

```

1 module Lomoba where
2 import Grafo
3 import List
4 import qualified Data.List (union)
5 import Tipos
6 -----Sección 6----- Lomoba -----
7
8 -- Ejercicio 10
9 foldExp :: (Prop -> a) -- Función para aplicar a (Var p)
10         -> (a -> a)     -- Función para aplicar a (Not e)
11         -> (a -> a -> a) -- Función para aplicar a (Or e1 e2)
12         -> (a -> a -> a) -- Función para aplicar a (And e1 e2)
13         -> (a -> a)     -- Función para aplicar a (D e)
14         -> (a -> a)     -- Función para aplicar a (B e)
15         -> Exp -> a     -- Función que construye el fold
16 foldExp fVar fNot fOr fAnd fd fb (Var p) = fVar p
17 foldExp fVar fNot fOr fAnd fd fb (Not e) = fNot (foldExp fVar fNot fOr fAnd fd fb e)
18 foldExp fVar fNot fOr fAnd fd fb (Or a b) = fOr
19                                     (foldExp fVar fNot fOr fAnd fd fb a)
20                                     (foldExp fVar fNot fOr fAnd fd fb b)
21 foldExp fVar fNot fOr fAnd fd fb (And a b) = fAnd (foldExp fVar fNot fOr fAnd fd fb a)
22                                     (foldExp fVar fNot fOr fAnd fd fb b)
23 foldExp fVar fNot fOr fAnd fd fb (D e) = fd (foldExp fVar fNot fOr fAnd fd fb e)
24 foldExp fVar fNot fOr fAnd fd fb (B e) = fb (foldExp fVar fNot fOr fAnd fd fb e)
25
26 -- Ejercicio 11
27 -- Calcula la visibilidad de la f ormula. Cada vez que aparece <> o [] debo
28 -- incrementar en 1 el valor de la misma.
29 -- * Var es el caso base, por lo que utilizare const 0.
30 -- * Not no sumara ningun valor, por lo que utilizare id.
31 -- * Or y And que seran bifurcaciones en el arbol de recursion y por lo tanto
32 -- tomaremos el maximo resultante de ambas ramas de la recursion.
33 -- * D y B seran los casos en donde tendre que sumar uno, por lo que aplicare
34 -- la funcion + con la valuacion parcial en el primero de sus parametros.
35 visibilidad :: Exp -> Integer
36 visibilidad = foldExp fVar fNot fOr fAnd fd fb
37   where fVar = const 0
38         fNot = id
39         fOr = max
40         fAnd = max
41         fd = (+1)
42         fb = (+1)
43
44 -- Ejercicio 12
45 -- Extraer las variables proposicionales que aparecen en la formula, sin repetir.
46 -- * Var es el caso base, por lo que utilizare una lambda que dado p me devuelve [p].
47 -- * Or y And que seran bifurcaciones en el arbol de recursion y por lo tanto
48 -- utilizaremos la union de conjuntos.
49 -- * Not, D y B no adicionaran ningun simbolo, por lo que utilizare id.
50 extraer :: Exp -> [Prop]
51 extraer = foldExp fVar fNot fOr fAnd fd fb
52   where fVar = (\p -> [p])
53         fNot = id
54         fOr = Data.List.union
55         fAnd = Data.List.union
56         fd = id
57         fb = id
58
59 -- Ejercicio 13
60 eval :: Modelo -> Mundo -> Exp -> Bool
61 eval mod w exp = (eval' mod exp) w
62
63 -- Dado un modelo y una expresion, devuelve una función
64 -- que para un Mundo dado (en el modelo), devuelve la evaluación.
65 -- En los pasos de D y B, se aplica la función recursiva sobre todos
66 -- los mundos vecinos y se busca que en alguno o en todos la expresión
67 -- sea true.
68 eval' :: Modelo -> Exp -> (Mundo -> Bool)
69 eval' (K g mundosTrue) =
70   foldExp

```

```

71      (\p w -> w `elem` (mundosTrue p)) -- ::Prop -> (Mundo -> Bool)
72      (\rec w -> not (rec w)) -- ::(Mundo -> Bool) -> Mundo -> Bool
73      (\rec1 rec2 w -> (rec1 w) || (rec2 w))
74      (\rec1 rec2 w -> (rec1 w) && (rec2 w))
75      (\rec w -> or (map rec (vecinos g w))) -- rec::(Mundo -> Bool)
76      (\rec w -> and (map rec (vecinos g w)))
77
78
79
80 -- Ejercicio 14
81 -- Dadas todas las variables proposicionales del grafo, se devuelven los mundos
82 -- que al evaluarlos dan verdadero
83 valeEn :: Exp -> Modelo -> [Mundo]
84 valeEn exp mod@(K g mundosTrue) = filter (eval' mod exp) (nodos g)
85
86 -- Ejercicio 15
87 -- Usando foldr, voy construyendo un nuevo modelo de Kripke, partiendo
88 -- del modelo pasado por argumento y sacándole los mundos donde no vale
89 -- la expresión. Por cada mundo que saco, le saco el nodo al grafo y
90 -- lo saco del valor de retorno de la función (para cada símbolo
91 -- proposicional)
92 quitar :: Exp -> Modelo -> Modelo
93 quitar e mod@(K g mundosTrue) =
94     foldr (\w (K gRec fRec) ->
95         K (sacarNodo w gRec)
96           (\prop -> filter (/=w) (fRec prop))
97         )
98     mod -- Si e vale en todos los mundos, devuelvo el
99     modelo original
100     (valeEn (Not e) mod) -- mundos donde NO vale e
101
102 -- Ejercicio 16
103 -- Compara el modelo original con el modelo resultante de quitarle los mundos tales
104 -- que no valga e.
105 cierto :: Modelo -> Exp -> Bool
106 cierto mod@(K g mundosTrue) e = (sort (nodos g) == sort (valeEn e mod))
107
108

```