# Práctica de autómatas y desarrollo avanzado de software
# Gramáticas top down

10 de octubre de 2018

## Enunciado

### Gramática sencilla para un razonador

Escribe en un fichero una gramática que valga para un parser descendente recursivo para el lenguaje descrito a continuación (sólo la gramática). El lenguaje es de un razonador y entiende lógica de primer orden. Los comentarios comienzan y acaban con **.

Tiene 3 operadores binarios:

```
|  & then
```

Tiene un operador unario:

```
not
```

La asignación se hace con `:`. Hay dos valores literales, `true` y `false`. El lenguaje consiste en sentencias de asignación y expresiones de lógica de primer orden. Dos ejemplos de asignación:

```
a:true
b:a
```

Ejemplos de expresiones:

```
a then b
a | b
a then true | c
(a & b) then c
```

Se pueden usar paréntesis para desambigüar. El operador 'not' tiene más precedencia. En orden de precedencia luego está el operador 'then' y luego todos los demás, que tienen la misma precedencia. Todos los operadores se evaluan de izquierda a derecha. Un ejemplo del lenguaje es:

```
** this is a comment  **
** this is an or expression  **
a | b
** the and is the & **
a & x
** identifiers can be any alphabetic followed by any alphanumeric **
a2424bbb
** these are assignments **
abc:true
c: d
** any unassigned variable is false **
** the implication is then **
Zaaaa2342u then bbbRNEe
** comments start and end with two ** a  then c
a & b then c | dabb1c
not a | c then c & (b | d) then true
** literals for truth values are true and false, for example **
true & false then c & (b | d)
true & false then false
false
```

## Parser descendente para fx simplificado

Escribe un parser descendente recursivo en el lenguaje de programación go para la versión básica del lenguaje gráfico gix descrita a continuación (no tiene expresiones, ni tipos de datos definidos por el usuario, ni tipos de datos compuestos, ni declaraciones de variables locales). Escribe primero la gramática en un fichero para entregarla también.

Para escribir el parser, puedes usar el lexer que ya escribiste en la práctica anterior. Sólo debe ir escribiendo las estructuras que reconoce. No es necesario que genere código ni que construya ninguna estructura de datos (ni siquiera la tabla de símbolos, con lo que tampoco tendrá ámbitos ni declaraciones). El parser debe estar en un paquete separado que se llame gixparse y debe tener tests que lo prueben para ejecutar con `go test`. Un ejemplo de programa de la versión simplificada de gix con su descripción es:

```
//basic types bool, int (64 bits),
//literals are of type int, 2, 3, or 0x2dfadfd
//no operators, no expressions, no composite data types
//only implicit declarations in loops,
//no explicit local variable declaration

//builtins
//circle(p, 2, 0x1100001f);
//      at point p, int radius r, color: transparency and rgb
//rect(p, α, col);
//      at point p, int angle alpha (degrees),
//      color: transparency (0-100) and rgb



//macro definition
func line(int x, in y){
                                //last number in loop is the step
        iter (i := 0; x, 1){    //declares it, scope is the loop
                circle(2, 3, y, 5);
        }
}

//macro entry
func main(){
        iter (i := 0; 3, 1){
                rect(i, i, 3, 0xff);
        }
        iter (j := 0; 8, 2){    //loops 0 2 4 6 8
                rect(j, j, 8, 0xff);
        }
        circle(4, 5, 2, 0x11000011);
}
```