```haskell
import Grafo
import Tipos
import Lomoba
import Parser
import Test.HUnit

-- evaluar t para correr todos los tests
t = runTestTT allTests

allTests = test [
    "parser" ~: testsParser,
    "grafo" ~: testsGrafo,
    "lomoba" ~: testsLomoba
    ]

testsParser = test [
    (Var "p")                       ~=? (parse "p"),
    (And (Var "p") (Var "q"))       ~=? (parse "p && q"),
    (Or (Var "p") (Var "q"))        ~=? (parse "p || q"),
    (Or (Not (Var "p")) (Var "q"))  ~=? (parse "!p || q"),
    (And (D (Var "p")) (Var "q"))   ~=? (parse "<>p && q"),
    (And (B (Var "p")) (Var "q"))   ~=? (parse "[]p && q"),
    (D (And (Var "p") (Var "q")))   ~=? (parse "<>(p && q)"),
    (B (And (Var "p") (Var "q")))   ~=? (parse "[](p && q)")]

testsGrafo = test [
    -- Ej 1,2,4 (agregar nodos, ver nodos, grafo vacío)
    [1] ~~? (nodos (agNodo 1 vacio)),
    [1,2] ~~? (nodos (agNodo 2 (agNodo 1 vacio))),
    [1,2] ~~? (nodos (agNodo 2 (agNodo 2 (agNodo 1 vacio)))),

    -- Ej 3,6 (agrega ejes, ver vecinos)
    [] ~~? (vecinos (agNodo 3 (agNodo 2 (agNodo 1 vacio))) 1),
    [] ~~? (vecinos (agNodo 3 (agNodo 2 (agNodo 1 vacio))) 5), -- es total
    [2] ~~? (vecinos (agEje (3,2) (agNodo 3 (agNodo 2 (agNodo 1 vacio)))) 3),
    [2,3] ~~? (vecinos (agEje (3,3) (agEje (3,2)
                  (agNodo 3 (agNodo 2 (agNodo 1 vacio))))) 3),
    [2] ~~? (vecinos (agEje (3,2) (agEje (3,2) (agNodo 3 (agNodo 2 (agNodo 1 vacio))))) 3),
    [1,2,3,4] ~~? (vecinos (agEje (5,1) (agEje (5,2) (agEje (5,3) (agEje (5,4)
                     (agNodo 5 (agNodo 4 (agNodo 3 (agNodo 2 (agNodo 1 vacio)))))))))) 5),

    -- Ej 5 (sacar nodo)
    [1,3] ~~? nodos (sacarNodo 2 (agNodo 3 (agNodo 2 (agNodo 1 vacio)))),
    [1,2,3] ~~? nodos (agNodo 2 (sacarNodo 2
                         (agNodo 3 (agNodo 2 (agNodo 1 vacio))))),
    [1] ~~? vecinos (sacarNodo 2 (agEje (3,2) (agEje (3,1) (agNodo 3 (agNodo 2 (agNodo 1
    vacio))))))) 3,
    [] ~~? vecinos (sacarNodo 2 (agNodo 3 (agNodo 2 (agNodo 1 vacio)))) 2,

    -- Ej 7 (lineal)
    (agEje (2,3) (agEje (1,2) (agNodo 3 (agNodo 2 (agNodo 1 vacio))))) ~=? lineal [1,2,3],

    -- Ej 8 (union de grafos)
    -- Grafos disjuntos
    (agEje (3,4) (agEje (1,2) (agNodo 4 (agNodo 3 (agNodo 2 (agNodo 1 vacio))))))
                         ~=? union (agEje (3,4) (agNodo 4 (agNodo 3 vacio)))
                                   (agEje (1,2)(agNodo 2 (agNodo 1 vacio))),
    -- Grafo vacio
    (agEje (1,2) (agNodo 2 (agNodo 1 vacio)))
                         ~=? union vacio
                                   (agEje (1,2)(agNodo 2 (agNodo 1 vacio))),
    -- Grafos lineales
    (lineal [1,2,3,4,5,6]) ~=? union (lineal [1,2,3]) (lineal [3,4,5,6]),
    -- Algunos nodos en común
    (agEje (1,2) (agEje (1,3) (agEje (2,3) (agNodo 3 (agNodo 2 (agNodo 1 vacio))))))
                         ~=? union (agEje (1,2) (agNodo 1 (agNodo 2 vacio)))
                                   (agEje (1,3) (agEje (2,3)
