

### Tema 3. Ejercicios.

- 3.1** Obtén una gramática independiente del contexto no ambigua que defina el lenguaje formado por las gramáticas independientes del contexto. Sobre esa gramática construye un E.T.D.S. que defina el proceso de traducción de una gramática en un string *a*, tal que:

*a es el string correspondiente al nombre del no terminal que tiene asociada la regla de producción con mayor número de símbolos en su parte derecha.*

La siguiente gramática constituye un ejemplo del formato que tienen las gramáticas pertenecientes al lenguaje que debes definir. Como puede observarse el ';' señala el final de las reglas correspondientes a un no terminal.

```
S    →   A B C1 ;
A    →   'a' B 'a'
      /   'o' ;
B    →   'b' B 'c'
      /   'b' 'c' DD
      /   'b' 'c' 'd' ;
C1   →   'c' 'e' 'r' 'e' 'z' 'o'
      /   'konstante' ;
DD   →   'd' 'a' 'm' 'o' 'c' 'l' 'e' 's'
      /   'd' 'e'
      /   'a' 'r' 'c' 'o' ;
```

La traducción asociada a esta gramática es D.

- 3.2** Obtén una gramática independiente del contexto **que permita el análisis descendente predictivo** y que defina el lenguaje formado por las expresiones regulares. Las expresiones regulares tendrán la forma que se describe a continuación:

```
Cualquier carácter es una expresión regular
λ (la cadena vacía) es una expresión regular
Si e, e1 y e2 son expresiones regulares
    e+ es una expresión regular
    e1 | e2 es una expresión regular
    e1 e2 es una expresión regular
    (e) es una expresión regular
```

Sobre esa gramática, construye un E.T.D.S. que obtenga sobre un atributo asociado al símbolo inicial de la gramática si una expresión regular genera solamente cadenas de longitud *tres*.

- 3.3** El siguiente ETDS está construido sobre una gramática ambigua. A partir de este ETDS, basándote en su especificación de las restricciones semánticas, obtén una gramática equivalente a la original, pero que permita el análisis descendente predictivo.

```

Expresion → entero {Expresion.tipo := 'entero'}
Expresion → id {Expresion.tipo := obtener_tipo(id.nombre)}
Expresion → Expresion mod Expresion
    {Si Expresion(1).tipo = 'entero' y Expresion(2).tipo = 'entero'
    entonces Expresion.tipo := 'entero'
    si no Expresion.tipo := 'error'}
Expresion → Expresion [ Expresion ]
    {Si Expresion(2).tipo = 'entero' y Expresion(1).tipo = array(n,t)
    entonces Expresion.tipo := t
    si no Expresion.tipo := 'error'}
Expresion → Expresion . all
    {Si Expresion(1).tipo = apuntador(t) entonces Expresion.tipo := t
    si no Expresion.tipo := 'error'}

```

**3.4** ¿Qué especifica el ETDS que se presenta a continuación? A partir de este ETDS obtén una gramática equivalente a la original pero que cumpla las condiciones LL(1).

```

LE → ( E RE ) {Si RE.val > E.val entonces LE.a:=RE.st
                si no LE.a:=E.st}
RE → , E RE { Si RE(1).val > E.val entonces
              RE.val:=RE(1).val; RE.st:=RE(1).st
              si no RE.val:=E.val; RE.st:=E.st;}
RE → ξ {RE.val:=-1; RE.st:=";}

E → E E + {E.val:=E(1).val + E(2).val; E.st:=E(1).st || '+' || E(2).st; E.op:='+';}
E → E E *
    {E.val:=E(1).val * E(2).val; E.op:='*';
    Si E(1).op=E(2).op='+' entonces
        E.st:='(' || E(1).st || ')' * ('||E(2).st||');
    si no si E(1).op='+' entonces E.st:='(' || E(1).st || ')' * ||E(2).st ;
    si no si E(2).op='+' entonces E.st:=E(1).st || '*' ('||E(2).st ||');
    si no E.st:=E(1).st || '*' || E(2).st ;}
E → digito {E.val:=digito.val; E.st:=digito.st; E.op:=null}

```

**3.5** Escribir un ETDS para traducir una expresión a notación postfija.  
Notación postfija:

- La traducción de  $E1 + E2$  es: traducción de  $E1$ , traducción de  $E2$  +
- Si  $E$  es un número o identificador, entonces su traducción es  $E$
- Una expresión del tipo  $(E)$  dará como resultado la traducción de  $E$  (no se necesitan paréntesis en notación postfija).

Ejemplo:	$x * y * z$	----->	$xy*z*$
	$3 + x * (a + b)$	----->	$3xab+*+$
	$3 + (a + b) + c$	----->	$3ab++c+$

Obtener los árboles de análisis y la traducción correspondientes a:  $9-5+2$  y  $9-5*2$

**3.6** Haz lo mismo para realizar la traducción en sentido contrario. También para pasar de notación normal a prefija (es decir, la traducción de la expresión  $x+y$  es  $+xy$ )

**3.7** Construir un E.T.D.S. que especifique la traducción de expresiones aritméticas en notación postfija a expresiones en notación infija sin paréntesis redundantes.

