PRÁCTICA GENERAL

(Curso 2022-2023)

Objetivo:

El objetivo la práctica es construir un traductor sencillo de un lenguaje para la especificación gráfica de un árbol (**LEA**), que genere la estructura del árbol con formato de grafo, definido por el lenguaje **DOT** (lenguaje de definición de grafos).

En este sentido, se parte de un fichero de texto que describe un árbol y obtendremos otro texto, que describe la estructura del árbol leída, con una estructura de grafo, en el lenguaje DOT.

Se usarán las herramientas Jflex y Cup. La práctica se realizará por módulos, y de forma incremental en las distintas sesiones de prácticas, según se avance en las sesiones de teoría.

Jflex => Análisis de léxico

- Definiremos los elementos léxicos del lenguaje de árboles (LEA), de forma que se genera el java con la implementación del autómata finito que los reconoce.

CUP => Análisis sintáctico y generación del código en DOT

- Definiremos la gramática en CUP, generando el java que implementa el autómata a pila (análisis sintáctico)
- Añadiremos reglas semánticas que nos permitan generar el fichero DOT a la vez que se realiza el análisis sintáctico

Antecedentes:

El lenguaje **DOT** permite definir, en texto plano, las características de un grafo; no solo los nodos y los arcos si no también el formato de ellos (color, fuente, forma de linea o de nodo etc).

Existen varios programas que procesan un fichero dot y generan su representación gráfica en diferentes formatos (jpg, png, svg, pdf etc) y muchos de ellos utilizan el paquete **Graphviz. (https://graphviz.gitlab.io/documentation/)**

Para instalar Graphviz vamos a: https://graphviz.org/download/ , descargamos el instalador y lo instalamos

Para compilar un código DOT, en la linea de comandos:

- `dot ejemplo1.dot -o ejemplo1.png -Tpng`

donde -Tpng indica que el formato de salida del grafo es png, en general, tendremos T<tipodesalida>

Descripción del lenguaje dot

El lenguaje dot permite definir nodos y arcos en cualquier orden. A cada nodo o arco se le puede asociar una lista de etiquetas que definen su formato.
Los etiquetas que tendremos en cuenta son:

- **label**, cuyos valores son cadenas
- **color**, cuyos valores pueden ser, entre otros, blue, green, red, yellow
- **fontcolor** cuyos valores pueden ser, entre otros, blue, green, red, yellow
- **style** cuyos valores pueden ser, entre otros, solid|dashed|bold
- **shape** cuyos valores pueden ser, entre otros, **s**quare|circle|doublecircle|rectangle
- **dir c**uyos valores pueden ser, entre otros, none|forward|back|both.

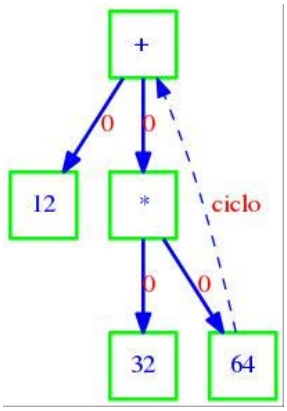
Donde la etiqueta **shape** solo se aplica a nodos y la etiqueta **dir** solo se aplica a arcos. El resto de las etiquetas se aplican tanto a nodos como a arcos.

También podemos definir una lista de atributos para todos los nodos o una lista de atributos para todos los arcos, que afectará a todos los nodos o arcos definidos posteriormente (hasta que haya una posible nueva definición).

Por ejemplo:

node [shape = square, label = "a", color = "green", fontcolor = blue , style = bold]
edge [dir = forward, label = "0", color = "blue", fontcolor = red , style = bold]

Ejemplo de grafo en dot:




```
digraph ejemplo {
node [shape = square, label = "a", color = "green", fontcolor = blue , style = bold]
edge [dir = forward, label = "0", color = "blue", fontcolor = red , style = bold]

n1 [label="+"];
n11 [label="12"];
n12 [label="*"];
n121 [label="32"];
n122 [label="64"];
n1 -> n11;
n1 -> n12;
n12 -> n121;
n12 -> n122;
n122 -> n1 [label="ciclo", style = dashed ];
}
```

Lenguaje (LEA) para describir un árbol

Vamos a desarrollar el lenguaje LEA, que es muy parecido a DOT, en el que vamos a especificar:

- Una primera linea con la palabra reservada árbol, el nombre del arbol y una apertura de llave, que se cerrará cuando se termine de definir el árbol.
    - arbol ejemplo {
- Podremos también definir las características generales de un nodo o de un arco, con la misma sintaxis de DOT.
    - node [shape = square, label = "a", color = "green", fontcolor = blue , style = bold]
    - edge [dir = forward, label = "0", color = "blue", fontcolor = red , style = dashed]
- Definición de cada nodo: viene dada por el nombre del nodo, seguido de la lista de hijos (si tiene hijos) seguido del formato del nodo.
    - d hijos = {e [label = "14"] } [label ="D"];
    - e [label = "E 14", shape = circle];
    - donde cada hijo podrá tener asociada una lista de atributos que corresponden al arco (Edge). Por defecto, las características de cada arco son las definidas en Edge.

En general, cualquier valor puede ir entre comillas o sin comillas

Se cumplen las reglas sobre atributos y sus valores, descritas para dot.

Se pueden incluir comentarios del tipo:

```
/* comentario en varias lineas */

// comentario en una linea

# comentario en una linea
```

Ejemplo de código del árbol

```
arbol ejemplo {
node [shape = square, label = "a", color = "green", fontcolor = blue , style = bold]
edge [dir = forward, label = "0", color = "blue", fontcolor = red , style = dashed]

a hijos = {b [label = "6", color = "red"],c [label = "100"]}    [label = "A", color = "blue", shape = circle];
b hijos = {d [label = "80"] }  [label = "B"];
c hijos = {f [label = "20"], g [label = "22"]}   [label = "C"];
d hijos = {e [label = "14"] }  [label ="D"];
e [label = "E 14", shape = circle];
f [label = "F 5",  shape = circle];
g [label = "G 4", shape = circle];
}
```

Salida en formato dot

```
digraph ejemplo {
 node [ shape = square, label = "a", color = "green", fontcolor = blue, style = bold ]
 edge [ direction = forward, label = "0", color = "blue", fontcolor = red, style = dashed ]

a [ label = "A", color = "blue", shape = circle ];
a -> c [ label = "100" ];
a -> b [ label = "6", color = "red" ];

b [ label = "B" ];
b -> d [ label = "80" ];

c [ label = "C" ];
c -> g [ label = "22" ];
c -> f [ label = "20" ];

d [ label = "D" ];
d -> e [ label = "14" ];

e [ label = "E 14", shape = circle ];
f [ label = "F 5", shape = circle ];
g [ label = "G 4", shape = circle ];
}
```