                                         (agNodo 3 (agNodo 2 (agNodo 1 vacio))))),

```

```
69      -- Grafos idénticos
70      (agEje (1,3) (agEje (2,3) (agNodo 3 (agNodo 2 (agNodo 1 vacio)))))
71                           ~=? union (agEje (1,3) (agEje (2,3)
72                                                  (agNodo 3 (agNodo 2 (agNodo 1 vacio)))))
73                                     (agEje (1,3) (agEje (2,3)
74                                                  (agNodo 3 (agNodo 2 (agNodo 1 vacio))))),
75      -- Grafos idénticos en distinto orden
76      (agEje (1,3) (agEje (2,3) (agNodo 3 (agNodo 2 (agNodo 1 vacio)))))
77                           ~=? union (agEje (1,3) (agNodo 1 (agEje (2,3)
78                                                  (agNodo 3 (agNodo 2 vacio)))))
79                                     (agEje (1,3) (agEje (2,3)
80                                                  (agNodo 3 (agNodo 2 (agNodo 1 vacio))))),
81
82      -- Ej 9 (clausura transitiva)
83      -- hago un grafo que es un ciclo y deberia obtener un completo con la clausura
84      [1,2,3,4] ~~? vecinos (clausura (agEje (4,1) (lineal [1,2,3,4]))) 1,
85      [1,2,3,4] ~~? vecinos (clausura (agEje (4,1) (lineal [1,2,3,4]))) 2,
86      [1,2,3,4] ~~? vecinos (clausura (agEje (4,1) (lineal [1,2,3,4]))) 3,
87      [1,2,3,4] ~~? vecinos (clausura (agEje (4,1) (lineal [1,2,3,4]))) 4
88      ]
89
90  testsLomoba = test [
91      -- Ej 11
92      0 ~=? visibilidad (parse "p"),
93      1 ~=? visibilidad (parse "<>p"),
94      2 ~=? visibilidad (parse "<>!(<>p)"),
95      2 ~=? visibilidad (parse "<><>p||<><>q"),
96      3 ~=? visibilidad (parse "<>(<>p||<><>q)"),
97      3 ~=? visibilidad (parse "[](<>p&&<>[]q)"),
98      2 ~=? visibilidad (parse "<><>p||<><>q||<><>r"),
99      0 ~=? visibilidad (parse "p||q||r||s||t"),
100     10 ~=? visibilidad (parse "[]<>[]<>[]<>[]<>[]<>p"),
101     4 ~=? visibilidad (parse "[][][](p|||[]q||r||<>s||t)"),
102
103     -- Ej 12
104     ["p"] ~=? extraer (parse "p"),
105     ["p"] ~=? extraer (parse "<>p"),
106     ["p"] ~=? extraer (parse "<>!<>p"),
107     ["p","q"] ~=? extraer (parse "<><>p||<><>q"),
108     ["p","q"] ~=? extraer (parse "<>(<>p||<><>q)"),
109     ["p","q"] ~=? extraer (parse "[](<>p&&<>[]q)"),
110     ["p","q","r"] ~=? extraer (parse "<><>p||<><>q||<><>r"),
111     ["p","q","r","s","t"] ~=? extraer (parse "p||q||r||s||t"),
112     ["p"] ~=? extraer (parse "<>[]<>[]<>[]<>[]<>p"),
113     ["p","q","r","s","t"] ~=? extraer (parse "[][][](p|||[]q||r||<>s||t)"),
114
115     -- Ej 13
116     True ~=? eval modeloKrEnunciado 1 (parse "p&&[]q"),
117     True ~=? eval modeloKrEnunciado 1 (parse "p&&<>r"),
118     False ~=? eval modeloKrEnunciado 1 (parse "[]r"),
119     True ~=? eval modeloKrEnunciado 1 (parse "<>(q&&r)"),
120     False ~=? eval modeloKr1 1 (parse "<>(q&&r)"),
121     True ~=? eval modeloKr1 1 (parse "<>(<>r)"),
122     True ~=? eval ciclo 2 (parse "<>q"),
123
124     -- Ej 14
125     [1] ~=? valeEn (parse "p") modeloKrEnunciado,