Unos paréntesis son redundantes si tras eliminarlos las operaciones se realizan en el mismo orden en que se realizarían con los paréntesis.

Ejemplo:

$(3 + 5) + 2$	Los paréntesis son redundantes.
$3 + (5 + 2)$	Los paréntesis no son redundantes
$(3 + 5) * 2$	Los paréntesis no son redundantes.
$3 + (5 * 2)$	Los paréntesis son redundantes.

**3.8** Se desea añadir una nueva instrucción al lenguaje con el que hemos trabajado en el tema 2. Se trata de la instrucción *do*. Su forma es la siguiente:

```
do
    instrucción1;
    ...
    instrucción n;
while (E) ;
```

Su semántica es la repetición de las instrucciones hasta que la expresión sea falsa. Por tanto, las instrucciones interiores se ejecutarán al menos una vez.

Para complementar el comportamiento anterior existe una segunda instrucción: **continue**. Dicha instrucción puede aparecer dentro de un **do**, y se utiliza para terminar la iteración. La consecuencia es que el control de flujo irá a la evaluación de la expresión. Por ejemplo:

```
do
    i=i*2;
    if (i==10) continue;
    ...
while (i>100) ;
```

Suponiendo que los atributos asociados a expresión son NOMBRE, TRUE y FALSE y suponiendo que el atributo asociado a *instruccion* es NEXT, y que su significado es el ya conocido y presentado en el tema 2:

- Construye un E.T.D.S. que defina el proceso de traducción a código de tres direcciones de estas nuevas instrucciones.
- Si se quisiera añadir como restricción semántica a la instrucción **do** que la instrucción **continue** sólo pueda aparecer dentro de un **do**, ¿se trataría de validación estática o dinámica? Razona tu respuesta. Incluye en el ETDS la acciones necesarias para realizar dicha validación.

**3.9** Se desea añadir una nueva instrucción al lenguaje con el que hemos trabajado en el tema 2. Se trata de la instrucción *new\_loop*. Su forma es la siguiente:

```
new_loop
    instrucción;
end_new_loop identificador if expresión;
```

Su semántica consiste en repetir la ejecución de *instrucción* hasta que se cumpla la condición representada por la *expresión*.

Para complementar el comportamiento anterior existe una segunda instrucción: **exit\_new\_loop** *identificador*. Esta instrucción puede aparecer en las instrucciones englobadas en un **new\_loop** y se utiliza para acabar “bruscamente” con la ejecución de la instrucción **new\_loop** etiquetada con el *identificador* en cuestión. Tiene como efecto transferir el flujo de control a la siguiente instrucción al **new\_loop** etiquetado con ese *identificador*.

Suponiendo que los atributos asociados a expresión son NOMBRE, TRUE y FALSE y suponiendo que el atributo asociado a *instrucción* es NEXT, y que su significado es el ya conocido y presentado en el tema 2, y considerando que la solución deberá desarrollarse **sin** recurrir a la tabla de símbolos ni a objetos globales:

- Construye un E.T.D.S. que defina el proceso de traducción a código de tres direcciones de estas nuevas instrucciones.
- La restricción semántica que exige que la *expresión* sea de tipo booleano, nos llevaría a hablar de validación estática o dinámica ?
- Incluye en el ETDS las acciones necesarias para verificar que cada **exit\_new\_loop** *identificador* se encuentra correctamente anidado dentro de un **new\_loop** etiquetado con el *identificador*.

**3.10** Se desea añadir una nueva instrucción al lenguaje con el que hemos trabajado en el tema 2, se trata de la instrucción **newfor**. Su forma es la siguiente:

```
newfor id [ascending / descending] from E to E
do
    instrucción;
endnewfor;
```

Su semántica es similar a la de la instrucción *for*, de forma que el identificador índice del *newfor* irá variando su valor (aumentando o disminuyendo de uno en uno) desde el valor inicial, determinado por la primera expresión, hasta el valor final, determinado por la segunda.

**Ejemplos:**      **newfor** I **ascending** **from** 1 **to** n+1 **to**...**endnewfor**;  
                  **newfor** adibidea **descending** **from** n\*2 **to** n **do** ... **endnewfor**;

Suponiendo que el atributo asociado a E es E.nombre y el que atributo asociado a *instrucción* es NEXT y que su significado es el ya conocido y presentado en el tema 2:

- Construye un ETDS que defina el proceso de traducción a código de tres direcciones de esta nueva instrucción.
- Si se quisiera añadir como restricción semántica a la instrucción *newfor* que la primera expresión debe tomar como valor uno menor o igual que la segunda, si se trata de un “*newfor ascending*”, y que, caso de tratarse de un “*newfor descending*”, la primera expresión debe tomar un valor mayor o igual que la segunda, ¿se estaría hablando de validación estática o dinámica? Razona tu respuesta y añade al ETDS las acciones encargadas de plasmar la validación propuesta.