Lógica en Haskell

José A. Alonso Jiménez

Grupo de Lógica Computacional
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, 20 de Diciembre de 2007 (Versión de 24 de enero de 2008)

Índice general

Capítulo 1

Sintaxis y semántica de la lógica proposicional

```
module SintaxisSemantica where

-- ----------------------------------------------------------------------
-- Librerías auxiliares                                                 --
-- ----------------------------------------------------------------------

import Data.List
import Test.HUnit
import Verificacion

-- ----------------------------------------------------------------------
-- Gramática de fórmulas prosicionales                                  --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 1: Definir los siguientes tipos de datos:
-- * SímboloProposicional para representar los símbolos de proposiciones
-- * Prop para representar las fórmulas proposicionales usando los
--   constructores Atom, Neg, Conj, Disj, Impl y Equi para las fórmulas
--   atómicas, negaciones, conjunciones, implicaciones y equivalencias,
--   respectivamente.
-- ----------------------------------------------------------------------

type SímboloProposicional = String
```

```
data Prop = Atom SímboloProposicional
          | Neg Prop
          | Conj Prop Prop
          | Disj Prop Prop
          | Impl Prop Prop
          | Equi Prop Prop
          deriving (Eq,Ord)

instance Show Prop where
    show (Atom p)   = p
    show (Neg p)    = "no " ++ show p
    show (Conj p q) = "(" ++ show p ++ " /\\ " ++ show q ++ ")"
    show (Disj p q) = "(" ++ show p ++ " \\/ " ++ show q ++ ")"
    show (Impl p q) = "(" ++ show p ++ " --> " ++ show q ++ ")"
    show (Equi p q) = "(" ++ show p ++ " <--> " ++ show q ++ ")"

-- ----------------------------------------------------------------------
-- Ejercicio 2: Definir las siguientes fórmulas proposicionales
-- atómicas: p, p1, p2, q, r, s, t y u.
-- ----------------------------------------------------------------------

p, p1, p2, q, r, s, t, u :: Prop
p  = Atom "p"
p1 = Atom "p1"
p2 = Atom "p2"
q  = Atom "q"
r  = Atom "r"
s  = Atom "s"
t  = Atom "t"
u  = Atom "u"

-- ----------------------------------------------------------------------
-- Ejercicio 3: Definir la función
--    no :: Prop -> Prop
-- tal que (no f) es la negación de f.
-- ----------------------------------------------------------------------

no :: Prop -> Prop
no = Neg
```

```
-- ----------------------------------------------------------------------
-- Ejercicio 4: Definir los siguientes operadores
--    (/\), (\/), (-->), (<-->) :: Prop -> Prop -> Prop
-- tales que
--    f /\ g      es la conjunción de f y g
--    f \/ g      es la disyunción de f y g
--    f --> g     es la implicación de f a g
--    f <--> g    es la equivalencia entre f y g
-- ----------------------------------------------------------------------

infixr 5 \/
infixr 4 /\
infixr 3 -->
infixr 2 <-->
(/\), (\/), (-->), (<-->) :: Prop -> Prop -> Prop
(/\)   = Conj
(\/)   = Disj
(-->)  = Impl
(<-->) = Equi

-- ----------------------------------------------------------------------
-- Símbolos proposicionales de una fórmula                              --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 5: Definir la función
--    símbolosPropFórm :: Prop -> [Prop]
-- tal que (símbolosPropFórm f) es el conjunto formado por todos los
-- símbolos proposicionales que aparecen en f. Por ejemplo,
--    símbolosPropFórm (p /\ q --> p)  ==> [p,q]
-- ----------------------------------------------------------------------

símbolosPropFórm :: Prop -> [Prop]
símbolosPropFórm (Atom f)   = [(Atom f)]
símbolosPropFórm (Neg f)    = símbolosPropFórm f
símbolosPropFórm (Conj f g) = símbolosPropFórm f `union` símbolosPropFórm g
símbolosPropFórm (Disj f g) = símbolosPropFórm f `union` símbolosPropFórm g
símbolosPropFórm (Impl f g) = símbolosPropFórm f `union` símbolosPropFórm g
símbolosPropFórm (Equi f g) = símbolosPropFórm f `union` símbolosPropFórm g
```

```
-- ----------------------------------------------------------------------
-- Interpretaciones                                                     --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 6: Definir el tipo de datos Interpretación para
-- representar las interpretaciones como listas de fórmulas atómicas.
-- ----------------------------------------------------------------------

type Interpretación = [Prop]

-- ----------------------------------------------------------------------
-- Significado de una fórmula en una interpretación                     --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 7: Definir la función
--    significado :: Prop -> Interpretación -> Bool
-- tal que (significado f i) es el significado de f en i. Por ejemplo,
--    significado ((p \/ q) /\ ((no q) \/ r)) [r]    ==>  False
--    significado ((p \/ q) /\ ((no q) \/ r)) [p,r]  ==>  True
-- ----------------------------------------------------------------------

significado :: Prop -> Interpretación -> Bool
significado (Atom f)   i = (Atom f) `elem` i
significado (Neg f)    i = not (significado f i)
significado (Conj f g) i = (significado f i) && (significado g i)
significado (Disj f g) i = (significado f i) || (significado g i)
significado (Impl f g) i = significado (Disj (Neg f) g) i
significado (Equi f g) i = significado (Conj (Impl f g) (Impl g f)) i

-- ----------------------------------------------------------------------
-- Interpretaciones de una fórmula                                      --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 8: Definir la función
--    subconjuntos :: [a] -> [[a]]
-- tal que (subconjuntos x) es la lista de los subconjuntos de x. Por
-- ejmplo,
```

```
--    subconjuntos "abc"  ==>  ["abc","ab","ac","a","bc","b","c",""]
-- ----------------------------------------------------------------------

subconjuntos :: [a] -> [[a]]
subconjuntos []     = [[]]
subconjuntos (x:xs) = [x:ys | ys <- xss] ++ xss
                      where xss = subconjuntos xs

-- ----------------------------------------------------------------------
-- Ejercicio 9: Definir la función
--    interpretacionesFórm :: Prop -> [Interpretación]
-- tal que (interpretacionesFórm f) es la lista de todas las
-- interpretaciones de f. Por ejemplo,
--    interpretacionesFórm (p /\ q --> p)  ==>  [[p,q],[p],[q],[]]
-- ----------------------------------------------------------------------

interpretacionesFórm :: Prop -> [Interpretación]
interpretacionesFórm f = subconjuntos (símbolosPropFórm f)

-- ----------------------------------------------------------------------
-- Modelos de fórmulas                                                  --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 10: Definir la función
--    esModeloFórmula :: Interpretación -> Prop -> Bool
-- tal que (esModeloFórmula i f) se verifica si i es un modelo de f. Por
-- ejemplo,
--    esModeloFórmula [r]   ((p \/ q) /\ ((no q) \/ r))    ==>  False
--    esModeloFórmula [p,r] ((p \/ q) /\ ((no q) \/ r))    ==>  True
-- ----------------------------------------------------------------------

esModeloFórmula :: Interpretación -> Prop -> Bool
esModeloFórmula i f = significado f i

-- ----------------------------------------------------------------------
-- Ejercicio 11: Definir la función
--    modelosFórmula :: Prop -> [Interpretación]
-- tal que (modelosFórmula f) es la lista de todas las interpretaciones
-- de f que son modelo de F. Por ejemplo,
```

```
--    modelosFórmula ((p \/ q) /\ ((no q) \/ r))
--    ==> [[p,q,r],[p,r],[p],[q,r]]
-- ----------------------------------------------------------------------

modelosFórmula :: Prop -> [Interpretación]
modelosFórmula f =
    [i | i <- interpretacionesFórm f,
         esModeloFórmula i f]

-- ----------------------------------------------------------------------
-- Fórmulas válidas, satisfacibles e insatisfacibles                  --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 12: Definir la función
--    esVálida :: Prop -> Bool
-- tal que (esVálida f) se verifica si f es válida. Por ejemplo,
--    esVálida (p --> p)                ==>  True
--    esVálida (p --> q)                ==>  False
--    esVálida ((p --> q) \/ (q --> p))  ==>  True
-- ----------------------------------------------------------------------

esVálida :: Prop -> Bool
esVálida f =
    modelosFórmula f == interpretacionesFórm f

-- ----------------------------------------------------------------------
-- Ejercicio 13: Definir la función
--    esInsatisfacible :: Prop -> Bool
-- tal que (esInsatisfacible f) se verifica si f es insatisfacible. Por
-- ejemplo,
--    esInsatisfacible (p /\ (no p))            ==>  True
--    esInsatisfacible ((p --> q) /\ (q --> r))  ==>  False
-- ----------------------------------------------------------------------

esInsatisfacible :: Prop -> Bool
esInsatisfacible f =
    modelosFórmula f == []

-- ----------------------------------------------------------------------
```

```
-- Ejercicio 14: Definir la función
--    esSatisfacible :: Prop -> Bool
-- tal que (esSatisfacible f) se verifica si f es satisfacible. Por
-- ejemplo,
--    esSatisfacible (p /\ (no p))             ==>  False
--    esSatisfacible ((p --> q) /\ (q --> r))  ==>  True
-- ----------------------------------------------------------------------

esSatisfacible :: Prop -> Bool
esSatisfacible f =
    modelosFórmula f /= []

-- ----------------------------------------------------------------------
-- Símbolos proposicionales de un conjunto de fórmulas                --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 15: Definir la función
--    uniónGeneral :: Eq a => [[a]] -> [a]
-- tal que (uniónGeneral x) es la unión de los conjuntos de la lista de
-- conjuntos x. Por ejemplo,
--    uniónGeneral []                 ==>  []
--    uniónGeneral [[1]]              ==>  [1]
--    uniónGeneral [[1],[1,2],[2,3]]  ==>  [1,2,3]
-- ----------------------------------------------------------------------

uniónGeneral :: Eq a => [[a]] -> [a]
uniónGeneral []     = []
uniónGeneral (x:xs) = x `union` uniónGeneral xs

-- ----------------------------------------------------------------------
-- Ejercicio 16: Definir la función
--    símbolosPropConj :: [Prop] -> [Prop]
-- tal que (símbolosPropConj s) es el conjunto de los símbolos
-- proposiciones de s. Por ejemplo,
--    símbolosPropConj [p /\ q --> r, p --> s]  ==>  [p,q,r,s]
-- ----------------------------------------------------------------------

símbolosPropConj :: [Prop] -> [Prop]
símbolosPropConj s
```

```
    = uniónGeneral [símbolosPropFórm f | f <- s]

-- ---------------------------------------------------------------------
-- Interpretaciones de un conjunto de fórmulas                         --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 17: Definir la función
--    interpretacionesConjunto :: [Prop] -> [Interpretación]
-- tal que (interpretacionesConjunto s) es la lista de las
-- interpretaciones de s. Por ejemplo,
--    interpretacionesConjunto [p --> q, q --> r]
--    ==> [[p,q,r],[p,q],[p,r],[p],[q,r],[q],[r],[]]
-- ---------------------------------------------------------------------

interpretacionesConjunto :: [Prop] -> [Interpretación]
interpretacionesConjunto s =
    subconjuntos (símbolosPropConj s)

-- ---------------------------------------------------------------------
-- Modelos de conjuntos de fórmulas                                    --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 18: Definir la función
--    esModeloConjunto :: Interpretación -> [Prop] -> Bool
-- tal que (esModeloConjunto i s) se verifica si i es modelo de s. Por
-- ejemplo,
--    esModeloConjunto [p,r] [(p \/ q) /\ ((no q) \/ r), q --> r]
--    ==> True
--    esModeloConjunto [p,r] [(p \/ q) /\ ((no q) \/ r), r --> q]
--    ==> False
-- ---------------------------------------------------------------------

esModeloConjunto :: Interpretación -> [Prop] -> Bool
esModeloConjunto i s =
    and [esModeloFórmula i f | f <- s]

-- ---------------------------------------------------------------------
-- Ejercicio 19: Definir la función
```

```
--    modelosConjunto :: [Prop] -> [Interpretación]
-- tal que (modelosConjunto s) es la lista de modelos del conjunto
-- s. Por ejemplo,
--    modelosConjunto [(p \/ q) /\ ((no q) \/ r), q --> r]
--    ==> [[p,q,r],[p,r],[p],[q,r]]
--    modelosConjunto [(p \/ q) /\ ((no q) \/ r), r --> q]
--    ==> [[p,q,r],[p],[q,r]]
-- ------------------------------------------------------------------

modelosConjunto :: [Prop] -> [Interpretación]
modelosConjunto s =
    [i | i <- interpretacionesConjunto s,
         esModeloConjunto i s]

-- ------------------------------------------------------------------
-- Conjuntos consistentes e inconsistentes de fórmulas              --
-- ------------------------------------------------------------------

-- ------------------------------------------------------------------
-- Ejercicio 20: Definir la función
--    esConsistente :: [Prop] -> Bool
-- tal que (esConsistente s) se verifica si s es consistente. Por
-- ejemplo,
--    esConsistente [(p \/ q) /\ ((no q) \/ r), p --> r]
--    ==> True
--    esConsistente [(p \/ q) /\ ((no q) \/ r), p --> r, no r]
--    ==> False
-- ------------------------------------------------------------------

esConsistente :: [Prop] -> Bool
esConsistente s =
    modelosConjunto s /= []

-- ------------------------------------------------------------------
-- Ejercicio 21: Definir la función
--    esInconsistente :: [Prop] -> Bool
-- tal que (esInconsistente s) se verifica si s es inconsistente. Por
-- ejemplo,
--    esInconsistente [(p \/ q) /\ ((no q) \/ r), p --> r]
--    ==> False
```

```
--    esInconsistente [(p \/ q) /\ ((no q) \/ r), p --> r, no r]
--    ==> True
-- ----------------------------------------------------------------------

esInconsistente :: [Prop] -> Bool
esInconsistente s =
    modelosConjunto s == []

-- ----------------------------------------------------------------------
-- Consecuencia lógica                                                  --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 22: Definir la función
--    esConsecuencia :: [Prop] -> Prop -> Bool
-- tal que (esConsecuencia s f) se verifica si f es consecuencia de
-- s. Por ejemplo,
--    esConsecuencia [p --> q, q --> r] (p --> r)  ==>  True
--    esConsecuencia [p] (p /\ q)                  ==>  False
-- ----------------------------------------------------------------------

esConsecuencia :: [Prop] -> Prop -> Bool
esConsecuencia s f =
    null [i | i <- interpretacionesConjunto (f:s),
              esModeloConjunto i s,
              not (esModeloFórmula i f)]

-- ----------------------------------------------------------------------
-- Verificación                                                         --
-- ----------------------------------------------------------------------

ejemplosSintaxisSemantica :: [Test]
ejemplosSintaxisSemantica =
    ["simbolosPropForm ej 1" ~:
     símbolosPropFórm (p /\ q --> p)
     ==> [p,q],
     "significado ej 1" ~:
     significado ((p \/ q) /\ ((no q) \/ r)) [r]
     ==> False,
     "significado ej 2" ~:
```

```
significado ((p \/ q) /\ ((no q) \/ r)) [p,r]
==> True,
"interpretacionesForm ej 1" ~:
interpretacionesFórm (p /\ q --> p)
==> [[p,q],[p],[q],[]],
"esModeloFormula ej 1" ~:
esModeloFórmula [r]   ((p \/ q) /\ ((no q) \/ r))
==> False,
"esModeloFormula ej 2" ~:
esModeloFórmula [p,r] ((p \/ q) /\ ((no q) \/ r))
==> True,
"modelosFormula ej 1" ~:
modelosFórmula ((p \/ q) /\ ((no q) \/ r))
==> [[p,q,r],[p,r],[p],[q,r]],
"esValida ej 1" ~:
esVálida (p --> p)
==> True,
"esValida ej 2" ~:
esVálida (p --> q)
==> False,
"esValida ej 3" ~:
esVálida ((p --> q) \/ (q --> p))
==> True,
"esInsatisfacible ej 1" ~:
esInsatisfacible (p /\ (no p))
==> True,
"esInsatisfacible ej 2" ~:
esInsatisfacible ((p --> q) /\ (q --> r))
==> False,
"esSatisfacible ej 1" ~:
esSatisfacible (p /\ (no p))
==> False,
"esSatisfacible ej 2" ~:
esSatisfacible ((p --> q) /\ (q --> r))
==> True,
"unionGeneral ej 1" ~:
uniónGeneral [[1]]
==> [1],
"unionGeneral ej 2" ~:
uniónGeneral [[1],[1,2],[2,3]]
```

```
    ==> [1,2,3],
    "simbolosPropConj ej 1" ~:
    símbolosPropConj [p /\ q --> r, p --> s]
    ==> [p,q,r,s],
    "interpretacionesConjunto ej 1" ~:
    interpretacionesConjunto [p --> q, q --> r]
    ==> [[p,q,r],[p,q],[p,r],[p],[q,r],[q],[r],[]],
    "esModeloConjunto ej 1" ~:
    esModeloConjunto [p,r] [(p \/ q) /\ ((no q) \/ r), q --> r]
    ==> True,
    "esModeloConjunto ej 2" ~:
    esModeloConjunto [p,r] [(p \/ q) /\ ((no q) \/ r), r --> q]
    ==> False,
    "modelosConjunto ej 1" ~:
    modelosConjunto [(p \/ q) /\ ((no q) \/ r), q --> r]
    ==> [[p,q,r],[p,r],[p],[q,r]],
    "modelosConjunto ej 2" ~:
    modelosConjunto [(p \/ q) /\ ((no q) \/ r), r --> q]
    ==> [[p,q,r],[p],[q,r]],
    "esConsistente ej 1" ~:
    esConsistente [(p \/ q) /\ ((no q) \/ r), p --> r]
    ==> True,
    "esConsistente ej 2" ~:
    esConsistente [(p \/ q) /\ ((no q) \/ r), p --> r, no r]
    ==> False,
    "esInconsistente ej 1" ~:
    esInconsistente [(p \/ q) /\ ((no q) \/ r), p --> r]
    ==> False,
    "esInconsistente ej 2" ~:
    esInconsistente [(p \/ q) /\ ((no q) \/ r), p --> r, no r]
    ==> True,
    "esConsecuencia ej 1" ~:
    esConsecuencia [p --> q, q --> r] (p --> r)
    ==> True,
    "esConsecuencia ej 2" ~:
    esConsecuencia [p] (p /\ q)
    ==> False
   ]

verificaSintaxisSemantica :: IO Counts
```

```
verificaSintaxisSemantica =
    runTestTT (test ejemplosSintaxisSemantica)

-- SintaxisSemantica> verificaSintaxisSemántica
-- Cases: 28  Tried: 28  Errors: 0  Failures: 0
```