126     [5,4,3,2,1] ~=? valeEn (parse "<>p") ciclo,
127     [] ~=? valeEn (parse "<>!<>p") ciclo,
128     [3,2,1] ~=? valeEn (parse "<>r||<>q") modeloKr1,
129     [5,4,3,2,1] ~=? valeEn (parse "<>(<>p||<><>q)") ciclo,
130     [5,4,3,2] ~=? valeEn (parse "!p||q||r") modeloKr1,
131
132     -- Ej 15
133     [1] ~~? (\(K g f) -> nodos g)(quitar (parse "p&&[]q") modeloKrEnunciado),
134     [] ~~? ((\(K g f) -> f)(quitar (parse "p&&[]q") modeloKrEnunciado)) "q",
135     [1] ~~? ((\(K g f) -> f)(quitar (parse "p&&[]q") modeloKrEnunciado)) "p",
136
137
```

```haskell

    [2,3] ~~? (\(K g f) -> nodos g)(quitar (parse "q") modeloKrEnunciado),
    [3] ~~? ((\(K g f) -> f)(quitar (parse "q") modeloKrEnunciado)) "r",
    [2,3] ~~? ((\(K g f) -> f)(quitar (parse "q") modeloKrEnunciado)) "q",

    (\(K g f) -> g)modeloKrEnunciado ~=?
                (\(K g f) -> g)(quitar (parse "q||<>r") modeloKrEnunciado),

    -- Ej 16
    True ~=? cierto ciclo (parse "p"),
    False ~=? cierto ciclo (parse "q"),
    False ~=? cierto modeloKrEnunciado (parse "q"),
    False ~=? cierto modeloKrEnunciado (parse "p&&r || q"),
    True ~=? cierto modeloKrEnunciado (parse "p || q || r"),
    True ~=? cierto ciclo (parse "p || q || r"),
    True ~=? cierto modeloKr1 (parse "p || q || r"),
    False ~=? cierto modeloKr1 (parse "[]p"),
    False ~=? cierto ciclo (parse "<>(q&&r)")
    ]


-- El grafo del enunciado
modeloKrEnunciado = K (agEje (1,2) (agEje (1,3) (agNodo 3 (agNodo 2 (agNodo 1 vacio)))))
                        (\p -> case () of
                                  _ | p=="p" -> [1]
                                    | p=="q" -> [2,3]
                                    | p=="r" -> [3]
                                    | otherwise -> []) -- debe ser total

modeloKr1 = K (agEje (1,2) (agEje (1,3) (agEje (3,4) (agEje (2,5)
                (agNodo 5 (agNodo 4 (agNodo 3 (agNodo 2 (agNodo 1 vacio))))))))
                        (\p -> case () of
                                  _ | p=="p" -> [1]
                                    | p=="q" -> [2,3]
                                    | p=="r" -> [4,5]
                                    | otherwise -> []) -- debe ser total

ciclo = K (agEje (1,2) (agEje (2,3) (agEje (3,4) (agEje (4,5) (agEje (5,1)
                (agNodo 5 (agNodo 4 (agNodo 3 (agNodo 2 (agNodo 1 vacio))))))))))
                        (\p -> case () of
                                  _ | p=="p" -> [1, 2, 3, 4, 5]
                                    | p=="q" -> [3]
                                    | p=="r" -> [4]
                                    | otherwise -> []) -- debe ser total

---------------
--  helpers  --
---------------

-- idem ~=? pero sin importar el orden
(~~?) :: (Ord a, Eq a, Show a) => [a] -> [a] -> Test
expected ~~? actual = (sort expected) ~=? (sort actual)
    where
        sort = foldl (\r e -> push r e) []
        push r e = (filter (e<=) r) ++ [e] ++ (filter (e>) r)

```