utilice para ello los tokens **new**, **lparen**, **rparen**, **lbracket**, **rbracket**, **comma** e **id**, así como los símbolos lingüísticos predefinidos *SimpleType* (que reconoce un tipo de datos simple) y *Expression* (que reconoce una expresión).

Ejercicio 3.24

La sintaxis de declaración de variables en C admite la posibilidad de declarar una única variable (“`int x;`”) o una lista de variables del mismo tipo base (“`int x, y, z;`”). Cada variable declarada puede tener asociado una expresión para calcular su valor inicial (“`int x=1, y;`”). Además, la sintaxis de C permite definir tanto tipos simples como arrays y punteros. Para definir punteros se introducen asteriscos delante del identificador de la variable (“`int **x;`”). Por su parte, los arrays pueden ser multidimensionales y cada dimensión viene expresada por una expresión de tipo entero entre corchetes (“`int x [2][2*a];`”). Es posible definir arrays de punteros (“`int *x[3];`”).

Describa por medio de una gramática LL(1) la sintaxis descrita anteriormente. Considere que se dispone de los símbolos no terminales *TypeIdentifier*, que reconoce el tipo base, y *Expression*, que reconoce expresiones de cualquier tipo.

Ejercicio 3.25

La declaración de las funciones en Haskell consta de su nombre (un identificador que comienza con una letra minúscula) y su tipo de datos. Por ejemplo, la declaración de la función divide, que verifica si un número entero es divisible por otro número entero, se declara así:

```
divide :: Integer -> Integer -> Bool
```

En general, las declaraciones de tipos en Haskell pueden ser de las siguientes formas:

(a) Un identificador de tipo predefinido, que en este caso comienza por una letra mayúscula (por ejemplo, "Integer").

(b) Una lista de datos de un cierto tipo base, que se expresa con la declaración del tipo base entre corchetes (por ejemplo, "[Integer]").

(c) Una tupla de diferentes tipos, que se expresa con la lista de tipos separada por comas y entre paréntesis (por ejemplo, "(Integer , Bool , String)").

(d) Una relación entre tipos, que se expresa uniendo los tipos mediante flechas (por ejemplo, " Integer -> Integer").

La combinación de estos tipos de construcciones permite declarar tipos de datos muy complejos. Por ejemplo, la expresión "[Integer -> Integer] -> (String , Integer) -> Bool" permite definir el tipo de una función que toma como primer argumento una lista de funciones (que toman un parámetro entero y devuelven un valor entero), como segundo argumento una tupla formada por una cadena y un entero y genera como salida un booleano.

Diseñe una gramática LL(1) de describa la declaración de tipos de Haskell. Considere para ello los siguientes tokens: **id**, **lbracket**, **rbracket**, **lparen**, **rparen**, **comma** y **arrow**.