Capítulo 2

Formas normales y cláusulas

```
module FormasNormales where

-- ----------------------------------------------------------------------
-- Librería suxiliares                                                  --
-- ----------------------------------------------------------------------

import SintaxisSemantica
import Data.List
import Test.HUnit
import Verificacion

-- ----------------------------------------------------------------------
-- Equivalencia lógica                                                  --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 1: Definir la función
--    esEquivalente :: Prop -> Prop -> Bool
-- tal que (esEquivalente f g) se verifica si f y g son
-- equivalentes. Por ejemplo,
--    esEquivalente (p <--> q) ((p --> q) /\ (q --> p))  ==>  True
--    esEquivalente (p --> q)  ((no p) \/ q)             ==>  True
--    esEquivalente (p /\ q)   (no ((no p) \/ (no q)))   ==>  True
--    esEquivalente (p \/ q)   (no ((no p) /\ (no q)))   ==>  True
-- ----------------------------------------------------------------------

esEquivalente :: Prop -> Prop -> Bool
```

```
esEquivalente f g =
    esVálida (Equi f g)

-- ------------------------------------------------------------------
-- Transformación a forma normal negativa                           --
-- ------------------------------------------------------------------

-- ------------------------------------------------------------------
-- Ejercicio 2: Definir la función
--    eliminaEquivalencias :: Prop -> Prop
-- tal que (eliminaEquivalencias f) es una fórmula equivalente a f sin
-- signos de equivalencia. Por ejemplo,
--    eliminaEquivalencias (p <--> q)
--    ==> ((p --> q) /\ (q --> p))
--    eliminaEquivalencias ((p <--> q) /\ (q <--> r))
--    ==> (((p --> q) /\ (q --> p)) /\ ((q --> r) /\ (r --> q)))
-- ------------------------------------------------------------------

eliminaEquivalencias :: Prop -> Prop
eliminaEquivalencias (Atom f)   =
    (Atom f)
eliminaEquivalencias (Neg f)    =
    Neg (eliminaEquivalencias f)
eliminaEquivalencias (Conj f g) =
    Conj (eliminaEquivalencias f) (eliminaEquivalencias g)
eliminaEquivalencias (Disj f g) =
    Disj (eliminaEquivalencias f) (eliminaEquivalencias g)
eliminaEquivalencias (Impl f g) =
    Impl (eliminaEquivalencias f) (eliminaEquivalencias g)
eliminaEquivalencias (Equi f g) =
    Conj (Impl (eliminaEquivalencias f) (eliminaEquivalencias g))
         (Impl (eliminaEquivalencias g) (eliminaEquivalencias f))

-- ------------------------------------------------------------------
-- Ejercicio 3: Definir la función
--    eliminaImplicaciones :: Prop -> Prop
-- tal que (eliminaImplicaciones f) es una fórmula equivalente a f sin
-- signos de implicación. Por ejemplo,
--    eliminaImplicaciones (p --> q)
--    ==> (no p \/ q)
```

```
--    eliminaImplicaciones (eliminaEquivalencias (p <--> q))
--    ==> ((no p \/ q) /\ (no q \/ p))
-- Nota: Se supone que f no tiene signos de equivalencia.
-- ----------------------------------------------------------------------

eliminaImplicaciones :: Prop -> Prop
eliminaImplicaciones (Atom f)   =
    (Atom f)
eliminaImplicaciones (Neg f)    =
    Neg (eliminaImplicaciones f)
eliminaImplicaciones (Conj f g) =
    Conj (eliminaImplicaciones f) (eliminaImplicaciones g)
eliminaImplicaciones (Disj f g) =
    Disj (eliminaImplicaciones f) (eliminaImplicaciones g)
eliminaImplicaciones (Impl f g) =
    Disj (Neg (eliminaImplicaciones f)) (eliminaImplicaciones g)

-- ----------------------------------------------------------------------
-- Ejercicio 4: Definir la función
--    interiorizaNegación :: Prop -> Prop
-- tal que (interiorizaNegación f) es una fórmula equivalente a f donde
-- las negaciones se aplican sólo a fórmulas atómicas. Por ejemplo,
--    interiorizaNegación (no (no p))         ==>  p
--    interiorizaNegación (no (p /\ q))       ==>  (no p \/ no q)
--    interiorizaNegación (no (p \/ q))       ==>  (no p /\ no q)
--    interiorizaNegación (no (no (p \/ q)))  ==>  (p \/ q)
--    interiorizaNegación (no ((no p) \/ q))  ==>  (p /\ no q)
-- Nota: Se supone que f no tiene equivalencias ni implicaciones.
-- ----------------------------------------------------------------------

interiorizaNegación :: Prop -> Prop
interiorizaNegación (Atom f)   =
    (Atom f)
interiorizaNegación (Neg f)    =
    interiorizaNegaciónAux f
interiorizaNegación (Conj f g) =
    Conj (interiorizaNegación f) (interiorizaNegación g)
interiorizaNegación (Disj f g) =
    Disj (interiorizaNegación f) (interiorizaNegación g)
```

```
interiorizaNegaciónAux :: Prop -> Prop
interiorizaNegaciónAux (Atom f)   =
    Neg (Atom f)
interiorizaNegaciónAux (Neg f)    =
    interiorizaNegación f
interiorizaNegaciónAux (Conj f g) =
    Disj (interiorizaNegaciónAux f) (interiorizaNegaciónAux g)
interiorizaNegaciónAux (Disj f g) =
    Conj (interiorizaNegaciónAux f) (interiorizaNegaciónAux g)

-- ---------------------------------------------------------------------
-- Ejercicio 5: Definir la función
--    formaNormalNegativa :: Prop -> Prop
-- tal que (formaNormalNegativa f) es una fórmula equivalente a f en
-- forma normal negativa. Por ejemplo,
--    formaNormalNegativa (p <--> q)
--    ==> ((no p \/ q) /\ (no q \/ p))
--    formaNormalNegativa ((p \/ (no q)) --> r)
--    ==> ((no p /\ q) \/ r)
--    formaNormalNegativa ((p /\ (q --> r)) --> s)
--    ==> ((no p \/ (q /\ no r)) \/ s)
-- ---------------------------------------------------------------------

formaNormalNegativa :: Prop -> Prop
formaNormalNegativa f =
    interiorizaNegación (eliminaImplicaciones (eliminaEquivalencias f))

-- ---------------------------------------------------------------------
-- Literales                                                           --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 6: Definir la función
--    literal :: Prop -> Bool
-- tal que (literal f) se verifica si la fórmula F es un literal. Por
-- ejemplo,
--    literal p              ==>  True
--    literal (no p)         ==>  True
--    literal (no (p --> q))  ==>  False
-- ---------------------------------------------------------------------
```

```
literal :: Prop -> Bool
literal (Atom f)       = True
literal (Neg (Atom f)) = True
literal _              = False

-- ----------------------------------------------------------------------
-- Ejercicio 7: Definir el tipo de dato Literal como sinónimo de
-- fórmula.
-- ----------------------------------------------------------------------

type Literal = Prop

-- ----------------------------------------------------------------------
-- Ejercicio 8: Definir la función
--    complementario :: Literal -> Literal
-- tal que (complementario l) es el complementario de l. Por ejemplo,
--    complementario p       ==>  no p
--    complementario (no p)  ==>  p
-- ----------------------------------------------------------------------

complementario :: Literal -> Literal
complementario (Atom f)       = Neg (Atom f)
complementario (Neg (Atom f)) = Atom f

-- ----------------------------------------------------------------------
-- Ejercicio 9: Definir la función
--    literalesFórmulaFNN :: Prop -> [Literal]
-- tal que (literalesFórmulaFNN f) es el conjunto de los literales de la
-- fórmula en forma normal negativa f.
--    literalesFórmulaFNN (p \/ ((no q) \/ r))  ==>  [p,no q,r]
--    literalesFórmulaFNN p                     ==>  [p]
--    literalesFórmulaFNN (no p)                ==>  [no p]
-- ----------------------------------------------------------------------

literalesFórmulaFNN :: Prop -> [Literal]
literalesFórmulaFNN (Disj f g) =
    (literalesFórmulaFNN f) `union` (literalesFórmulaFNN g)
literalesFórmulaFNN (Conj f g) =
    (literalesFórmulaFNN f) `union` (literalesFórmulaFNN g)
```

```
literalesFórmulaFNN f          = [f]

-- ------------------------------------------------------------------
-- Transformación a forma normal conjuntiva                         --
-- ------------------------------------------------------------------

-- ------------------------------------------------------------------
-- Ejercicio 10: Definir la función
--    interiorizaDisyunción :: Prop -> Prop
-- tal que (interiorizaDisyunción f) es una fórmula equivalente a f
-- donde las disyunciones sólo se aplica a disyunciones o literales. Por
-- ejemplo,
--    interiorizaDisyunción (p \/ (q /\ r))  ==>  ((p \/ q) /\ (p \/ r))
--    interiorizaDisyunción ((p /\ q) \/ r)  ==>  ((p \/ r) /\ (q \/ r))
-- Nota: Se supone que f está en forma normal negativa.
-- ------------------------------------------------------------------

interiorizaDisyunción :: Prop -> Prop
interiorizaDisyunción (Disj (Conj f1 f2) g) =
    interiorizaDisyunción
    (Conj (Disj (interiorizaDisyunción f1) (interiorizaDisyunción g))
          (Disj (interiorizaDisyunción f2) (interiorizaDisyunción g)))
interiorizaDisyunción (Disj f (Conj g1 g2)) =
    interiorizaDisyunción
    (Conj (Disj (interiorizaDisyunción f) (interiorizaDisyunción g1))
          (Disj (interiorizaDisyunción f) (interiorizaDisyunción g2)))
interiorizaDisyunción (Conj f g) =
    Conj (interiorizaDisyunción f) (interiorizaDisyunción g)
interiorizaDisyunción f = f

-- ------------------------------------------------------------------
-- Ejercicio 11: Definir la función
--    formaNormalConjuntiva :: Prop -> Prop
-- tal que (formaNormalConjuntiva f) es una fórmula equivalente a f en
-- forma normal conjuntiva. Por ejemplo,
--    formaNormalConjuntiva (p /\ (q --> r))
--    ==> (p /\ (no q \/ r))
--    formaNormalConjuntiva (no (p /\ (q --> r)))
--    ==> ((no p \/ q) /\ (no p \/ no r))
--    formaNormalConjuntiva (no(p <--> r))
```

```
--    ==> (((p \/ r) /\ (p \/ no p)) /\ ((no r \/ r) /\ (no r \/ no p)))
-- ----------------------------------------------------------------------

formaNormalConjuntiva :: Prop -> Prop
formaNormalConjuntiva f =
    interiorizaDisyunción (formaNormalNegativa f)

-- ----------------------------------------------------------------------
-- Transformación a forma normal disyuntiva                             --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 12: Definir la función
--    interiorizaConjunción :: Prop -> Prop
-- tal que (interiorizaConjunción f) es una fórmula equivalente a f
-- donde las conjunciones sólo se aplica a conjunciones o literales. Por
-- ejemplo,
--    interiorizaConjunción (p /\ (q \/ r))  ==>  ((p /\ q) \/ (p /\ r))
--    interiorizaConjunción ((p \/ q) /\ r)  ==>  ((p /\ r) \/ (q /\ r))
-- Nota: Se supone que f está en forma normal negativa.
-- ----------------------------------------------------------------------

interiorizaConjunción :: Prop -> Prop
interiorizaConjunción (Conj (Disj f1 f2) g) =
    interiorizaConjunción
    (Disj (Conj (interiorizaConjunción f1) (interiorizaConjunción g))
          (Conj (interiorizaConjunción f2) (interiorizaConjunción g)))
interiorizaConjunción (Conj f (Disj g1 g2)) =
    interiorizaConjunción
    (Disj (Conj (interiorizaConjunción f) (interiorizaConjunción g1))
          (Conj (interiorizaConjunción f) (interiorizaConjunción g2)))
interiorizaConjunción (Disj f g) =
    Disj (interiorizaConjunción f) (interiorizaConjunción g)
interiorizaConjunción f = f

-- ----------------------------------------------------------------------
-- Ejercicio 13: Definir la función
--    formaNormalDisyuntiva :: Prop -> Prop
-- tal que (formaNormalDisyuntiva f) es una fórmula equivalente a f en
-- forma normal disyuntiva. Por ejemplo,
```

```
--    formaNormalDisyuntiva (p /\ (q --> r))
--    ==> ((p /\ no q) \/ (p /\ r))
--    formaNormalDisyuntiva (no (p /\ (q --> r)))
--    ==> (no p \/ (q /\ no r))
-- ----------------------------------------------------------------------

formaNormalDisyuntiva :: Prop -> Prop
formaNormalDisyuntiva f =
    interiorizaConjunción (formaNormalNegativa f)

-- ----------------------------------------------------------------------
-- Verificación                                                         --
-- ----------------------------------------------------------------------

ejemplosFormasNormales :: [Test]
ejemplosFormasNormales =
    [ "esEquivalente ej 1" ~:
      esEquivalente (p <--> q) ((p --> q) /\ (q --> p))
      ==> True,
      "esEquivalente ej 2" ~:
      esEquivalente (p --> q)  ((no p) \/ q)
      ==> True,
      "esEquivalente ej 3" ~:
      esEquivalente (p /\ q)   (no ((no p) \/ (no q)))
      ==> True,
      "esEquivalente ej 4" ~:
      esEquivalente (p \/ q)   (no ((no p) /\ (no q)))
      ==> True,
      "eliminaEquivalencias ej 1" ~:
      eliminaEquivalencias (p <--> q)
      ==>((p --> q) /\ (q --> p)),
      "eliminaEquivalencias ej 2" ~:
      eliminaEquivalencias ((p <--> q) /\ (q <--> r))
      ==>(((p --> q) /\ (q --> p)) /\ ((q --> r) /\ (r --> q))),
      "eliminaImplicaciones ej 1" ~:
      eliminaImplicaciones (p --> q)
      ==>(no p \/ q),
      "eliminaImplicaciones ej 2" ~:
      eliminaImplicaciones (eliminaEquivalencias (p <--> q))
      ==>((no p \/ q) /\ (no q \/ p)),
```

```
"interiorizaNegacion ej 1" ~:
interiorizaNegación (no (no p))
==> p,
"interiorizaNegacion ej 2" ~:
interiorizaNegación (no (p /\ q))
==> (no p \/ no q),
"interiorizaNegacion ej 3" ~:
interiorizaNegación (no (p \/ q))
==> (no p /\ no q),
"interiorizaNegacion ej 4" ~:
interiorizaNegación (no (no (p \/ q)))
==> (p \/ q),
"interiorizaNegacion ej 5" ~:
interiorizaNegación (no ((no p) \/ q))
==> (p /\ no q),
"formaNormalNegativa ej 1" ~:
formaNormalNegativa (p <--> q)
==>((no p \/ q) /\ (no q \/ p)),
"formaNormalNegativa ej 2" ~:
formaNormalNegativa ((p \/ (no q)) --> r)
==>((no p /\ q) \/ r),
"formaNormalNegativa ej 3" ~:
formaNormalNegativa ((p /\ (q --> r)) --> s)
==>((no p \/ (q /\ no r)) \/ s),
"literal ej 1" ~:
literal p
==> True,
"literal ej 2" ~:
literal (no p)
==> True,
"literal ej 2" ~:
literal (no (p --> q))
==> False,
"complementario ej 1" ~:
complementario p
==> no p,
"complementario ej 2" ~:
complementario (no p)
==> p,
"interiorizaDisyuncion ej 1" ~:
```

```
      interiorizaDisyunción (p \/ (q /\ r))
      ==> ((p \/ q) /\ (p \/ r)),
      "interiorizaDisyuncion ej 2" ~:
      interiorizaDisyunción ((p /\ q) \/ r)
      ==> ((p \/ r) /\ (q \/ r)),
      "formaNormalConjuntiva ej 1" ~:
      formaNormalConjuntiva (p /\ (q --> r))
      ==> (p /\ (no q \/ r)),
      "formaNormalConjuntiva ej 2" ~:
      formaNormalConjuntiva (no (p /\ (q --> r)))
      ==> ((no p \/ q) /\ (no p \/ no r)),
      "formaNormalConjuntiva ej 3" ~:
      formaNormalConjuntiva (no(p <--> r))
      ==> (((p \/ r) /\ (p \/ no p)) /\ ((no r \/ r) /\ (no r \/ no p))),
      "interiorizaConjuncion ej 1" ~:
      interiorizaConjunción (p /\ (q \/ r))
      ==> ((p /\ q) \/ (p /\ r)),
      "interiorizaConjuncion ej 2" ~:
      interiorizaConjunción ((p \/ q) /\ r)
      ==> ((p /\ r) \/ (q /\ r)),
      "formaNormalDisyuntiva ej 1" ~:
      formaNormalDisyuntiva (p /\ (q --> r))
      ==> ((p /\ no q) \/ (p /\ r)),
      "formaNormalDisyuntiva ej 2" ~:
      formaNormalDisyuntiva (no (p /\ (q --> r)))
      ==> (no p \/ (q /\ no r))
    ]

verificaFormasNormales :: IO Counts
verificaFormasNormales =
    runTestTT (test ejemplosFormasNormales)

-- FormasNormales> verificaFormasNormales
-- Cases: 30  Tried: 30  Errors: 0  Failures: 0
```

Capítulo 3

Cláusulas

```
module Clausulas where

-- ----------------------------------------------------------------------
-- Librería suxiliares                                                  --
-- ----------------------------------------------------------------------

import SintaxisSemantica
import FormasNormales
import Data.List
import Test.HUnit
import Verificacion

-- ----------------------------------------------------------------------
-- Cláusulas                                                            --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 1: Definir el tipo de datos Cláusula como una lista de
-- literales.
-- ----------------------------------------------------------------------

type Cláusula = [Literal]

-- ----------------------------------------------------------------------
-- Ejercicio 2: Definir la función
--    cláusula :: Prop -> Cláusula
-- tal que (cláusula f) es la cláusula de la fórmula-clausal f. Por
```

```
-- ejemplo,
--    cláusula p                                  ==>  [p]
--    cláusula (no p)                             ==>  [no p]
--    cláusula (((no p) \/ r) \/ ((no p) \/ q))   ==>  [q,r,no p]
-- ----------------------------------------------------------------------

cláusula :: Prop -> Cláusula
cláusula f
    | literal f     = [f]
cláusula (Disj f g) = sort ((cláusula f) `union` (cláusula g))

-- ----------------------------------------------------------------------
-- Ejercicio 3: Definir la función
--    cláusulasFNC :: Prop -> [Cláusula]
-- tal que (cláusulasFNC f) es el conjunto de cláusulas de la fórmula en
-- forma normal conjuntiva f. Por ejmplo,
--    cláusulasFNC (p /\ ((no q) \/ r))
--    ==> [[p],[r,-q]]
--    cláusulasFNC (((no p) \/ q) /\ ((no p) \/ (no r)))
--    ==> [[q,-p],[-p,-r]]
-- ----------------------------------------------------------------------

cláusulasFNC :: Prop -> [Cláusula]
cláusulasFNC (Conj f g) =
    (cláusulasFNC f) `union`  (cláusulasFNC g)
cláusulasFNC f =
    [cláusula f]

-- ----------------------------------------------------------------------
-- Ejercicio 4: Definir la función
--    cláusulas :: Prop -> [Cláusula]
-- tal que (cláusulas f) es un conjunto de cláusulas equivalente a
-- f. Por ejemplo,
--    cláusulas (p /\ (q --> r))
--    ==> [[p],[r,-q]]
--    cláusulas (no (p /\ (q --> r)))
--    ==> [[q,-p],[-p,-r]]
--    cláusulas (no(p <--> r))
--    ==> [[p,r],[p,no p],[r,no r],[no p,no r]]
-- ----------------------------------------------------------------------
```

```
cláusulas :: Prop -> [Cláusula]
cláusulas f =
    cláusulasFNC (formaNormalConjuntiva f)

-- ------------------------------------------------------------------
-- Cláusulas de un conjunto de fórmulas                             --
-- ------------------------------------------------------------------

-- ------------------------------------------------------------------
-- Ejercicio 5: Definir la función
--    cláusulasConjunto :: [Prop] -> [Cláusula]
-- tal que (cláusulasConjunto s) es un conjunto de cláusulas equivalente
-- a s. Por ejemplo,
--    cláusulasConjunto [p --> q, q --> r]   ==>  [[q,-p],[r,-q]]
--    cláusulasConjunto [p --> q, q <--> p]  ==>  [[q,-p],[p,-q]]
-- ------------------------------------------------------------------

cláusulasConjunto :: [Prop] -> [Cláusula]
cláusulasConjunto s =
    unióngeneral [cláusulas f | f <- s]

-- ------------------------------------------------------------------
-- Símbolos proposicionales de una cláusula                         --
-- ------------------------------------------------------------------

-- ------------------------------------------------------------------
-- Ejercicio 6: Definir la función
--    símbolosProposicionalesCláusula :: Cláusula -> [Prop]
-- tal que (símbolosProposicionalesCláusula c) es el conjunto de los
-- símbolos proposicionales de c. Por ejemplo,
--    símbolosProposicionalesCláusula [p, q, no p]  ==>  [p,q]
-- ------------------------------------------------------------------

símbolosProposicionalesCláusula :: Cláusula -> [Prop]
símbolosProposicionalesCláusula = símbolosPropConj

-- ------------------------------------------------------------------
-- Símbolos proposicionales de un conjunto de cláusulas             --
-- ------------------------------------------------------------------
```

```
-- ---------------------------------------------------------------------
-- Ejercicio 7: Definir la función
--    símbolosProposicionalesConjuntoCláusula :: [Cláusula] -> [Prop]
-- tal que (símbolosProposicionalesConjuntoCláusula s) es el conjunto de los
-- símbolos proposicionales de s. Por ejemplo,
--    símbolosProposicionalesConjuntoCláusula [[p, q],[no q, r]]
--    ==> [p,q,r]
-- ---------------------------------------------------------------------

símbolosProposicionalesConjuntoCláusula :: [Cláusula] -> [Prop]
símbolosProposicionalesConjuntoCláusula s =
    unión General [símbolosProposicionalesCláusula c | c <- s]

-- ---------------------------------------------------------------------
-- Interpretaciones de una cláusula                                    --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 8: Definir la función
--    interpretacionesCláusula :: Cláusula -> [Interpretación]
-- tal que (interpretacionesCláusula c) es el conjunto de
-- interpretaciones de c. Por ejemplo,
--    interpretacionesCláusula [p, q, no p]  ==>  [[p,q],[p],[q],[]]
--    interpretacionesCláusula []            ==>  [[]]
-- ---------------------------------------------------------------------

interpretacionesCláusula :: Cláusula -> [Interpretación]
interpretacionesCláusula c =
    subconjuntos (símbolosProposicionalesCláusula c)

-- ---------------------------------------------------------------------
-- Interpretaciones de un conjunto de cláusulas                        --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 9: Definir la función
--    interpretacionesConjuntoCláusula :: [Cláusula] -> [Interpretación]
-- tal que (interpretacionesConjuntoCláusula s) es el conjunto de
-- interpretaciones de s. Por ejemplo,
```

```
--    interpretacionesConjuntoCláusula [[p, no q],[no p, q]]
--    ==> [[p,q],[p],[q],[]]
--    interpretacionesConjuntoCláusula []
--    ==> [[]]
-- ---------------------------------------------------------------------

interpretacionesConjuntoCláusula :: [Cláusula] -> [Interpretación]
interpretacionesConjuntoCláusula c =
    subconjuntos (símbolosProposicionalesConjuntoCláusula c)

-- ---------------------------------------------------------------------
-- Modelos de cláusulas                                                --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 10: Definir la función
--    esModeloLiteral :: Interpretación -> Literal -> Bool
-- tal que (esModeloLiteral i l) se verifica si i es modelo de l. Por
-- ejemplo,
--    esModeloLiteral [p,r] p       ==>  True
--    esModeloLiteral [p,r] q       ==>  False
--    esModeloLiteral [p,r] (no p)  ==>  False
--    esModeloLiteral [p,r] (no q)  ==>  True
-- ---------------------------------------------------------------------

esModeloLiteral :: Interpretación -> Literal -> Bool
esModeloLiteral i (Atom s)       = elem (Atom s) i
esModeloLiteral i (Neg (Atom s)) = notElem (Atom s) i

-- ---------------------------------------------------------------------
-- Ejercicio 11: Definir la función
--    esModeloCláusula :: Interpretación -> Cláusula -> Bool
-- tal que (esModeloCláusula i c) se verifica si i es modelo de c . Por
-- ejemplo,
--    esModeloCláusula [p,r] [p, q]     ==>  True
--    esModeloCláusula [r] [p, no q]    ==>  True
--    esModeloCláusula [q,r] [p, no q]  ==>  False
--    esModeloCláusula [q,r] []         ==>  False
-- ---------------------------------------------------------------------
```

```
esModeloCláusula :: Interpretación -> Cláusula -> Bool
esModeloCláusula i c =
    or [esModeloLiteral i l | l <- c]

-- ---------------------------------------------------------------------
-- Ejercicio 12: Definir la función
--    modelosCláusula :: Cláusula -> [Interpretación]
-- tal que (modelosCláusula c) es la lista de los modelos de c. Por
-- ejemplo,
--    modelosCláusula [no p, q]  ==>  [[p,q],[q],[]]
--    modelosCláusula [no p, p]  ==>  [[p],[]]
--    modelosCláusula []         ==>  []
-- ---------------------------------------------------------------------

modelosCláusula :: Cláusula -> [Interpretación]
modelosCláusula c =
    [i | i <- interpretacionesCláusula c,
         esModeloCláusula i c]

-- ---------------------------------------------------------------------
-- Modelos de conjuntos de cláusulas                                   --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 13: Definir la función
--    esModeloConjuntoCláusulas :: Interpretación -> [Cláusula] -> Bool
-- tal que (esModeloConjuntoCláusulas i c) se verifica si i es modelo de
-- c. Por ejemplo,
--    esModeloConjuntoCláusulas [p,r] [[p, no q], [r]]  ==>  True
--    esModeloConjuntoCláusulas [p] [[p, no q], [r]]    ==>  False
--    esModeloConjuntoCláusulas [p] []                  ==>  True
-- ---------------------------------------------------------------------

esModeloConjuntoCláusulas :: Interpretación -> [Cláusula] -> Bool
esModeloConjuntoCláusulas i s =
    and [esModeloCláusula i c | c <- s]

-- ---------------------------------------------------------------------
-- Ejercicio 14: Definir la función
--    modelosConjuntoCláusulas :: [Cláusula] -> [Interpretación]
```

```
-- tal que (modelosConjuntoCláusulas s) es la lista de los modelos de
-- s. Por ejemplo,
--    modelosConjuntoCláusulas [[no p, q], [no q, p]]
--    ==> [[p,q],[]]
--    modelosConjuntoCláusulas [[no p, q], [p], [no q]]
--    ==> []
--    modelosConjuntoCláusulas [[p, no p, q]]
--    ==> [[p,q],[p],[q],[]]
-- ----------------------------------------------------------------------

modelosConjuntoCláusulas :: [Cláusula] -> [Interpretación]
modelosConjuntoCláusulas s =
    [i | i <- interpretacionesConjuntoCláusula s,
         esModeloConjuntoCláusulas i s]

-- ----------------------------------------------------------------------
-- Cláusulas válidas, satisfacibles e insatisfacibles                 --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 15: Definir la función
--    esCláusulaVálida :: Cláusula -> Bool
-- tal que (esCláusulaVálida c) se verifica si la cláusula c es
-- válida. Por ejemplo,
--    esCláusulaVálida [p, q, no p]  ==>  True
--    esCláusulaVálida [p, q, no r]  ==>  False
--    esCláusulaVálida []            ==>  False
-- ----------------------------------------------------------------------

esCláusulaVálida :: Cláusula -> Bool
esCláusulaVálida c =
    [l | l <- c, elem (complementario l) c] /= []

-- Definición alternativa:
esCláusulaVálida1 :: Cláusula -> Bool
esCláusulaVálida1 c =
    and [esModeloCláusula i c | i <- interpretacionesCláusula c]

-- ----------------------------------------------------------------------
-- Ejercicio 16: Definir la función
```

```
--    esCláusulaInsatisfacible :: Cláusula -> Bool
-- tal que (esCláusulaInsatisfacible c) se verifica si la cláusula c es
-- insatisfacible. Por ejemplo,
--    esCláusulaInsatisfacible [p, q, no p]  ==>  False
--    esCláusulaInsatisfacible [p, q, no r]  ==>  False
--    esCláusulaInsatisfacible []            ==>  True
-- ----------------------------------------------------------------------

esCláusulaInsatisfacible :: Cláusula -> Bool
esCláusulaInsatisfacible c =
    null c

-- Definición alternativa:
esCláusulaInsatisfacible1 :: Cláusula -> Bool
esCláusulaInsatisfacible1 c =
    and [not (esModeloCláusula i c) | i <- interpretacionesCláusula c]

-- ----------------------------------------------------------------------
-- Ejercicio 17: Definir la función
--    esCláusulaSatisfacible :: Cláusula -> Bool
-- tal que (esCláusulaSatisfacible c) se verifica si la cláusula c es
-- satisfacible. Por ejemplo,
--    esCláusulaSatisfacible [p, q, no p]  ==>  True
--    esCláusulaSatisfacible [p, q, no r]  ==>  True
--    esCláusulaSatisfacible []  ==>  False
-- ----------------------------------------------------------------------

esCláusulaSatisfacible :: Cláusula -> Bool
esCláusulaSatisfacible c =
    not (null c)

-- Definición alternativa:
esCláusulaSatisfacible1 :: Cláusula -> Bool
esCláusulaSatisfacible1 c =
    or [esModeloCláusula i c | i <- interpretacionesCláusula c]

-- ----------------------------------------------------------------------
-- Conjuntos válidos, consistentes e inconsistentes de cláusulas      --
-- ----------------------------------------------------------------------
```

```
-- ----------------------------------------------------------------------
-- Ejercicio 18: Definir la función
--    esConjuntoVálidoDeCláusulas :: [Cláusula] -> Bool
-- tal que (esConjuntoVálidoDeCláusulas s) se verifica si el conjunto de
-- cláusulas s es válido. Por ejemplo,
--    esConjuntoVálidoDeCláusulas [[no p, q], [no q, p]]  ==>  False
--    esConjuntoVálidoDeCláusulas [[no p, p], [no q, q]]  ==>  True
--    esConjuntoVálidoDeCláusulas []                      ==>  True
-- ----------------------------------------------------------------------

esConjuntoVálidoDeCláusulas :: [Cláusula] -> Bool
esConjuntoVálidoDeCláusulas s =
    and [esCláusulaVálida c | c <- s]

-- Definición alternativa:
esConjuntoVálidoDeCláusulas1 :: [Cláusula] -> Bool
esConjuntoVálidoDeCláusulas1 s =
    modelosConjuntoCláusulas s == interpretacionesConjuntoCláusula s

-- ----------------------------------------------------------------------
-- Ejercicio 19: Definir la función
--    esConjuntoConsistenteDeCláusulas :: [Cláusula] -> Bool
-- tal que (esConjuntoConsistenteDeCláusulas s) se verifica si el
-- conjunto de cláusulas s es consistente. Por ejemplo,
--    esConjuntoConsistenteDeCláusulas [[no p, q], [no q, p]]  ==>  True
--    esConjuntoConsistenteDeCláusulas [[no p, p], [no q, q]]  ==>  True
--    esConjuntoConsistenteDeCláusulas []                      ==>  True
-- ----------------------------------------------------------------------

esConjuntoConsistenteDeCláusulas :: [Cláusula] -> Bool
esConjuntoConsistenteDeCláusulas s =
    not (null (modelosConjuntoCláusulas s))

-- ----------------------------------------------------------------------
-- Ejercicio 20: Definir la función
--    esConjuntoInconsistenteDeCláusulas :: [Cláusula] -> Bool
-- tal que (esConjuntoInconsistenteDeCláusulas s) se verifica si el
-- conjunto de cláusulas s es consistente. Por ejemplo,
--    esConjuntoInconsistenteDeCláusulas [[no p,q],[no q,p]]  ==>  False
--    esConjuntoInconsistenteDeCláusulas [[no p],[p]]         ==>  True
```

```
-- ----------------------------------------------------------------------

esConjuntoInconsistenteDeCláusulas :: [Cláusula] -> Bool
esConjuntoInconsistenteDeCláusulas s =
    null (modelosConjuntoCláusulas s)

-- ----------------------------------------------------------------------
-- Validez de fórmulas mediante cláusulas                               --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 21: Definir la función
--    esVálidaPorCláusulas :: Prop -> Bool
-- tal que (esVálidaPorCláusulas f) se verifica si el conjunto de
-- cláusulas de f es válido. Por ejemplo,
--    esVálidaPorCláusulas (p --> q)                ==>  False
--    esVálidaPorCláusulas (p --> p)                ==>  True
--    esVálidaPorCláusulas ((p --> q) \/ (q --> p))  ==>  True
-- ----------------------------------------------------------------------

esVálidaPorCláusulas :: Prop -> Bool
esVálidaPorCláusulas f =
    esConjuntoVálidoDeCláusulas (cláusulas f)

-- ----------------------------------------------------------------------
-- Consecuencia mediante cláusulas                                      --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 22: Definir la función
--    esConsecuenciaEntreCláusulas :: [Cláusula] -> [Cláusula] -> Bool
-- tal que (esConsecuenciaEntreCláusulas s1 s2) se verifica si todos los
-- modelos de s1 son modelos de s2. Por ejemplo,
--    esConsecuenciaEntreCláusulas [[no p,q],[no q,r]] [[no p,r]]
--    ==> True
--    esConsecuenciaEntreCláusulas [[p]] [[p],[q]]
--    ==> False
-- ----------------------------------------------------------------------

esConsecuenciaEntreCláusulas :: [Cláusula] -> [Cláusula] -> Bool
```

```
esConsecuenciaEntreCláusulas s1 s2 =
    null [i | i <- interpretacionesConjuntoCláusula (s1++s2)
            , esModeloConjuntoCláusulas i s1
            , not ((esModeloConjuntoCláusulas i s2))]

-- ----------------------------------------------------------------------
-- Ejercicio 23: Definir la función
--    esConsecuenciaPorCláusulas1 :: [Prop] -> Prop -> Bool
-- tal que (esConsecuenciaPorCláusulas s f) se verifica si las cláusulas
-- de f son consecuencias de las de s. Por ejemplo,
--    esConsecuenciaPorCláusulas [(p --> q), (q --> r)] (p --> r)
--    ==> True
--    esConsecuenciaPorCláusulas [p] (p /\ q)
--    ==> False
-- ----------------------------------------------------------------------

esConsecuenciaPorCláusulas :: [Prop] -> Prop -> Bool
esConsecuenciaPorCláusulas s f =
    esConsecuenciaEntreCláusulas (cláusulasConjunto s) (cláusulas f)

-- Definición alternativa:
esConsecuenciaPorCláusulas1 :: [Prop] -> Prop -> Bool
esConsecuenciaPorCláusulas1 s f =
    esConjuntoInconsistenteDeCláusulas (cláusulasConjunto ((Neg f):s))

-- ----------------------------------------------------------------------
-- Verificación                                                         --
-- ----------------------------------------------------------------------

ejemplosClausulas :: [Test]
ejemplosClausulas =
    ["cláusula ej 1" ~:
     cláusula p
     ==> [p],
     "cláusula ej 2" ~:
     cláusula (no p)
     ==> [no p],
     "cláusula ej 3" ~:
     cláusula (((no p) \/ r) \/ ((no p) \/ q))
     ==> [q,r,no p],
```

```
"cláusulas ej 1" ~:
cláusulas (p /\ (q --> r))
==> [[p],[r,no q]],
"cláusulas ej 2" ~:
cláusulas (no (p /\ (q --> r)))
==> [[q,no p],[no p,no r]],
"cláusulas ej 3" ~:
cláusulas (no(p <--> r))
==> [[p,r],[p,no p],[r,no r],[no p,no r]],
"cláusulasConjunto ej 1" ~:
cláusulasConjunto [p --> q, q --> r]
==> [[q,no p],[r,no q]],
"cláusulasConjunto ej 2" ~:
cláusulasConjunto [p --> q, q <--> p]
==> [[q,no p],[p,no q]],
"símbolosProposicionalesCláusula ej 1" ~:
símbolosProposicionalesCláusula [p, q, no p]
==> [p,q],
"símbolosProposicionalesConjuntoCláusula ej 1" ~:
símbolosProposicionalesConjuntoCláusula [[p, q],[no q, r]]
==> [p,q,r],
"interpretacionesCláusula ej 1" ~:
interpretacionesCláusula [p, q, no p]
==> [[p,q],[p],[q],[]],
"interpretacionesCláusula ej 2" ~:
interpretacionesCláusula []
==> [[]],
"interpretacionesConjuntoCláusula ej 1" ~:
interpretacionesConjuntoCláusula [[p,no q],[no p,q]]
==> [[p,q],[p],[q],[]],
"interpretacionesConjuntoCláusula ej 2" ~:
interpretacionesConjuntoCláusula []
==> [[]],
"esModeloLiteral ej 1" ~:
esModeloLiteral [p,r] p
==> True,
"esModeloLiteral ej 2" ~:
esModeloLiteral [p,r] q
==> False,
"esModeloLiteral ej 3" ~:
```

```
esModeloLiteral [p,r] (no p)
==> False,
"esModeloLiteral ej 4" ~:
esModeloLiteral [p,r] (no q)
==> True,
"esModeloCláusula ej 1" ~:
esModeloCláusula [p,r] [p, q]
==> True,
"esModeloCláusula ej 2" ~:
esModeloCláusula [r] [p, no q]
==> True,
"esModeloCláusula ej 3" ~:
esModeloCláusula [q,r] [p, no q]
==> False,
"esModeloCláusula ej 4" ~:
esModeloCláusula [q,r] []
==> False,
"modelosCláusula ej 1" ~:
modelosCláusula [no p, q]
==> [[p,q],[q],[]],
"modelosCláusula ej 2" ~:
modelosCláusula [no p, p]
==> [[p],[]],
"modelosCláusula ej 3" ~:
modelosCláusula []
==> [],
"esModeloConjuntoCláusulas ej 1" ~:
esModeloConjuntoCláusulas [p,r] [[p, no q], [r]]
==> True,
"esModeloConjuntoCláusulas ej 2" ~:
esModeloConjuntoCláusulas [p] [[p, no q], [r]]
==> False,
"esModeloConjuntoCláusulas ej 3" ~:
esModeloConjuntoCláusulas [p] []
==> True,
"modelosConjuntoCláusulas ej 1" ~:
modelosConjuntoCláusulas [[no p, q], [no q, p]]
==> [[p,q],[]],
"modelosConjuntoCláusulas ej 2" ~:
modelosConjuntoCláusulas [[no p, q], [p], [no q]]
```

```
==> [],
"modelosConjuntoCláusulas ej 3" ~:
modelosConjuntoCláusulas [[p, no p, q]]
==> [[p,q],[p],[q],[]],
"esCláusulaVálida ej 1" ~:
esCláusulaVálida [p, q, no p]
==> True,
"esCláusulaVálida ej 2" ~:
esCláusulaVálida [p, q, no r]
==> False,
"esCláusulaVálida ej 3" ~:
esCláusulaVálida []
==> False,
"esCláusulaInsatisfacible ej 1" ~:
esCláusulaInsatisfacible [p, q, no p]
==> False,
"esCláusulaInsatisfacible ej 2" ~:
esCláusulaInsatisfacible [p, q, no r]
==> False,
"esCláusulaInsatisfacible ej 3" ~:
esCláusulaInsatisfacible []
==> True,
"esCláusulaSatisfacible ej 1" ~:
esCláusulaSatisfacible [p, q, no p]
==> True,
"esCláusulaSatisfacible ej 2" ~:
esCláusulaSatisfacible [p, q, no r]
==> True,
"esCláusulaSatisfacible ej 3" ~:
esCláusulaSatisfacible []
==> False,
"esConjuntoVálidoDeCláusulas ej 1" ~:
esConjuntoVálidoDeCláusulas [[no p, q], [no q, p]]
==> False,
"esConjuntoVálidoDeCláusulas ej 2" ~:
esConjuntoVálidoDeCláusulas [[no p, p], [no q, q]]
==> True,
"esConjuntoVálidoDeCláusulas ej 3" ~:
esConjuntoVálidoDeCláusulas []
==> True,
```

```
    "esConjuntoConsistenteDeCláusulas ej 1" ~:
    esConjuntoConsistenteDeCláusulas [[no p, q], [no q, p]]
    ==> True,
    "esConjuntoConsistenteDeCláusulas ej 2" ~:
    esConjuntoConsistenteDeCláusulas [[no p, p], [no q, q]]
    ==> True,
    "esConjuntoConsistenteDeCláusulas ej 3" ~:
    esConjuntoConsistenteDeCláusulas []
    ==> True,
    "esConjuntoInconsistenteDeCláusulas ej 1" ~:
    esConjuntoInconsistenteDeCláusulas [[no p,q],[no q,p]]
    ==> False,
    "esConjuntoInconsistenteDeCláusulas ej 2" ~:
    esConjuntoInconsistenteDeCláusulas [[no p],[p]]
    ==> True,
    "esVálidaPorCláusulas ej 1" ~:
    esVálidaPorCláusulas (p --> q)
    ==> False,
    "esVálidaPorCláusulas ej 2" ~:
    esVálidaPorCláusulas (p --> p)
    ==> True,
    "esVálidaPorCláusulas ej 3" ~:
    esVálidaPorCláusulas ((p --> q) \/ (q --> p))
    ==> True,
    "esConsecuenciaEntreCláusulas ej 1" ~:
    esConsecuenciaEntreCláusulas [[no p,q],[no q,r]] [[no p,r]]
    ==> True,
    "esConsecuenciaEntreCláusulas ej 2" ~:
    esConsecuenciaEntreCláusulas [[p]] [[p],[q]]
    ==> False,
    "esConsecuenciaPorCláusulas ej 1" ~:
    esConsecuenciaPorCláusulas [(p --> q), (q --> r)] (p --> r)
    ==> True,
    "esConsecuenciaPorCláusulas ej 2" ~:
    esConsecuenciaPorCláusulas [p] (p /\ q)
    ==> False
    ]

verificaClausulas :: IO Counts
verificaClausulas =
```

```
    runTestTT (test ejemplosClausulas)

-- Clausulas> verificaClausulas
-- Cases: 55  Tried: 55  Errors: 0  Failures: 0
```

Capítulo 4

Tableros semánticos proposicionales

```
module TablerosSemanticos where

-- ----------------------------------------------------------------------
-- Librerías auxiliares                                                 --
-- ----------------------------------------------------------------------

import SintaxisSemantica
import FormasNormales
import Data.List
import Debug.Trace
import Test.HUnit
import Verificacion

-- ----------------------------------------------------------------------
-- Notación uniforme                                                    --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 1: Definir la función
--    dobleNegación :: Prop -> Bool
-- tal que (dobleNegación f) se verifica si f es una doble negación. Por
-- ejemplo,
--    dobleNegación (no (no p))     ==>  True
--    dobleNegación (no (p --> q))  ==>  False
-- ----------------------------------------------------------------------

dobleNegación :: Prop -> Bool
```

```
doble_negación (Neg (Neg _)) = True
doble_negación _             = False

-- ----------------------------------------------------------------------
-- Ejercicio 2: Definir la función
--    alfa :: Prop -> Bool
-- tal que (alfa f) se verifica si f es una fórmula alfa.
-- ----------------------------------------------------------------------

alfa :: Prop -> Bool
alfa (Conj _ _)       = True
alfa (Neg (Impl _ _)) = True
alfa (Neg (Disj _ _)) = True
alfa _                = False

-- ----------------------------------------------------------------------
-- Ejercicio 3: Definir la función
--    beta :: Prop -> Bool
-- tal que (beta d) se verifica si f es una fórmula beta.
-- ----------------------------------------------------------------------

beta :: Prop -> Bool
beta (Disj _ _)       = True
beta (Impl _ _)       = True
beta (Neg (Conj _ _)) = True
beta (Equi _ _)       = True
beta (Neg (Equi _ _)) = True
beta _                = False

-- ----------------------------------------------------------------------
-- Ejercicio 4: Definir la función
--    componentes :: Prop -> [Prop]
-- tal que (componentes ) es la lista de las componentes de la fórmula
-- f. Por ejemplo,
--    componentes (p /\ q --> r)       ==>  [no (p /\ q),r]
--    componentes (no (p /\ q --> r))  ==>  [(p /\ q),no r]
-- ----------------------------------------------------------------------

componentes :: Prop -> [Prop]
componentes (Neg (Neg f))    = [f]
```

```
componentes (Conj f g)       = [f, g]
componentes (Neg (Impl f g)) = [f, Neg g]
componentes (Neg (Disj f g)) = [Neg f, Neg g]
componentes (Disj f g)       = [f, g]
componentes (Impl f g)       = [Neg f, g]
componentes (Neg (Conj f g)) = [Neg f, Neg g]
componentes (Equi f g)       = [Conj f g, Conj (Neg f) (Neg g)]
componentes (Neg (Equi f g)) = [Conj f (Neg g), Conj (Neg f) g]

-- ---------------------------------------------------------------------
-- Modelos mediante tableros                                           --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 5: Definir la función
--    conjuntoDeLiterales :: [Prop] -> Bool
-- tal que (conjuntoDeLiterales fs) se verifica si fs es un conjunto de
-- literales. Por ejemplo,
--    conjuntoDeLiterales [p --> q, no r, r /\ s, p]  ==>  False
--    conjuntoDeLiterales [p, no q, r]                ==>  True
-- ---------------------------------------------------------------------

conjuntoDeLiterales :: [Prop] -> Bool
conjuntoDeLiterales fs =
    and [literal f | f <- fs]

-- ---------------------------------------------------------------------
-- Ejercicio 6: Definir la función
--    tieneContradicción :: [Prop] -> Bool
-- tal que (tieneContradicción fs) se verifica si fs contiene una
-- fórmula y su negación. Por ejemplo,
--    tieneContradicción [r, p /\ q, s, no(p /\ q)]  ==>  True
-- ---------------------------------------------------------------------

tieneContradicción :: [Prop] -> Bool
-- tieneContradicción fs
--     | trace ("  " ++ show fs) False = undefined
tieneContradicción fs =
    [f | f <- fs, elem (Neg f) fs] /= []
```

```
-- ----------------------------------------------------------------------
-- Ejercicio 7: Definir la función
--    expansiónDN :: [Prop] -> Prop -> [[Prop]]
-- tal que (expansiónDN fs f) es la expansión de fs mediante la doble
-- negación f. Por ejemplo,
--    expansiónDN [p, no(no q), r] (no(no q))  ==>  [[q,p,r]]
-- ----------------------------------------------------------------------

expansiónDN :: [Prop] -> Prop -> [[Prop]]
expansiónDN fs f =
    [(componentes f) `union` (delete f fs)]

-- ----------------------------------------------------------------------
-- Ejercicio 8: Definir la función
--    expansiónAlfa :: [Prop] -> Prop -> [[Prop]]
-- tal que (expansiónAlfa fs f) es la expansión de fs mediante la
-- fórmula alfa f. Por ejemplo,
--    expansiónAlfa [q, (p1 /\ p2) , r] (p1 /\ p2)  ==>  [[p1,p2,q,r]]
-- ----------------------------------------------------------------------

expansiónAlfa :: [Prop] -> Prop -> [[Prop]]
expansiónAlfa fs f =
    [(componentes f) `union` (delete f fs)]

-- ----------------------------------------------------------------------
-- Ejercicio 9: Definir la función
--    expansiónBeta :: [Prop] -> Prop -> [[Prop]]
-- tal que (expansiónBeta fs f) es la expansión de fs mediante la
-- fórmula beta f. Por ejemplo,
--    expansiónBeta [q, (p1 \/ p2) , r] (p1 \/ p2)  ==>  [[p1,q,r],[p2,q,r]]
-- ----------------------------------------------------------------------

expansiónBeta :: [Prop] -> Prop -> [[Prop]]
expansiónBeta fs f =
    [[g] `union` (delete f fs) | g <- componentes f]

-- ----------------------------------------------------------------------
-- Ejercicio 10: Definir la función
--    sucesores :: [Prop] -> [[Prop]]
-- tal que (sucesores fs) es la lista de sucesores de fs. Por ejemplo,
```

```
--    sucesores [q \/ s, no(no r), p1 /\ p2] => [[r,(q \/ s),(p1 /\ p2)]]
--    sucesores [r,(q \/ s),(p1 /\ p2)]      => [[p1,p2,r,(q \/ s)]]
--    sucesores [p1,p2,r,(q \/ s)]           => [[q,p1,p2,r],[s,p1,p2,r]]
-- ----------------------------------------------------------------------

sucesores :: [Prop] -> [[Prop]]
sucesores fs
    | doblesNegación /= []  = expansiónDN   fs (head doblesNegación)
    | alfas /= []           = expansiónAlfa fs (head alfas)
    | betas /= []           = expansiónBeta fs (head betas)
    where doblesNegación = [f | f <- fs, dobleNegación f]
          alfas          = [f | f <- fs, alfa f]
          betas          = [f | f <- fs, beta f]

-- ----------------------------------------------------------------------
-- Ejercicio 11: Definir la función
--    modelosTab :: [Prop] -> [[Prop]]
-- tal que (modelosTab fs) es el conjunto de los modelos de fs
-- calculados mediante el método de tableros semánticos. Por ejemplo,
--    modelosTab [p --> q, no(q --> p)]
--    ==> [[no p,q],[q,no p]]
--    modelosTab [p --> q, no q --> no p]
--    ==> [[q,no p],[no p],[q],[no p,q]]
-- ----------------------------------------------------------------------

modelosTab :: [Prop] -> [[Prop]]
modelosTab fs
    | tieneContradicción fs  = []
    | conjuntoDeLiterales fs = [fs]
    | otherwise              = unión General [modelosTab gs
                                             | gs <- sucesores fs]

-- ----------------------------------------------------------------------
-- Ejercicio 12: Definir la función
--    subconjunto :: Eq a => [a] -> [a] -> Bool
-- tal que (subconjunto x y) se verifica si x es subconjunto de y. Por
-- ejemplo,
--    subconjunto [1,3] [3,2,1]    ==>  True
--    subconjunto [1,3,5] [3,2,1]  ==> False
-- ----------------------------------------------------------------------
```

```
subconjunto :: Eq a => [a] -> [a] -> Bool
subconjunto xs ys =
    and [elem x ys | x <- xs]

-- ---------------------------------------------------------------------
-- Ejercicio 13: Definir la función
--    modelosGenerales :: [Prop] -> [[Prop]]
-- tal que (modelosGenerales fs) es el conjunto de los modelos generales
-- de fs calculados mediante el método de tableros semánticos. Por
-- ejemplo,
--    modelosGenerales [p --> q, no q --> no p]  ==>  [[no p],[q]]
-- ---------------------------------------------------------------------

modelosGenerales :: [Prop] -> [[Prop]]
modelosGenerales fs =
    [gs | gs <- modelos
        , [hs | hs <- delete gs modelos, subconjunto hs gs] == []]
    where modelos = modelosTab fs

-- ---------------------------------------------------------------------
-- Teoremas por tableros                                               --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 14: Definir la función
--    esTeoremaPorTableros :: Prop -> Bool
-- tal que (esTeoremaPorTableros f) se verifica si la fórmula f es
-- teorema (mediante tableros semánticos). Por ejemplo,
--    esTeoremaPorTableros (p --> p)  ==>  True
--    esTeoremaPorTableros (p --> q)  ==>  False
-- ---------------------------------------------------------------------

esTeoremaPorTableros :: Prop -> Bool
esTeoremaPorTableros f =
    modelosTab [Neg f] == []

-- ---------------------------------------------------------------------
-- Consecuencia por tableros                                           --
-- ---------------------------------------------------------------------
```

```
-- ----------------------------------------------------------------------
-- Ejercicio 15: Definir la función
--    esDeduciblePorTableros :: [Prop] -> Prop -> Bool
-- tal que (esDeduciblePorTableros fs f) se verifica si la fórmula f es
-- consecuencia (mediante tableros) del conjunto de fórmulas fs. Por
-- ejemplo,
--    esDeduciblePorTableros [p --> q, q --> r] (p --> r)   ==>  True
--    esDeduciblePorTableros [p --> q, q --> r] (p <--> r)  ==>  False
-- ----------------------------------------------------------------------

esDeduciblePorTableros :: [Prop] -> Prop -> Bool
esDeduciblePorTableros fs f =
    modelosTab ((Neg f):fs) == []

-- ----------------------------------------------------------------------
-- Verificación                                                         --
-- ----------------------------------------------------------------------

ejemplosTablerosSemanticos :: [Test]
ejemplosTablerosSemanticos =
    ["dobleNegacion ej 1" ~:
     dobleNegación (no (no p))
     ==> True,
     "dobleNegacion ej 2" ~:
     dobleNegación (no (p --> q))
     ==> False,
     "componentes ej 1" ~:
     componentes (p /\ q --> r)
     ==> [no (p /\ q),r],
     "componentes ej 2" ~:
     componentes (no (p /\ q --> r))
     ==> [(p /\ q),no r],
     "conjuntoDeLiterales ej 1" ~:
     conjuntoDeLiterales [p --> q, no r, r /\ s, p]
     ==> False,
     "conjuntoDeLiterales ej 2" ~:
     conjuntoDeLiterales [p, no q, r]
     ==> True,
     "tieneContradiccion ej 1" ~:
```

```
tieneContradicción [r, p /\ q, s, no(p /\ q)]
==> True,
"expansionDN ej 1" ~:
expansiónDN [p, no(no q), r] (no(no q))
==> [[q,p,r]],
"expansionAlfa ej 1" ~:
expansiónAlfa [q, (p1 /\ p2) , r] (p1 /\ p2)
==> [[p1,p2,q,r]],
"expansionBeta ej 1" ~:
expansiónBeta [q, (p1 \/ p2) , r] (p1 \/ p2)
==> [[p1,q,r],[p2,q,r]],
"sucesores ej 1" ~:
sucesores [q \/ s, no(no r), p1 /\ p2]
==> [[r,(q \/ s),(p1 /\ p2)]],
"sucesores ej 2" ~:
sucesores [r,(q \/ s),(p1 /\ p2)]
==> [[p1,p2,r,(q \/ s)]],
"sucesores ej 3" ~:
sucesores [p1,p2,r,(q \/ s)]
==> [[q,p1,p2,r],[s,p1,p2,r]],
"modelosTab ej 1" ~:
modelosTab [p --> q, no(q --> p)]
==> [[no p,q],[q,no p]],
"modelosTab ej 2" ~:
modelosTab [p --> q, no q --> no p]
==> [[q,no p],[no p],[q],[no p,q]],
"subconjunto ej 1" ~:
subconjunto [1,3] [3,2,1]
==> True,
"subconjunto ej 2" ~:
subconjunto [1,3,5] [3,2,1]
==> False,
"modelosGenerales ej 1" ~:
modelosGenerales [p --> q, no q --> no p]
==> [[no p],[q]],
"esTeoremaPorTableros ej 1" ~:
esTeoremaPorTableros (p --> p)
==> True,
"esTeoremaPorTableros ej 2" ~:
esTeoremaPorTableros (p --> q)
```

```
     ==> False,
     "esDeduciblePorTableros ej 1" ~:
     esDeduciblePorTableros [p --> q, q --> r] (p --> r)
     ==> True,
     "esDeduciblePorTableros ej 2" ~:
     esDeduciblePorTableros [p --> q, q --> r] (p <--> r)
     ==> False
    ]

verificaTablerosSemanticos :: IO Counts
verificaTablerosSemanticos =
    runTestTT (test ejemplosTablerosSemanticos)

-- TablerosSemanticos> verificaTablerosSemanticos
-- Cases: 22  Tried: 22  Errors: 0  Failures: 0
```

Capítulo 5

Cálculo de secuentes proposicionales

```
module Secuentes where

import SintaxisSemantica
import Data.List
import Debug.Trace
import Test.HUnit
import Verificacion

-- ----------------------------------------------------------------------
-- Secuentes                                                            --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 1: Definir el tipo de datos Secuentes para representar los
-- secuentes como pares de listas de fórmulas.
-- ----------------------------------------------------------------------

type Secuente = ([Prop],[Prop])

-- ----------------------------------------------------------------------
-- Demostrabilidad por secuentes                                        --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 2: Definir la función
--    esAxioma :: Secuente -> Bool
-- tal que (esAxioma s) se verifica si el secuente s es un axioma. Por
```

```
-- ejemplo,
--    esAxioma ([p /\ q, r], [r --> q, p /\ q])  ==>  True
--    esAxioma ([p /\ q, r], [r --> q, p, q])    ==>  False
-- ----------------------------------------------------------------------

esAxioma :: Secuente -> Bool
esAxioma (i,d) =
    intersect i d /= []

-- ----------------------------------------------------------------------
-- Ejercicio 3: Definir la función
--    reglaIzquierda :: Prop -> Secuente -> [Secuente]
-- tal que (reglaIzquierda f s) es el secuente obtenido aplicando a s la
-- regla izquierda correspondiente a la fórmula f. Por ejemplo,
--    reglaIzquierda (no p) ([no p, q],[r])
--    ==> [([q],[p,r])]
--    reglaIzquierda (p /\ q) ([r, p /\ q],[s])
--    ==> [([p,q,r],[s])]
--    reglaIzquierda (p \/ q) ([r, p \/ q],[s])
--    ==> [([p,r],[s]),([q,r],[s])]
--    reglaIzquierda (p --> q) ([r, p --> q],[s])
--    ==> [([r],[p,s]),([q,r],[s])]
--    reglaIzquierda (p <--> q) ([r, p <--> q],[s])
--    ==> [([p,q,r],[s]),([r],[p,q,s])]
-- ----------------------------------------------------------------------

reglaIzquierda :: Prop -> Secuente -> [Secuente]
reglaIzquierda f@(Neg f1)     (i,d) = [(delete f i,         f1:d)]
reglaIzquierda f@(Conj f1 f2) (i,d) = [(f1:f2:(delete f i), d)]
reglaIzquierda f@(Disj f1 f2) (i,d) = [(f1:(delete f i),    d),
                                       (f2:(delete f i),    d)]
reglaIzquierda f@(Impl f1 f2) (i,d) = [(delete f i,         f1:d),
                                       (f2:(delete f i),    d)]
reglaIzquierda f@(Equi f1 f2) (i,d) = [(f1:f2:(delete f i), d),
                                       (delete f i,         f1:f2:d)]

-- ----------------------------------------------------------------------
-- Ejercicio 4: Definir la función
--    reglaDerecha :: Prop -> Secuente -> [Secuente]
-- tal que (reglaDerecha f s) es el secuente obtenido aplicando a s la
```

```
-- regla derecha correspondiente a la fórmula f. Por ejemplo,
--    reglaDerecha (no p) ([q],[no p, r])
--    ==> [([p,q],[r])]
--    reglaDerecha (p /\ q) ([s],[p /\ q, r])
--    ==> [([s],[p,r]),([s],[q,r])]
--    reglaDerecha (p \/ q) ([s],[p \/ q, r])
--    ==> [([s],[p,q,r])]
--    reglaDerecha (p --> q) ([s],[p --> q, r])
--    ==> [([p,s],[q,r])]
--    reglaDerecha (p <--> q) ([s],[p <--> q, r])
--    ==> [([p,s],[q,r]),([q,s],[p,r])]
-- ---------------------------------------------------------------------

reglaDerecha :: Prop -> Secuente -> [Secuente]
reglaDerecha f@(Neg f1)     (i,d) = [(f1:i, delete f d)]
reglaDerecha f@(Conj f1 f2) (i,d) = [(i,    f1:(delete f d)),
                                     (i,    f2:(delete f d))]
reglaDerecha f@(Disj f1 f2) (i,d) = [(i,    f1:f2:(delete f d))]
reglaDerecha f@(Impl f1 f2) (i,d) = [(f1:i, f2:(delete f d))]
reglaDerecha f@(Equi f1 f2) (i,d) = [(f1:i, f2:(delete f d)),
                                     (f2:i, f1:(delete f d))]

-- ---------------------------------------------------------------------
-- Ejercicio 5: Definir la función
--    esAtómica :: Prop -> Bool
-- tal que (esAtómica f) se verifica si la fórmula f es atómica. Por
-- ejemplo,
--    esAtómica p         ==>  True
--    esAtómica (p /\ q)  ==>  False
-- ---------------------------------------------------------------------

esAtómica :: Prop -> Bool
esAtómica (Atom _) = True
esAtómica _        = False

-- ---------------------------------------------------------------------
-- Ejercicio 6: Definir la función
--    compuestasIzquierda :: Secuente -> [Prop]
-- tal que (compuestasIzquierda s) es la lista de las fórmulas
-- compuestas en la parte izquierda del secuente s. Por ejemplo,
```

```
--    compuestasIzquierda ([no p, q, r /\ s],[no q])
--    ==> [no p,(r /\ s)]
-- ---------------------------------------------------------------------

compuestasIzquierda :: Secuente -> [Prop]
compuestasIzquierda (i,_) =
    [f | f <- i, not (esAtómica f)]

-- ---------------------------------------------------------------------
-- Ejercicio 7: Definir la función
--    compuestasDerecha :: Secuente -> [Prop]
-- tal que (compuestasDerecha s) es la lista de las fórmulas
-- compuestas en la parte derecha del secuente s. Por ejemplo,
--    compuestasDerecha ([no p, q, r /\ s],[no q, s --> p, r])
--    ==> [no q,(s --> p)]
-- ---------------------------------------------------------------------

compuestasDerecha :: Secuente -> [Prop]
compuestasDerecha (_,d) =
    [f | f <- d, not (esAtómica f)]

-- ---------------------------------------------------------------------
-- Ejercicio 8: Definir la función
--    esProbablePorSecuentes' :: Secuente -> Bool
-- tal que (esProbablePorSecuentes' s) se verifica si s es probable. Por
-- ejemplo,
--    esProbablePorSecuentes' ([p --> q, q --> r],[p --> r])   ==>  True
--    esProbablePorSecuentes' ([p --> q, q --> r],[p <--> r])  ==>  False
--    esProbablePorSecuentes' ([],[p --> p])                   ==>  True
--    esProbablePorSecuentes' ([p /\ no(p)],[])                ==>  True
-- ---------------------------------------------------------------------

esProbablePorSecuentes' :: Secuente -> Bool
esProbablePorSecuentes' s
    | esAxioma s = True
    | compuestasI /= [] =
        sonProbalesPorSecuentes (reglaIzquierda (head compuestasI) s)
    | compuestasD /= [] =
        sonProbalesPorSecuentes (reglaDerecha (head compuestasD) s)
    | otherwise = False
```

```
    where compuestasI = compuestasIzquierda s
          compuestasD = compuestasDerecha s

-- ---------------------------------------------------------------------
-- Ejercicio 9: Definir la función
--    sonProbalesPorSecuentes :: [Secuente] -> Bool
-- tal que (sonProbalesPorSecuentes ss) se verifica si los secuentes de
-- ss son probables. Por ejemplo,
--    sonProbalesPorSecuentes [([],[q,p,(p --> r)]),([r],[p,(p --> r)])]
--    ==> True
-- ---------------------------------------------------------------------

sonProbalesPorSecuentes :: [Secuente] -> Bool
sonProbalesPorSecuentes ss =
    and [esProbablePorSecuentes' s | s <- ss]

-- ---------------------------------------------------------------------
-- Ejercicio 10: Definir la función
--    esProbablePorSecuentes :: Prop -> Bool
-- tal que (esProbablePorSecuentes f) se verifica si la fórmula f es
-- probable mediante secuentes. Por ejemplo,
--    esProbablePorSecuentes ((p --> q) \/ (q --> p))  ==>  True
--    esProbablePorSecuentes (p --> q)                 ==>  False
-- ---------------------------------------------------------------------

esProbablePorSecuentes :: Prop -> Bool
esProbablePorSecuentes f
    = esProbablePorSecuentes' ([],[f])

-- ---------------------------------------------------------------------
-- Prueba por secuentes                                              --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 11: Definir la función
--    marcas :: Int -> String
-- tal que (marcas n) es la cadena correspondiente a n marcas. Por
-- ejemplo,
--    > marcas 3
--    " ||| "
```

```
-- ----------------------------------------------------------------------

marcas :: Int -> String
marcas n =
    " "++['|' | i <- [1..n]]++" "

-- ----------------------------------------------------------------------
-- Ejercicio 12: Definir la función
--    pruebaPorSecuentes' :: Secuente -> Int -> Bool
-- tal que (pruebaPorSecuentes' s) escribe la prueba por secuentes de
-- s. Por ejemplo,
--    Secuentes> pruebaPorSecuentes' ([p --> q, q --> r],[p --> r]) 1
--     | [(p --> q),(q --> r)] ==> [(p --> r)]
--     || [(q --> r)] ==> [p,(p --> r)]
--     ||| [] ==> [q,p,(p --> r)]
--     ||| [p] ==> [r,q,p]
--     ||| [r] ==> [p,(p --> r)]
--     ||| [p,r] ==> [r,p]
--     || [q,(q --> r)] ==> [(p --> r)]
--     ||| [q] ==> [q,(p --> r)]
--     ||| [r,q] ==> [(p --> r)]
--     ||| [p,r,q] ==> [r]
--    True
-- ----------------------------------------------------------------------

pruebaPorSecuentes' :: Secuente -> Int -> Bool
pruebaPorSecuentes' (i,d) n
    | trace ((marcas n) ++ show i ++ " ==> " ++ show d) False = undefined
pruebaPorSecuentes' s n
    | esAxioma s = True
    | compuestasI /= [] =
        pruebasPorSecuentes' (reglaIzquierda (head compuestasI) s) n
    | compuestasD /= [] =
        pruebasPorSecuentes' (reglaDerecha (head compuestasD) s) n
    | otherwise = False
    where compuestasI = compuestasIzquierda s
          compuestasD = compuestasDerecha s

-- ----------------------------------------------------------------------
-- Ejercicio 13: Definir la función
```

```
--    pruebasPorSecuentes' :: [Secuente] -> Int -> Bool
-- tal que (pruebasPorSecuentes' ss) escribe las pruebas por secuentes
-- de ss. Por ejemplo,
--    > pruebasPorSecuentes' [([],[q,p,(p --> r)]),([r],[p,(p --> r)])] 1
--     || [] ==> [q,p,(p --> r)]
--     || [p] ==> [r,q,p]
--     || [r] ==> [p,(p --> r)]
--     || [p,r] ==> [r,p]
--    True
-- ----------------------------------------------------------------------

pruebasPorSecuentes' :: [Secuente] -> Int -> Bool
pruebasPorSecuentes' [s] n =
    pruebaPorSecuentes' s n
pruebasPorSecuentes' ss n =
    and [pruebaPorSecuentes' s (n+1) | s <- ss]

-- ----------------------------------------------------------------------
-- Ejercicio 14: Definir la función
--    pruebaPorSecuentes :: Prop -> Bool
-- tal que (pruebaPorSecuentes f) escribe las pruebas por secuentes de
-- la fórmula f. Por ejemplo,
-- > pruebaPorSecuentes ((p --> q) /\ (q --> r) --> (p --> r))
--  | [] ==> [(((p --> q) /\ (q --> r)) --> (p --> r))]
--  | [((p --> q) /\ (q --> r))] ==> [(p --> r)]
--  | [(p --> q),(q --> r)] ==> [(p --> r)]
--  || [(q --> r)] ==> [p,(p --> r)]
--  ||| [] ==> [q,p,(p --> r)]
--  ||| [p] ==> [r,q,p]
--  ||| [r] ==> [p,(p --> r)]
--  ||| [p,r] ==> [r,p]
--  || [q,(q --> r)] ==> [(p --> r)]
--  ||| [q] ==> [q,(p --> r)]
--  ||| [r,q] ==> [(p --> r)]
--  ||| [p,r,q] ==> [r]
-- True
-- > pruebaPorSecuentes ((p --> q) /\ (q --> r) --> (p --> s))
--  | [] ==> [(((p --> q) /\ (q --> r)) --> (p --> s))]
--  | [((p --> q) /\ (q --> r))] ==> [(p --> s)]
--  | [(p --> q),(q --> r)] ==> [(p --> s)]
```

```
--  || [(q --> r)] ==> [p,(p --> s)]
--  ||| [] ==> [q,p,(p --> s)]
--  ||| [p] ==> [s,q,p]
--  ||| [r] ==> [p,(p --> s)]
--  ||| [p,r] ==> [s,p]
--  || [q,(q --> r)] ==> [(p --> s)]
--  ||| [q] ==> [q,(p --> s)]
--  ||| [r,q] ==> [(p --> s)]
--  ||| [p,r,q] ==> [s]
-- False
-- ------------------------------------------------------------------

pruebaPorSecuentes :: Prop -> Bool
pruebaPorSecuentes f =
    pruebaPorSecuentes' ([],[f]) 1

-- ------------------------------------------------------------------
-- Deducción por secuentes                                          --
-- ------------------------------------------------------------------

-- ------------------------------------------------------------------
-- Ejercicio 15: Definir la función
--    esDeduciblePorSecuentes :: [Prop] -> Prop -> Bool
-- tal que (esDeduciblePorSecuentes s f) se verifica si la fórmula f es
-- deducible por secuentes a partir de s. Por ejemplo,
--    esDeduciblePorSecuentes [p --> q, q --> r] (p --> r)
--    ==> True
--    esDeduciblePorSecuentes [p --> q, q --> r] (p <--> r)
--    ==> False
-- ------------------------------------------------------------------

esDeduciblePorSecuentes :: [Prop] -> Prop -> Bool
esDeduciblePorSecuentes fs g =
    esProbablePorSecuentes' (fs,[g])

-- ------------------------------------------------------------------
-- Ejercicio 16: Definir la función
--    deducciónPorSecuentes  :: [Prop] -> Prop -> Bool
-- tal que (deducciónPorSecuentes s f) escribe la deducción por
-- secuentes de la fórmula f a partir del conjunto de fórmulas s. Por
```

```
-- ejemplo,
--    > deducciónPorSecuentes [p --> q, q --> r] (p --> r)
--     | [(p --> q),(q --> r)] ==> [(p --> r)]
--     || [(q --> r)] ==> [p,(p --> r)]
--     ||| [] ==> [q,p,(p --> r)]
--     ||| [p] ==> [r,q,p]
--     ||| [r] ==> [p,(p --> r)]
--     ||| [p,r] ==> [r,p]
--     || [q,(q --> r)] ==> [(p --> r)]
--     ||| [q] ==> [q,(p --> r)]
--     ||| [r,q] ==> [(p --> r)]
--     ||| [p,r,q] ==> [r]
--    True
-- ----------------------------------------------------------------------

deducciónPorSecuentes  :: [Prop] -> Prop -> Bool
deducciónPorSecuentes fs g =
    pruebaPorSecuentes'  (fs,[g]) 1

-- ----------------------------------------------------------------------
-- Verificación                                                         --
-- ----------------------------------------------------------------------

ejemplosSecuentes :: [Test]
ejemplosSecuentes =
    ["esAxioma ej 1" ~:
     esAxioma ([p /\ q, r], [r --> q, p /\ q])
     ==> True,
     "esAxioma ej 2" ~:
     esAxioma ([p /\ q, r], [r --> q, p, q])
     ==> False,
     "reglaIzquierda ej 1" ~:
     reglaIzquierda (no p) ([no p, q],[r])
     ==> [([q],[p,r])],
     "reglaIzquierda ej 2" ~:
     reglaIzquierda (p /\ q) ([r, p /\ q],[s])
     ==> [([p,q,r],[s])],
     "reglaIzquierda ej 3" ~:
     reglaIzquierda (p \/ q) ([r, p \/ q],[s])
     ==> [([p,r],[s]),([q,r],[s])],
```

```
"reglaIzquierda ej 4" ~:
reglaIzquierda (p --> q) ([r, p --> q],[s])
==> [([r],[p,s]),([q,r],[s])],
"reglaIzquierda ej 5" ~:
reglaIzquierda (p <--> q) ([r, p <--> q],[s])
==> [([p,q,r],[s]),([r],[p,q,s])],
"reglaDerecha ej 1" ~:
reglaDerecha (no p) ([q],[no p, r])
==> [([p,q],[r])],
"reglaDerecha ej 2" ~:
reglaDerecha (p /\ q) ([s],[p /\ q, r])
==> [([s],[p,r]),([s],[q,r])],
"reglaDerecha ej 3" ~:
reglaDerecha (p \/ q) ([s],[p \/ q, r])
==> [([s],[p,q,r])],
"reglaDerecha ej 4" ~:
reglaDerecha (p --> q) ([s],[p --> q, r])
==> [([p,s],[q,r])],
"reglaDerecha ej 5" ~:
reglaDerecha (p <--> q) ([s],[p <--> q, r])
==> [([p,s],[q,r]),([q,s],[p,r])],
"esAtomica ej 1" ~:
esAtómica p
==> True,
"esAtomica ej 2" ~:
esAtómica (p /\ q)
==> False,
"compuestasIzquierda ej 1" ~:
compuestasIzquierda ([no p, q, r /\ s],[no q])
==> [no p,(r /\ s)],
"compuestasDerecha ej 1" ~:
compuestasDerecha ([no p, q, r /\ s],[no q, s --> p, r])
==> [no q,(s --> p)],
"esProbablePorSecuentes' ej 1" ~:
esProbablePorSecuentes' ([p --> q, q --> r],[p --> r])
==> True,
"esProbablePorSecuentes' ej 2" ~:
esProbablePorSecuentes' ([p --> q, q --> r],[p <--> r])
==> False,
"esProbablePorSecuentes' ej 3" ~:
```

```
    esProbablePorSecuentes' ([],[p --> p])
    ==> True,
    "esProbablePorSecuentes' ej 4" ~:
    esProbablePorSecuentes' ([p /\ no(p)],[])
    ==> True,
    "sonProbalesPorSecuentes ej 1" ~:
    sonProbalesPorSecuentes [([],[q,p,(p --> r)]),([r],[p,(p --> r)])]
    ==> True,
    "esProbablePorSecuentes ej 2" ~:
    esProbablePorSecuentes ((p --> q) \/ (q --> p))
    ==> True,
    "esProbablePorSecuentes ej 3" ~:
    esProbablePorSecuentes (p --> q)
    ==> False,
    "marcas  ej 1" ~:
    marcas 3
    ==>   " ||| ",
    "esDeduciblePorSecuentes ej 1" ~:
    esDeduciblePorSecuentes [p --> q, q --> r] (p --> r)
    ==> True,
    "esDeduciblePorSecuentes ej 2" ~:
    esDeduciblePorSecuentes [p --> q, q --> r] (p <--> r)
    ==> False
  ]

verificaSecuentes :: IO Counts
verificaSecuentes =
    runTestTT (test ejemplosSecuentes)

-- Secuentes> verificaSecuentes
-- Cases: 26  Tried: 26  Errors: 0  Failures: 0
```

Capítulo 6

El procedimiento de Davis y Putnam

```
module DavisPutnam where

import SintaxisSemantica
import FormasNormales
import Clausulas
import Data.List
import Test.HUnit
import Verificacion

-- ------------------------------------------------------------------
-- Eliminación de tautologías                                       --
-- ------------------------------------------------------------------

-- ------------------------------------------------------------------
-- Ejercicio 1: Definir la función
--    esTautología :: Cláusula -> Bool
-- tal que (esTautología c) se verifica si c es una tautología. Por
-- ejemplo,
--    esTautología [p, q, no p]  ==>  True
--    esTautología [p, q, no r]  ==>  False
--    esTautología []            ==>  False
-- ------------------------------------------------------------------

esTautología :: Cláusula -> Bool
esTautología c =
    [f | f <- c, elem (complementario f) c] /= []
```

```
-- ----------------------------------------------------------------------
-- Ejercicio 2: Definir la función
--    eliminaTautologías :: [Cláusula] -> [Cláusula]
-- tal que (eliminaTautologías s) es el conjunto obtenido eliminando las
-- tautologías de s. Por ejemplo,
--    eliminaTautologías [[p, q], [p, q, no p]]  ==>  [[p,q]]
-- ----------------------------------------------------------------------

eliminaTautologías :: [Cláusula] -> [Cláusula]
eliminaTautologías s =
    [c | c <- s, not (esTautología c)]

-- ----------------------------------------------------------------------
-- Eliminación de cláusulas unitarias                                   --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 3: Definir la función
--    esUnitaria :: Cláusula -> Bool
-- tal que (esUnitaria c) se verifica si la cláusula c es unitaria . Por
-- ejemplo,
--    esUnitaria [p]     ==>  True
--    esUnitaria [no p]  ==>  True
--    esUnitaria [p, q]  ==>  False
-- ----------------------------------------------------------------------

esUnitaria :: Cláusula -> Bool
esUnitaria [_] = True
esUnitaria _   = False

-- ----------------------------------------------------------------------
-- Ejercicio 4: Definir la función
--    eliminaCláusulaUnitaria :: Literal -> [Cláusula] -> [Cláusula]
-- tal que (eliminaCláusulaUnitaria l s) es el conjunto obtenido al
-- reducir s por la eliminación de la cláusula unitaria formada por el
-- literal l. Por ejemplo,
--    eliminaCláusulaUnitaria (no p) [[p,q,no r],[p,no q],[no p],[r]]
--    ==> [[q,no r],[no q],[r]]
--    eliminaCláusulaUnitaria (no q) [[q,no r],[no q],[r]]
--    ==> [[no r],[r]]
```

```
--    eliminaCláusulaUnitaria (no r) [[no r],[r],[p]]
--    ==> [[],[p]]
-- ---------------------------------------------------------------------

eliminaCláusulaUnitaria :: Literal -> [Cláusula] -> [Cláusula]
eliminaCláusulaUnitaria l s =
    [delete (complementario l) c | c <- s, notElem l c]

-- ---------------------------------------------------------------------
-- Ejercicio 5: Definir la función
--    cláusulaUnitaria :: [Cláusula] -> Maybe Literal
-- tal que (cláusulaUnitaria s) es la primera cláusula unitaria de s, si
-- s tiene cláusulas unitarias y nada en caso contrario. Por ejemplo,
--    cláusulaUnitaria [[p,q],[p],[no q]]  ==>  Just p
--    cláusulaUnitaria [[p,q],[p,no q]]    ==>  Nothing
-- ---------------------------------------------------------------------

cláusulaUnitaria :: [Cláusula] -> Maybe Literal
cláusulaUnitaria [] = Nothing
cláusulaUnitaria (c:cs)
    | esUnitaria c = Just (head c)
    | otherwise    = cláusulaUnitaria cs

-- ---------------------------------------------------------------------
-- Ejercicio 6: Definir la función
--    eliminaCláusulasUnitarias :: [Cláusula] -> [Cláusula]
-- tal que (eliminaCláusulasUnitarias s) es el conjunto obtenido
-- aplicando el proceso de eliminación de cláusulas unitarias a s. Por
-- ejemplo,
--    eliminaCláusulasUnitarias [[p,q,no r],[p,no q],[no p],[r],[u]]
--    ==> [[],[u]]
--    eliminaCláusulasUnitarias [[p,q],[no q],[no p,q,no r]]
--    ==> []
--    eliminaCláusulasUnitarias [[no p,q],[p],[r,u]]
--    ==> [[r,u]]
-- ---------------------------------------------------------------------

eliminaCláusulasUnitarias :: [Cláusula] -> [Cláusula]
eliminaCláusulasUnitarias s
    | elem [] s                    = s
```

```
    | cláusulaUnitaria s == Nothing = s
    | otherwise                    =
        eliminaCláusulasUnitarias (eliminaCláusulaUnitaria c s)
        where Just c = cláusulaUnitaria s

-- ---------------------------------------------------------------------
-- Eliminación de literales puros                                      --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 7: Definir la función
--    literales :: [Cláusula] -> [Literal]
-- tal que (literales c) es el conjunto de literales de s. Por ejemplo,
--    literales [[p,q],[p,q,no p]]  ==>  [p,q,no p]
-- ---------------------------------------------------------------------

literales :: [Cláusula] -> [Literal]
literales s =
    unión General s

-- ---------------------------------------------------------------------
-- Ejercicio 8: Definir la función
--    esLiteralPuro :: Literal -> [Cláusula] -> Bool
-- tal que (esLiteralPuro l s) se verifica si l es puro en s. Por
-- ejemplo,
--    esLiteralPuro p [[p,q],[p,no q],[r,q],[r,no q]]  ==>  True
--    esLiteralPuro q [[p,q],[p,no q],[r,q],[r,no q]]  ==>  False
-- ---------------------------------------------------------------------

esLiteralPuro :: Literal -> [Cláusula] -> Bool
esLiteralPuro l s =
    and [notElem l' c | c <- s]
    where l' = complementario l

-- ---------------------------------------------------------------------
-- Ejercicio 9: Definir la función
--    eliminaLiteralPuro :: Literal -> [Cláusula] -> [Cláusula]
-- tal que (eliminaLiteralPuro l s) es el conjunto obtenido eliminando
-- el literal puro l de s. Por ejemplo,
--    eliminaLiteralPuro p [[p,q],[p,no q],[r,q],[r,no q]]
```

```
--    ==> [[r,q],[r,no q]]
--    eliminaLiteralPuro r [[r,q],[r,no q]]
--    ==> []
-- ---------------------------------------------------------------------

eliminaLiteralPuro :: Literal -> [Cláusula] -> [Cláusula]
eliminaLiteralPuro l s =
    [c | c <- s, notElem l c]

-- ---------------------------------------------------------------------
-- Ejercicio 10: Definir la función
--    literalesPuros :: [Cláusula] -> [Literal]
-- tal que (literalesPuros s) es el conjunto de los literales puros de
-- s. Por ejemplo,
--    literalesPuros [[p,q],[p,no q],[r,q],[r,no q]]  ==>  [p,r]
-- ---------------------------------------------------------------------

literalesPuros :: [Cláusula] -> [Literal]
literalesPuros s =
    [l | l <- literales s, esLiteralPuro l s]

-- ---------------------------------------------------------------------
-- Ejercicio 11: Definir la función
--    eliminaLiteralesPuros :: [Cláusula] -> [Cláusula]
-- tal que (eliminaLiteralesPuros s) es el conjunto obtenido aplicando a
-- s el proceso de eliminación de literales puros. Por ejemplo,
--    eliminaLiteralesPuros [[p,q],[p,no q],[r,q],[r,no q]]
--    ==> []
--    eliminaLiteralesPuros [[p,q],[r,no s],[no r,s]]
--    ==> [[r,no s],[no r,s]]
-- ---------------------------------------------------------------------

eliminaLiteralesPuros :: [Cláusula] -> [Cláusula]
eliminaLiteralesPuros s
    | null lp   = s
    | otherwise =
        eliminaLiteralesPuros (eliminaLiteralPuro (head lp) s)
    where lp = literalesPuros s

-- ---------------------------------------------------------------------
```

```
-- Bifurcación                                                          --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 12: Definir la función
--    bifurcación :: [Cláusula] -> Literal -> ([Cláusula],[Cláusula])
-- tal que (bifurcación s l) es la bifurcación de s según el literal
-- l. Por ejemplo,
--    bifurcación [[p,no q],[no p,q],[q,no r],[no q,no r]] p
--    ==> ([[no q],[q,no r],[no q,no r]],[[q],[q,no r],[no q,no r]])
-- ---------------------------------------------------------------------

bifurcación :: [Cláusula] -> Literal -> ([Cláusula],[Cláusula])
bifurcación s l =
    ([delete l c  | c <- s, elem l c]  ++ cláusulas_sin_l_ni_l',
     [delete l' c | c <- s, elem l' c] ++ cláusulas_sin_l_ni_l')
    where l'                    = complementario l
          cláusulas_sin_l_ni_l' = [c | c <- s, notElem l c, notElem l' c]

-- ---------------------------------------------------------------------
-- Algoritmo de inconsistencia de Davis y Putnam                        --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 13: Definir la función
--    tieneCláusulasUnitarias :: [Cláusula] -> Bool
-- tal que (tieneCláusulasUnitarias s) se verifica si s tiene cláusulas
-- unitarias. Por ejemplo,
--    tieneCláusulasUnitarias [[p,q],[p],[no q]]  ==>  True
--    tieneCláusulasUnitarias [[p,q],[p,no q]]    ==>  False
-- ---------------------------------------------------------------------

tieneCláusulasUnitarias :: [Cláusula] -> Bool
tieneCláusulasUnitarias s =
    cláusulaUnitaria s /= Nothing

-- ---------------------------------------------------------------------
-- Ejercicio 14: Definir la función
--    tieneLiteralesPuros :: [Cláusula] -> Bool
-- tal que (tieneLiteralesPuros s) se verifica si s tiene literales
```

```
-- puros. Por ejemplo,
--    tieneLiteralesPuros [[p,q],[p,no q],[r,q],[r,no q]]
--    ==> True
--    tieneLiteralesPuros [[p,q],[no p,no q],[no r,q],[r,no q]]
--    ==> False
-- ----------------------------------------------------------------------

tieneLiteralesPuros :: [Cláusula] -> Bool
tieneLiteralesPuros s =
    literalesPuros s /= []

-- ----------------------------------------------------------------------
-- Ejercicio 15: Definir la función
--    esInconsistentePorDP :: [Cláusula] -> Bool
-- tal que (esInconsistentePorDP s) se verifica si s es inconsistente
-- mediante el algoritmo de inconsistencia de Davis y Putnam. Por
-- ejemplo,
--    esInconsistentePorDP [[p,q],[p,q,no p]]
--    ==> False
--    esInconsistentePorDP [[p,q,no r],[p,no q],[no p],[r],[u]]
--    ==> True
--    esInconsistentePorDP [[p,q],[no q],[no p,q,no r]]
--    ==> False
--    esInconsistentePorDP [[no p,q],[p],[r,u]]
--    ==> False
--    esInconsistentePorDP [[p,q],[p,no q],[r,q],[r,no q]]
--    ==> False
--    esInconsistentePorDP [[p,q],[r,no s],[no r,s]]
--    ==> False
--    esInconsistentePorDP [[p,no q],[no p,q],[q,no r],[no q,no r]]
--    ==> False
--    esInconsistentePorDP [[p,q],[p,no q],[r,q],[r,no q]]
--    ==> False
-- ----------------------------------------------------------------------

esInconsistentePorDP :: [Cláusula] -> Bool
esInconsistentePorDP s =
    esInconsistentePorDP' (eliminaTautologías s)

esInconsistentePorDP' :: [Cláusula] -> Bool
```

```
esInconsistentePorDP' s
    | null s = False
    | elem [] s = True
    | tieneCláusulasUnitarias s =
        esInconsistentePorDP' (eliminaCláusulasUnitarias s)
    | tieneLiteralesPuros s =
        esInconsistentePorDP' (eliminaLiteralesPuros s)
    | otherwise =
        (esInconsistentePorDP' s1) && (esInconsistentePorDP' s2)
            where l = head (head s)
                  (s1,s2) = bifurcación s l

-- ------------------------------------------------------------------
-- Decisión de validez mediante Davis y Putnam                      --
-- ------------------------------------------------------------------

-- ------------------------------------------------------------------
-- Ejercicio 16: Definir la función
--    esVálidaPorDP :: Prop -> Bool
-- tal que (esVálidaPorDP f) se verifica si f es válida mediante el
-- algoritmo de Davis y Putnam. Por ejemplo,
--    esVálidaPorDP (p --> p)                ==>  True
--    esVálidaPorDP ((p --> q) \/ (q --> p))  ==>  True
--    esVálidaPorDP (p --> q)                ==>  False
-- ------------------------------------------------------------------

esVálidaPorDP :: Prop -> Bool
esVálidaPorDP f =
    esInconsistentePorDP (cláusulas (Neg f))

-- ------------------------------------------------------------------
-- Decisión de consecuencia mediante Davis y Putnam                 --
-- ------------------------------------------------------------------

-- ------------------------------------------------------------------
-- Ejercicio 17: Definir la función
--    esConsecuenciaPorDP :: [Prop] -> Prop -> Bool
-- tal que (esConsecuenciaPorDP s f) se verifica si f es consecuencia de
-- s mediante el algoritmo de Davis y Putnam. Por ejemplo,
--    esConsecuenciaPorDP [p --> q, q --> r] (p --> r)  ==>  True
```

```
--    esConsecuenciaPorDP [p] (p /\ q)  ==>  False
-- ---------------------------------------------------------------------

esConsecuenciaPorDP :: [Prop] -> Prop -> Bool
esConsecuenciaPorDP s f =
    esInconsistentePorDP (cláusulasConjunto ((Neg f):s))

-- ---------------------------------------------------------------------
-- Verificación                                                        --
-- ---------------------------------------------------------------------

ejemplosDavisPutnam :: [Test]
ejemplosDavisPutnam =
    ["esTautologia ej 1" ~:
     esTautología [p, q, no p]
     ==> True,
     "esTautologia ej 2" ~:
     esTautología [p, q, no r]
     ==> False,
     "esTautologia ej 3" ~:
     esTautología []
     ==> False,
     "eliminaTautologias ej 1" ~:
     eliminaTautologías [[p, q], [p, q, no p]]
     ==> [[p,q]],
     "esUnitaria ej 1" ~:
     esUnitaria [p]
     ==> True,
     "esUnitaria ej 2" ~:
     esUnitaria [no p]
     ==> True,
     "esUnitaria ej 3" ~:
     esUnitaria [p, q]
     ==> False,
     "eliminaClausulaUnitaria ej 1" ~:
     eliminaCláusulaUnitaria (no p) [[p,q,no r],[p,no q],[no p],[r]]
     ==> [[q,no r],[no q],[r]],
     "eliminaClausulaUnitaria ej 2" ~:
     eliminaCláusulaUnitaria (no q) [[q,no r],[no q],[r]]
     ==> [[no r],[r]],
```

```
"eliminaClausulaUnitaria ej 3" ~:
eliminaCláusulaUnitaria (no r) [[no r],[r],[p]]
==> [[],[p]],
"clausulaUnitaria ej 1" ~:
cláusulaUnitaria [[p,q],[p],[no q]]
==> Just p,
"clausulaUnitaria ej 2" ~:
cláusulaUnitaria [[p,q],[p,no q]]
==> Nothing,
"eliminaClausulasUnitarias ej 1" ~:
eliminaCláusulasUnitarias [[p,q,no r],[p,no q],[no p],[r],[u]]
==> [[],[u]],
"eliminaClausulasUnitarias ej 2" ~:
eliminaCláusulasUnitarias [[p,q],[no q],[no p,q,no r]]
==> [],
"eliminaClausulasUnitarias ej 3" ~:
eliminaCláusulasUnitarias [[no p,q],[p],[r,u]]
==> [[r,u]],
"literales ej 1" ~:
literales [[p,q],[p,q,no p]]
==> [p,q,no p],
"esLiteralPuro ej 1" ~:
esLiteralPuro p [[p,q],[p,no q],[r,q],[r,no q]]
==> True,
"esLiteralPuro ej 2" ~:
esLiteralPuro q [[p,q],[p,no q],[r,q],[r,no q]]
==> False,
"eliminaLiteralPuro ej 1" ~:
eliminaLiteralPuro p [[p,q],[p,no q],[r,q],[r,no q]]
==> [[r,q],[r,no q]],
"eliminaLiteralPuro ej 2" ~:
eliminaLiteralPuro r [[r,q],[r,no q]]
==> [],
"literalesPuros ej 1" ~:
literalesPuros [[p,q],[p,no q],[r,q],[r,no q]]
==> [p,r],
"eliminaLiteralesPuros ej 1" ~:
eliminaLiteralesPuros [[p,q],[p,no q],[r,q],[r,no q]]
==> [],
"eliminaLiteralesPuros ej 2" ~:
```

```
eliminaLiteralesPuros [[p,q],[r,no s],[no r,s]]
==> [[r,no s],[no r,s]],
"bifurcacion ej 1" ~:
bifurcación [[p,no q],[no p,q],[q,no r],[no q,no r]] p
==> ([[no q],[q,no r],[no q,no r]],[[q],[q,no r],[no q,no r]]),
"tieneClausulasUnitarias ej 1" ~:
tieneCláusulasUnitarias [[p,q],[p],[no q]]
==> True,
"tieneClausulasUnitarias ej 2" ~:
tieneCláusulasUnitarias [[p,q],[p,no q]]
==> False,
"tieneLiteralesPuros ej 1" ~:
tieneLiteralesPuros [[p,q],[p,no q],[r,q],[r,no q]]
==> True,
"tieneLiteralesPuros ej 2" ~:
tieneLiteralesPuros [[p,q],[no p,no q],[no r,q],[r,no q]]
==> False,
"esInconsistentePorDP ej 1" ~:
esInconsistentePorDP [[p,q],[p,q,no p]]
==> False,
"esInconsistentePorDP ej 2" ~:
esInconsistentePorDP [[p,q,no r],[p,no q],[no p],[r],[u]]
==> True,
"esInconsistentePorDP ej 3" ~:
esInconsistentePorDP [[p,q],[no q],[no p,q,no r]]
==> False,
"esInconsistentePorDP ej 4" ~:
esInconsistentePorDP [[no p,q],[p],[r,u]]
==> False,
"esInconsistentePorDP ej 5" ~:
esInconsistentePorDP [[p,q],[p,no q],[r,q],[r,no q]]
==> False,
"esInconsistentePorDP ej 6" ~:
esInconsistentePorDP [[p,q],[r,no s],[no r,s]]
==> False,
"esInconsistentePorDP ej 7" ~:
esInconsistentePorDP [[p,no q],[no p,q],[q,no r],[no q,no r]]
==> False,
"esInconsistentePorDP ej 8" ~:
esInconsistentePorDP [[p,q],[p,no q],[r,q],[r,no q]]
```

```
     ==> False,
     "esValidaPorDP ej 1" ~:
     esVálidaPorDP (p --> p)
     ==> True,
     "esValidaPorDP ej 2" ~:
     esVálidaPorDP ((p --> q) \/ (q --> p))
     ==> True,
     "esValidaPorDP ej 3" ~:
     esVálidaPorDP (p --> q)
     ==> False,
     "esConsecuenciaPorDP ej 1" ~:
     esConsecuenciaPorDP [p --> q, q --> r] (p --> r)
     ==> True,
     "esConsecuenciaPorDP ej 2" ~:
     esConsecuenciaPorDP [p] (p /\ q)
     ==> False
    ]

verificaDavisPutnam :: IO Counts
verificaDavisPutnam =
    runTestTT (test ejemplosDavisPutnam)

-- DavisPutnam> verificaDavisPutnam
-- Cases: 41  Tried: 41  Errors: 0  Failures: 0
```

Capítulo 7

Resolución proposicional

```
module ResolucionProposicional where

import SintaxisSemantica
import FormasNormales
import Clausulas
import DavisPutnam
import Data.List
import Test.HUnit
import Verificacion

-- ----------------------------------------------------------------------
-- Resolventes                                                          --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 1: Definir la función
--    resolvente :: Cláusula -> Cláusula -> Literal -> Cláusula
-- tal que (resolvente c1 c2 l) es la resolvente de c1 y c2 respecto del
-- literal l. Por ejemplo,
--    resolvente [no p,q] [no q,r] q  ==>  [no p,r]
--    resolvente [no p,no q] [q,r] (no q)  ==>  [no p,r]
--    resolvente [no p,q] [no p,no q] q  ==>  [no p]
-- ----------------------------------------------------------------------

resolvente :: Cláusula -> Cláusula -> Literal -> Cláusula
resolvente c1 c2 l =
    union (delete l c1) (delete (complementario l) c2)
```

```
-- ---------------------------------------------------------------------
-- Ejercicio 2: Definir la función
--    resolventes :: Cláusula -> Cláusula -> [Cláusula]
-- tal que (resolventes c1 c2) es el conjunto de las resolventes de c1 y
-- c2. Por ejemplo,
--    resolventes [no p,q] [p,no q]  ==>  [[q,no q],[no p,p]]
--    resolventes [no p,q] [p,q]     ==>  [[q]]
-- ---------------------------------------------------------------------

resolventes :: Cláusula -> Cláusula -> [Cláusula]
resolventes c1 c2 =
    [resolvente c1 c2 l | l <- c1, (complementario l) `elem` c2]

-- ---------------------------------------------------------------------
-- Ejercicio 3: Definir la función
--    resolventesCláusulaConjunto :: Cláusula -> [Cláusula] -> [Cláusula]
-- tal que (resolventes c s) es el conjunto de las resolventes de c y
-- s. Por ejemplo,
--    resolventesCláusulaConjunto [no p,q] [[p,q],[p,r],[no q,s]]
--    ==> [[q],[q,r],[no p,s]]
-- ---------------------------------------------------------------------

resolventesCláusulaConjunto :: Cláusula -> [Cláusula] -> [Cláusula]
resolventesCláusulaConjunto c s =
    unión General [resolventes c c1 | c1 <- s]

-- ---------------------------------------------------------------------
-- Decisión de inconsistencia por resolución                           --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 4: Definir la función
--    esInconsistentePorResolución :: [Cláusula] -> Bool
-- tal que (esInconsistentePorResolución s) se verifica si s es
-- inconsistente mediante resolución. Por ejemplo,
--    esInconsistentePorResolución [[p],[no p,q],[no q]]
--    ==> True
--    esInconsistentePorResolución [[p],[no p,q]]
--    ==> False
```

```
--    esInconsistentePorResolución [[p,q],[no p,q],[p,no q],[no p,no q]]
--    ==> True
--    esInconsistentePorResolución [[p,q],[p,r],[no q,no r],[no p]]
--    ==> True
-- ---------------------------------------------------------------------

esInconsistentePorResolución :: [Cláusula] -> Bool
esInconsistentePorResolución s =
    esInconsistentePorResolución' s []

esInconsistentePorResolución' :: [Cláusula] -> [Cláusula] -> Bool
esInconsistentePorResolución' soporte usables
    | null soporte    = False
    | elem [] soporte = True
    | otherwise       =
        esInconsistentePorResolución' soporte' usables'
        where actual   = head soporte
              usables' = union [actual] usables
              soporte' = union (tail soporte)
                               [c
                                | c <- resolventesCláusulaConjunto
                                       actual
                                       usables'
                                , not (esTautología c)
                                , notElem c soporte
                                , notElem c usables']

-- ---------------------------------------------------------------------
-- Validez mediante resolución                                         --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 5: Definir la función
--    esVálidaPorResolución :: Prop -> Bool
-- tal que (esVálidaPorResolución f) se verifica si f es válida por
-- resolución. Por ejemplo,
--    esVálidaPorResolución (p --> p)                ==>  True
--    esVálidaPorResolución ((p --> q) \/ (q --> p))  ==>  True
--    esVálidaPorResolución (p --> q)                ==>  False
-- ---------------------------------------------------------------------
```

```
esVálidaPorResolución :: Prop -> Bool
esVálidaPorResolución f =
    esInconsistentePorResolución
    (eliminaTautologías (cláusulas (Neg f)))

-- ---------------------------------------------------------------------
-- Consecuencia mediante resolución                                    --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 6: Definir la función
--    esConsecuenciaPorResolución :: [Prop] -> Prop -> Bool
-- tal que (esConsecuenciaPorResolución s f) se verifica si f es
-- consecuencia de s mediante el método de resolución. Por ejemplo,
--    esConsecuenciaPorResolución [p --> q, q --> r] (p --> r)
--    ==> True
--    esConsecuenciaPorResolución [p --> q, q --> r] (p <--> r)
--    ==> False
-- ---------------------------------------------------------------------

esConsecuenciaPorResolución :: [Prop] -> Prop -> Bool
esConsecuenciaPorResolución s f =
    esInconsistentePorResolución
    (eliminaTautologías (cláusulasConjunto ((Neg f):s)))

-- ---------------------------------------------------------------------
-- Verificación                                                        --
-- ---------------------------------------------------------------------

ejemplosResolucionProposicional :: [Test]
ejemplosResolucionProposicional =
    ["resolvente ej 1" ~:
     resolvente [no p,q] [no q,r] q
     ==> [no p,r],
     "resolvente ej 2" ~:
     resolvente [no p,no q] [q,r] (no q)
     ==> [no p,r],
     "resolvente ej 3" ~:
     resolvente [no p,q] [no p,no q] q
```

```
    ==> [no p],
    "resolventes ej 1" ~:
    resolventes [no p,q] [p,no q]
    ==> [[q,no q],[no p,p]],
    "resolventes ej 2" ~:
    resolventes [no p,q] [p,q]
    ==> [[q]],
    "resolventesClausulaConjunto ej 1" ~:
    resolventesCláusulaConjunto [no p,q] [[p,q],[p,r],[no q,s]]
    ==> [[q],[q,r],[no p,s]],
    "esInconsistentePorResolucion ej 1" ~:
    esInconsistentePorResolución [[p],[no p,q],[no q]]
    ==> True,
    "esInconsistentePorResolucion ej 2" ~:
    esInconsistentePorResolución [[p],[no p,q]]
    ==> False,
    "esInconsistentePorResolucion ej 3" ~:
    esInconsistentePorResolución [[p,q],[no p,q],[p,no q],[no p,no q]]
    ==> True,
    "esInconsistentePorResolucion ej 4" ~:
    esInconsistentePorResolución [[p,q],[p,r],[no q,no r],[no p]]
    ==> True,
    "esValidaPorResolucion ej 1" ~:
    esVálidaPorResolución (p --> p)
    ==> True,
    "esValidaPorResolucion ej 2" ~:
    esVálidaPorResolución ((p --> q) \/ (q --> p))
    ==> True,
    "esValidaPorResolucion ej 3" ~:
    esVálidaPorResolución (p --> q)
    ==> False,
    "esConsecuenciaPorResolucion ej 1" ~:
    esConsecuenciaPorResolución [p --> q, q --> r] (p --> r)
    ==> True,
    "esConsecuenciaPorResolucion ej 2" ~:
    esConsecuenciaPorResolución [p --> q, q --> r] (p <--> r)
    ==> False
   ]

verificaResolucionProposicional :: IO Counts
```

```
verificaResolucionProposicional =
    runTestTT (test ejemplosResolucionProposicional)

-- ResolucionProposicional> verificaResolucionProposicional
-- Cases: 15  Tried: 15  Errors: 0  Failures: 0
```

Capítulo 8

Refinamientos de resolución

```
module RefinamientosResolucion where

import SintaxisSemantica
import FormasNormales
import Clausulas
import DavisPutnam
import ResolucionProposicional
import Data.List
import Debug.Trace
import Test.HUnit
import Verificacion

-- ----------------------------------------------------------------------
-- Resolución positiva                                                  --
-- ----------------------------------------------------------------------

-- Resolventes positivas                                                --
-- ---------------------

-- ----------------------------------------------------------------------
-- Ejercicio 1: Definir la función
--    positivo :: Literal -> Bool
-- tal que (positivo l) se verifica si el literal l es positivo. Por
-- ejemplo,
--    positivo p       ==>  True
--    positivo (no p)  ==>  False
-- ----------------------------------------------------------------------
```

```
positivo :: Literal -> Bool
positivo (Atom _) = True
positivo _        = False

-- ---------------------------------------------------------------------
-- Ejercicio 2: Definir la función
--    positiva :: Cláusula -> Bool
-- tal que (positiva c) se verifica si la cláusula c es positiva. Por
-- ejemplo,
--    positiva [p,q]      ==>  True
--    positiva [p, no q]  ==>  False
-- ---------------------------------------------------------------------

positiva :: Cláusula -> Bool
positiva c =
    and [positivo l | l <- c]

-- ---------------------------------------------------------------------
-- Ejercicio 3: Definir la función
--    resPosCláusulaConjunto :: Cláusula -> [Cláusula] -> [Cláusula]
-- tal que (resPosCláusulaConjunto c s) es el conjunto de las
-- resolventes positivas de c con las cláusulas de s. Por ejemplo,
--    resPosCláusulaConjunto [no p,q] [[p,q],[p,r],[no q,s]]
--    ==> [[q],[q,r]]
-- ---------------------------------------------------------------------

resPosCláusulaConjunto :: Cláusula -> [Cláusula] -> [Cláusula]
resPosCláusulaConjunto c s
    | positiva c = resolventesCláusulaConjunto c s
    | otherwise  = unión General [resolventes c c1 | c1 <- s, positiva c1]

-- Decisión de inconsistencia por resolución positiva
-- --------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 4: Definir la función
--    esInconsistentePorResoluciónPositiva :: [Cláusula] -> Bool
-- tal que (esInconsistentePorResoluciónPositiva s) se verifica si s es
-- inconsistente mediante el método de resolución positiva. Por ejemplo,
```

```
--    esInconsistentePorResoluciónPositiva [[p],[no p,q],[no q]]
--    ==> True
--    esInconsistentePorResoluciónPositiva [[p],[no p,q]]
--    ==> False
--    esInconsistentePorResoluciónPositiva [[p,q],[p,r],[no q,no r],[no p]]
--    ==> True
-- ---------------------------------------------------------------------

esInconsistentePorResoluciónPositiva :: [Cláusula] -> Bool
esInconsistentePorResoluciónPositiva s =
    esInconsistentePorResoluciónPositiva' s []

esInconsistentePorResoluciónPositiva' :: [Cláusula] -> [Cláusula] -> Bool
esInconsistentePorResoluciónPositiva' soporte usables
    | null soporte    = False
    | elem [] soporte = True
    | otherwise       =
        esInconsistentePorResoluciónPositiva' soporte' usables'
        where actual   = head soporte
              usables' = union [actual] usables
              soporte' = union (tail soporte)
                               [c
                                | c <- resPosCláusulaConjunto
                                       actual
                                       usables'
                                , not (esTautología c)
                                , notElem c soporte
                                , notElem c usables']

-- Validez mediante resolución positiva
-- ------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 5: Definir la función
--    esVálidaPorResoluciónPositiva :: Prop -> Bool
-- tal que (esVálidaPorResoluciónPositiva f) se verifica si f es válida
-- por resolución positiva. Por ejemplo,
--    esVálidaPorResoluciónPositiva (p --> p)                ==>  True
--    esVálidaPorResoluciónPositiva ((p --> q) \/ (q --> p))  ==>  True
--    esVálidaPorResoluciónPositiva (p --> q)                ==>  False
```

```
-- ----------------------------------------------------------------------

esVálidaPorResoluciónPositiva :: Prop -> Bool
esVálidaPorResoluciónPositiva f =
    esInconsistentePorResoluciónPositiva
    (eliminaTautologías (cláusulas (Neg f)))

-- Consecuencia mediante resolución positiva
-- -----------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 6: Definir la función
--    esConsecuenciaPorResoluciónPositiva :: [Prop] -> Prop -> Bool
-- tal que (esConsecuenciaPorResoluciónPositiva s f) se verifica si f es
-- consecuencia de s mediante el método de resolución positiva. Por
-- ejemplo,
--    esConsecuenciaPorResoluciónPositiva [p --> q, q --> r] (p --> r)
--    ==> True
--    esConsecuenciaPorResoluciónPositiva [p --> q, q --> r] (p <--> r)
--    ==> False
-- ----------------------------------------------------------------------

esConsecuenciaPorResoluciónPositiva :: [Prop] -> Prop -> Bool
esConsecuenciaPorResoluciónPositiva s f =
    esInconsistentePorResoluciónPositiva
    (eliminaTautologías (cláusulasConjunto ((Neg f):s)))

-- ----------------------------------------------------------------------
-- Resolución negativa                                                  --
-- ----------------------------------------------------------------------

-- Resolventes negativas
-- ---------------------

-- ----------------------------------------------------------------------
-- Ejercicio 7: Definir la función
--    negativo :: Literal -> Bool
-- tal que (negativo l) se verifica si el literal l es negativo. Por
-- ejemplo,
--    negativo p       ==>  False
```

```
--    negativo (no p)  ==>  True
-- ---------------------------------------------------------------------

negativo :: Literal -> Bool
negativo (Neg (Atom _)) = True
negativo _              = False

-- ---------------------------------------------------------------------
-- Ejercicio 8: Definir la función
--    negativa :: Cláusula -> Bool
-- tal que (negativa c) se verifica si la cláusula c es negativa. Por
-- ejemplo,
--    negativa [no p,no q]  ==>  True
--    negativa [p, no q]    ==>  False
-- ---------------------------------------------------------------------

negativa :: Cláusula -> Bool
negativa c =
    and [negativo l | l <- c]

-- ---------------------------------------------------------------------
-- Ejercicio 9: Definir la función
--    resNegCláusulaConjunto :: Cláusula -> [Cláusula] -> [Cláusula]
-- tal que (resNegCláusulaConjunto c s) es el conjunto de las
-- resolventes negativas de c con las cláusulas de s. Por ejemplo,
--    resNegCláusulaConjunto [no p,q] [[p,q],[p,r],[no q,s]]
--    ==> []
--    resNegCláusulaConjunto [no p,q] [[p,q],[p,r],[no q,no s]]
--    ==> [[no p,no s]]
-- ---------------------------------------------------------------------

resNegCláusulaConjunto :: Cláusula -> [Cláusula] -> [Cláusula]
resNegCláusulaConjunto c s
    | negativa c = resolventesCláusulaConjunto c s
    | otherwise  = unión General [resolventes c c1 | c1 <- s, negativa c1]

-- Decisión de inconsistencia por resolución negativa
-- ------------------------------------------------

-- ---------------------------------------------------------------------
```

```
-- Ejercicio 10: Definir la función
--    esInconsistentePorResoluciónNegativa :: [Cláusula] -> Bool
-- tal que (esInconsistentePorResoluciónNegativa s) se verifica si s es
-- inconsistente mediante el método de resolución negativa. Por ejemplo,
--    esInconsistentePorResoluciónNegativa [[p],[no p,q],[no q]]
--    ==> True
--    esInconsistentePorResoluciónNegativa [[p],[no p,q]]
--    ==> False
--    esInconsistentePorResoluciónNegativa [[p,q],[p,r],[no q,no r],[no p]]
--    ==> True
-- ----------------------------------------------------------------------

esInconsistentePorResoluciónNegativa :: [Cláusula] -> Bool
esInconsistentePorResoluciónNegativa s =
    esInconsistentePorResoluciónNegativa' s []

esInconsistentePorResoluciónNegativa' :: [Cláusula] -> [Cláusula] -> Bool
esInconsistentePorResoluciónNegativa' soporte usables
    | null soporte    = False
    | elem [] soporte = True
    | otherwise       =
        esInconsistentePorResoluciónNegativa' soporte' usables'
        where actual   = head soporte
              usables' = union [actual] usables
              soporte' = union (tail soporte)
                               [c
                                | c <- resNegCláusulaConjunto
                                       actual
                                       usables'
                                , not (esTautología c)
                                , notElem c soporte
                                , notElem c usables']

-- Validez mediante resolución negativa
-- ------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 11: Definir la función
--    esVálidaPorResoluciónNegativa :: Prop -> Bool
-- tal que (esVálidaPorResoluciónNegativa f) se verifica si f es válida
```

```
-- por resolución negativa. Por ejemplo,
--    esVálidaPorResoluciónNegativa (p --> p)                 ==>  True
--    esVálidaPorResoluciónNegativa ((p --> q) \/ (q --> p))  ==>  True
--    esVálidaPorResoluciónNegativa (p --> q)                 ==>  False
-- ---------------------------------------------------------------------

esVálidaPorResoluciónNegativa :: Prop -> Bool
esVálidaPorResoluciónNegativa f =
    esInconsistentePorResoluciónNegativa
    (eliminaTautologías (cláusulas (Neg f)))

-- Consecuencia mediante resolución negativa
-- -----------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 12: Definir la función
--    esConsecuenciaPorResoluciónNegativa :: [Prop] -> Prop -> Bool
-- tal que (esConsecuenciaPorResoluciónNegativa s f) se verifica si f es
-- consecuencia de s mediante el método de resolución negativa. Por
-- ejemplo,
--    esConsecuenciaPorResoluciónNegativa [p --> q, q --> r] (p --> r)
--    ==> True
-- ---------------------------------------------------------------------

esConsecuenciaPorResoluciónNegativa :: [Prop] -> Prop -> Bool
esConsecuenciaPorResoluciónNegativa s f =
    esInconsistentePorResoluciónNegativa
    (eliminaTautologías (cláusulasConjunto ((Neg f):s)))

-- ---------------------------------------------------------------------
-- Resolución unidad                                                   --
-- ---------------------------------------------------------------------

-- Resolventes unitarias
-- ---------------------

-- ---------------------------------------------------------------------
-- Ejercicio 13: Definir la función
--    unitaria :: Cláusula -> Bool
-- tal que (unitaria c) se verifica si c es unitaria. Por ejemplo,
```

```
--    unitaria [p]     ==>  True
--    unitaria [no p]  ==>  True
--    unitaria [p, q]  ==>  False
-- ---------------------------------------------------------------------

unitaria :: Cláusula -> Bool
unitaria [_] = True
unitaria _   = False

-- ---------------------------------------------------------------------
-- Ejercicio 14: Definir la función
--    resUniCláusulaConjunto :: Cláusula -> [Cláusula] -> [Cláusula]
-- tal que (resUniCláusulaConjunto c a) es el conjunto de las
-- resolventes unitarias de c con las cláusulas de s. Por ejemplo,
--    resUniCláusulaConjunto [no p,q] [[p,q],[p,r],[no q,s]]
--    ==> []
--    resUniCláusulaConjunto [no p,q] [[p,q],[p,r],[no q]]
--    ==> [[no p]]
-- ---------------------------------------------------------------------

resUniCláusulaConjunto :: Cláusula -> [Cláusula] -> [Cláusula]
resUniCláusulaConjunto c s
    | unitaria c = resolventesCláusulaConjunto c s
    | otherwise  = unión General [resolventes c c1 | c1 <- s, unitaria c1]

-- Decisión de inconsistencia por resolución unidad
-- ------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 15: Definir la función
--    esInconsistentePorResoluciónUnidad :: [Cláusula] -> Bool
-- tal que (esInconsistentePorResoluciónUnidad s) se verifica si s es
-- inconsistente mediante el método de resolución unidad. Por ejemplo,
--    esInconsistentePorResoluciónUnidad [[p],[no p,q],[no q]]
--    ==> True
--    esInconsistentePorResoluciónUnidad [[p],[no p,q]]
--    ==> False
--    esInconsistentePorResoluciónUnidad [[p,q],[p,r],[no q,no r],[no p]]
--    ==> True
-- ---------------------------------------------------------------------
```

```
esInconsistentePorResoluciónUnidad :: [Cláusula] -> Bool
esInconsistentePorResoluciónUnidad s =
    esInconsistentePorResoluciónUnidad' s []

esInconsistentePorResoluciónUnidad' :: [Cláusula] -> [Cláusula] -> Bool
esInconsistentePorResoluciónUnidad' soporte usables
    | null soporte    = False
    | elem [] soporte = True
    | otherwise       =
        esInconsistentePorResoluciónUnidad' soporte' usables'
        where actual   = head soporte
              usables' = union [actual] usables
              soporte' = union (tail soporte)
                               [c
                                | c <- resUniCláusulaConjunto
                                       actual
                                       usables'
                                , not (esTautología c)
                                , notElem c soporte
                                , notElem c usables']

-- Validez mediante resolución unidad
-- ----------------------------------

-- ------------------------------------------------------------------------
-- Ejercicio 16: Definir la función
--    esVálidaPorResoluciónUnidad :: Prop -> Bool
-- tal que (esVálidaPorResoluciónUnidad f) se verifica si f es válida
-- por resolución unitaria. Por ejemplo,
--    esVálidaPorResoluciónUnidad (p --> p)                ==>  True
--    esVálidaPorResoluciónUnidad ((p --> q) \/ (q --> p))  ==>  True
--    esVálidaPorResoluciónUnidad (p --> q)                ==>  False
-- ------------------------------------------------------------------------

esVálidaPorResoluciónUnidad :: Prop -> Bool
esVálidaPorResoluciónUnidad f =
    esInconsistentePorResoluciónUnidad
    (eliminaTautologías (cláusulas (Neg f)))
```

```
-- Consecuencia mediante resolución unidad
-- ---------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 17: Definir la función
--    esConsecuenciaPorResoluciónUnidad :: [Prop] -> Prop -> Bool
-- tal que (esConsecuenciaPorResoluciónUnidad s f) se verifica si f es
-- consecuencia de s mediante el método de resolución unitaria. Por
-- ejemplo,
--    esConsecuenciaPorResoluciónUnidad [p --> q, q --> r] (p --> r)
--    ==> True
-- ----------------------------------------------------------------------

esConsecuenciaPorResoluciónUnidad :: [Prop] -> Prop -> Bool
esConsecuenciaPorResoluciónUnidad s f =
    esInconsistentePorResoluciónUnidad (cláusulasConjunto ((Neg f):s))

-- ----------------------------------------------------------------------
-- Resolución por entrada                                               --
-- ----------------------------------------------------------------------

-- Resolventes por entrada
-- -----------------------

-- ----------------------------------------------------------------------
-- Ejercicio 18: Definir la función
--    resEntCláusulaConjunto :: [Cláusula] ->  Cláusula -> [Cláusula] ->
--                              [Cláusula]
-- tal que (resEntCláusulaConjunto ent c s) es el conjunto de las
-- resolventes por entradas de c con las cláusulas de s siendo ent el
-- conjunto de entrada. Por ejemplo,
--    resEntCláusulaConjunto [] [no p,q] [[p,q],[p,r],[no q,s]]
--    ==> []
--    resEntCláusulaConjunto [[p,r]] [no p,q] [[p,q],[p,r],[no q,s]]
--    ==> [[q,r]]
-- ----------------------------------------------------------------------

resEntCláusulaConjunto :: [Cláusula] ->  Cláusula -> [Cláusula] -> [Cláusula]
resEntCláusulaConjunto ent c s
    | elem c ent = resolventesCláusulaConjunto c s
```

```
    | otherwise  = uniónGeneral [resolventes c c1 | c1 <- s, elem c1 ent]

-- Decisión de inconsistencia por resolución por entrada
-- -----------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 19: Definir la función
--    esInconsistentePorResoluciónEntrada :: [Cláusula] -> [Cláusula] ->
--                                           Bool
-- tal que (esInconsistentePorResoluciónEntrada ent s) se verifica si s
-- es inconsistente mediante el método de resolución por entradas usando
-- el conjunto de entrada ent. Por ejemplo,
--    esInconsistentePorResoluciónEntrada [] [[p],[no p,q],[no q]]
--    ==> False
--    esInconsistentePorResoluciónEntrada [[p],[no p,q],[no q]]
--                                        [[p],[no p,q],[no q]]
--    ==> True
--    esInconsistentePorResoluciónEntrada [[p],[no p,q]] [[p],[no p,q]]
--    ==> False
-- ---------------------------------------------------------------------

esInconsistentePorResoluciónEntrada :: [Cláusula] -> [Cláusula] -> Bool
esInconsistentePorResoluciónEntrada ent s =
    esInconsistentePorResoluciónEntrada' ent s []

esInconsistentePorResoluciónEntrada' :: [Cláusula] ->
                                        [Cláusula] -> [Cláusula] -> Bool
esInconsistentePorResoluciónEntrada' ent soporte usables
    | null soporte    = False
    | elem [] soporte = True
    | otherwise       =
        esInconsistentePorResoluciónEntrada' ent soporte' usables'
        where actual   = head soporte
              usables' = union [actual] usables
              soporte' = union (tail soporte)
                               [c
                                | c <- resEntCláusulaConjunto
                                       ent
                                       actual
                                       usables'
```

```
                                , not (esTautología c)
                                , notElem c soporte
                                , notElem c usables']

-- Validez mediante resolución por entradas
-- ----------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 20: Definir la función
--    esVálidaPorResoluciónEntrada :: Prop -> Bool
-- tal que (esVálidaPorResoluciónEntrada f) se verifica si f es válida
-- por resolución por entradas. Por ejemplo,
--    esVálidaPorResoluciónEntrada (p --> p)                ==>  True
--    esVálidaPorResoluciónEntrada ((p --> q) \/ (q --> p))  ==>  True
--    esVálidaPorResoluciónEntrada (p --> q)                ==>  False
-- ----------------------------------------------------------------------

esVálidaPorResoluciónEntrada :: Prop -> Bool
esVálidaPorResoluciónEntrada f =
    esInconsistentePorResoluciónEntrada s s
    where s = eliminaTautologías (cláusulas (Neg f))

-- Consecuencia mediante resolución por entrada
-- --------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 21: Definir la función
--    esConsecuenciaPorResoluciónEntrada :: [Prop] -> Prop -> Bool
-- tal que (esConsecuenciaPorResoluciónEntrada s f) se verifica si f es
-- consecuencia de s mediante el método de resolución por entradas. Por
-- ejemplo,
--    esConsecuenciaPorResoluciónEntrada [p --> q, q --> r] (p --> r)
--    ==> True
-- ----------------------------------------------------------------------

esConsecuenciaPorResoluciónEntrada :: [Prop] -> Prop -> Bool
esConsecuenciaPorResoluciónEntrada s f =
    esInconsistentePorResoluciónEntrada cs cs
    where cs = (eliminaTautologías (cláusulasConjunto ((Neg f):s)))
```

```
-- ---------------------------------------------------------------------
-- Resolución lineal                                                   --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 22: Definir la función
--    esInconsistentePorResoluciónLineal :: [Cláusula] -> Cláusula ->
--                                          Bool
-- tal que (esInconsistentePorResoluciónLineal s c) se verifica si s es
-- inconsitente mediante el método de resolución lineal a partir de
-- c. Por ejemplo,
--  > esInconsistentePorResoluciónLineal [[no p,q],[p,no q],[no p,no q]] [p,q]
--    ==> [[no p,q],[p,no q],[no p,no q]]  [p,q]
--    ==> [[p,q],[no p,q],[p,no q],[no p,no q]]  [q]
--    ==> [[q],[p,q],[no p,q],[p,no q],[no p,no q]]  [p]
--    ==> [[p],[q],[p,q],[no p,q],[p,no q],[no p,no q]]  [no q]
--    ==> [[no q],[p],[q],[p,q],[no p,q],[p,no q],[no p,no q]]  []
--  True
-- ---------------------------------------------------------------------

esInconsistentePorResoluciónLineal :: [Cláusula] -> Cláusula -> Bool
-- esInconsistentePorResoluciónLineal s c
--     | trace ("==> " ++ show s ++ "  " ++ show c) False = undefined
esInconsistentePorResoluciónLineal s c
    | null s    = False
    | null c    = True
    | otherwise = or [esInconsistentePorResoluciónLineal (c:s) c1
                      | c1 <- resolventesCláusulaConjunto c s,
                        notElem c1 (c:s) ]

-- ---------------------------------------------------------------------
-- Verificación                                                        --
-- ---------------------------------------------------------------------

ejemplosRefinamientosResolucion :: [Test]
ejemplosRefinamientosResolucion =
    ["positivo ej 1" ~:
     positivo p
     ==> True,
     "positivo ej 2" ~:
```

```
positivo (no p)
==> False,
"positiva ej 1" ~:
positiva [p,q]
==> True,
"positiva ej 2" ~:
positiva [p, no q]
==> False,
"resPosClausulaConjunto ej 1" ~:
resPosCláusulaConjunto [no p,q] [[p,q],[p,r],[no q,s]]
==> [[q],[q,r]],
"esInconsistentePorResolucionPositiva ej 1" ~:
esInconsistentePorResoluciónPositiva [[p],[no p,q],[no q]]
==> True,
"esInconsistentePorResolucionPositiva ej 2" ~:
esInconsistentePorResoluciónPositiva [[p],[no p,q]]
==> False,
"esInconsistentePorResolucionPositiva ej 3" ~:
esInconsistentePorResoluciónPositiva [[p,q],[p,r],[no q,no r],[no p]]
==> True,
"esValidaPorResolucionPositiva ej 1" ~:
esVálidaPorResoluciónPositiva (p --> p)
==> True,
"esValidaPorResolucionPositiva ej 2" ~:
esVálidaPorResoluciónPositiva ((p --> q) \/ (q --> p))
==> True,
"esValidaPorResolucionPositiva ej 3" ~:
esVálidaPorResoluciónPositiva (p --> q)
==> False,
"esConsecuenciaPorResolucionPositiva ej 1" ~:
esConsecuenciaPorResoluciónPositiva [p --> q, q --> r] (p --> r)
==> True,
"esConsecuenciaPorResolucionPositiva ej 2" ~:
esConsecuenciaPorResoluciónPositiva [p --> q, q --> r] (p <--> r)
==> False,
"negativo ej 1" ~:
negativo p
==> False,
"negativo ej 2" ~:
negativo (no p)
```

```
==> True,
"negativa ej 1" ~:
negativa [no p,no q]
==> True,
"negativa ej 2" ~:
negativa [p, no q]
==> False,
"resNegClausulaConjunto ej 1" ~:
resNegCláusulaConjunto [no p,q] [[p,q],[p,r],[no q,s]]
==> [],
"resNegClausulaConjunto ej 2" ~:
resNegCláusulaConjunto [no p,q] [[p,q],[p,r],[no q,no s]]
==> [[no p,no s]],
"esInconsistentePorResolucionNegativa ej 1" ~:
esInconsistentePorResoluciónNegativa [[p],[no p,q],[no q]]
==> True,
"esInconsistentePorResolucionNegativa ej 2" ~:
esInconsistentePorResoluciónNegativa [[p],[no p,q]]
==> False,
"esInconsistentePorResolucionNegativa ej 3" ~:
esInconsistentePorResoluciónNegativa [[p,q],[p,r],[no q,no r],[no p]]
==> True,
"esValidaPorResolucionNegativa ej 1" ~:
esVálidaPorResoluciónNegativa (p --> p)
==> True,
"esValidaPorResolucionNegativa ej 2" ~:
esVálidaPorResoluciónNegativa ((p --> q) \/ (q --> p))
==> True,
"esValidaPorResolucionNegativa ej 3" ~:
esVálidaPorResoluciónNegativa (p --> q)
==> False,
"esConsecuenciaPorResolucionNegativa ej 1" ~:
esConsecuenciaPorResoluciónNegativa [p --> q, q --> r] (p --> r)
==> True,
"unitaria ej 1" ~:
unitaria [p]
==> True,
"unitaria ej 2" ~:
unitaria [no p]
==> True,
```

```
"unitaria ej 3" ~:
unitaria [p, q]
==> False,
"resUniClausulaConjunto ej 1" ~:
resUniCláusulaConjunto [no p,q] [[p,q],[p,r],[no q,s]]
==> [],
"resUniClausulaConjunto ej 2" ~:
resUniCláusulaConjunto [no p,q] [[p,q],[p,r],[no q]]
==> [[no p]],
"esInconsistentePorResolucionUnidad ej 1" ~:
esInconsistentePorResoluciónUnidad [[p],[no p,q],[no q]]
==> True,
"esInconsistentePorResolucionUnidad ej 2" ~:
esInconsistentePorResoluciónUnidad [[p],[no p,q]]
==> False,
"esInconsistentePorResolucionUnidad ej 3" ~:
esInconsistentePorResoluciónUnidad [[p,q],[p,r],[no q,no r],[no p]]
==> True,
"esValidaPorResolucionUnidad ej 1" ~:
esVálidaPorResoluciónUnidad (p --> p)
==> True,
"esValidaPorResolucionUnidad ej 2" ~:
esVálidaPorResoluciónUnidad ((p --> q) \/ (q --> p))
==> True,
"esValidaPorResolucionUnidad ej 3" ~:
esVálidaPorResoluciónUnidad (p --> q)
==> False,
"esConsecuenciaPorResolucionUnidad ej 1" ~:
esConsecuenciaPorResoluciónUnidad [p --> q, q --> r] (p --> r)
==> True,
"resEntClausulaConjunto ej 1" ~:
resEntCláusulaConjunto [] [no p,q] [[p,q],[p,r],[no q,s]]
==> [],
"resEntClausulaConjunto ej 2" ~:
resEntCláusulaConjunto [[p,r]] [no p,q] [[p,q],[p,r],[no q,s]]
==> [[q,r]],
"esInconsistentePorResolucionEntrada ej 1" ~:
esInconsistentePorResoluciónEntrada [] [[p],[no p,q],[no q]]
==> False,
"esInconsistentePorResolucionEntrada ej 2" ~:
```

```
     esInconsistentePorResoluciónEntrada [[p],[no p,q],[no q]]
                                         [[p],[no p,q],[no q]]

     ==> True,
     "esInconsistentePorResolucionEntrada ej 3" ~:
     esInconsistentePorResoluciónEntrada [[p],[no p,q]] [[p],[no p,q]]
     ==> False,
     "esValidaPorResolucionEntrada ej 1" ~:
     esVálidaPorResoluciónEntrada (p --> p)
     ==> True,
     "esValidaPorResolucionEntrada ej 2" ~:
     esVálidaPorResoluciónEntrada ((p --> q) \/ (q --> p))
     ==> True,
     "esValidaPorResolucionEntrada ej 3" ~:
     esVálidaPorResoluciónEntrada (p --> q)
     ==> False,
     "esConsecuenciaPorResolucionEntrada ej 1" ~:
     esConsecuenciaPorResoluciónEntrada [p --> q, q --> r] (p --> r)
     ==> True,
     "esInconsistentePorResolucionLineal ej 1" ~:
     esInconsistentePorResoluciónLineal [[no p,q],[p,no q],[no p,no q]]
                                        [p,q]
     ==> True
    ]

verificaRefinamientosResolucion :: IO Counts
verificaRefinamientosResolucion =
    runTestTT (test ejemplosRefinamientosResolucion)

-- RefinamientosResolucion> verificaRefinamientosResolucion
-- Cases: 48  Tried: 48  Errors: 0  Failures: 0
```

Capítulo 9

Programación lógica proposicional

```
module ProgramacionLogicaProposicional where

import SintaxisSemantica
import FormasNormales
import Clausulas
import Data.List
import Debug.Trace
import Test.HUnit
import Verificacion

-- ------------------------------------------------------------------
-- Secuentes                                                        --
-- ------------------------------------------------------------------

-- ------------------------------------------------------------------
-- Ejercicio 1: Definir el tipo de datos Secuente para los pares
-- formados por un átomo y una cláusula.
-- ------------------------------------------------------------------

type Secuente = (Prop,Cláusula)

-- ------------------------------------------------------------------
-- Ejercicio 2: Definir la función
--    cabeza :: Secuente -> Prop
-- tal que (cabeza s) es la cabeza de s. Por ejemplo,
--    cabeza (p,[q,r])  ==>  p
-- ------------------------------------------------------------------
```

```
cabeza :: Secuente -> Prop
cabeza = fst

-- ---------------------------------------------------------------------
-- Ejercicio 3: Definir la función
--    cuerpo :: Secuente -> Cláusula
-- tal que (cuerpo s) es el cuerpo de s. Por ejemplo,
--    cuerpo (p,[q,r])  ==>  [q,r]
-- ---------------------------------------------------------------------

cuerpo :: Secuente -> Cláusula
cuerpo = snd

-- ---------------------------------------------------------------------
-- Ejercicio 4: Definir la función
--    creaSecuente :: Prop -> Cláusula -> Secuente
-- tal que (creaSecuente x y) es el secuente de cabeza x y cuerpo y. Por
-- ejemplo,
--    creaSecuente p [q,r]  ==>  (p,[q,r])
-- ---------------------------------------------------------------------

creaSecuente :: Prop -> Cláusula -> Secuente
creaSecuente x y = (x,y)

-- ---------------------------------------------------------------------
-- Ejercicio 5: Definir el tipo de datos Programa como una lista de
-- secuentes.
-- ---------------------------------------------------------------------

type Programa = [Secuente]

-- Ejemplos de programas:
programa1 :: Programa
programa1 = [(p,[q,r]),
             (p,[s]),
             (r,[]),
             (q,[]),
             (p1,[s]),
             (p2,[s])]
```

```
programa2 = [(p,[q]),
             (q,[])]
programa3 = [(p,[r]),
             (p,[])]

-- ------------------------------------------------------------------
-- Transformación de secuentes a cláusulas                          --
-- ------------------------------------------------------------------

-- ------------------------------------------------------------------
-- Ejercicio 6: Definir la función
--    secuente2Cláusula :: Secuente -> Cláusula
-- tal que (secuente2Cláusula s) es la cláusula correspondiente al
-- secuente s. Por ejemplo,
--    secuente2Cláusula (p,[q,no r])  ==>  [p,no q,r]
--    secuente2Cláusula (p,[q,r])     ==>  [p,no q,no r]
--    secuente2Cláusula (p,[])        ==>  [p]
-- ------------------------------------------------------------------

secuente2Cláusula :: Secuente -> Cláusula
secuente2Cláusula (cabeza,cuerpo) =
    cabeza : [complementario l | l <- cuerpo]

-- ------------------------------------------------------------------
-- Semántica declarativa                                            --
-- ------------------------------------------------------------------

-- ------------------------------------------------------------------
-- Ejercicio 7: Definir la función
--    baseDeHerbrand :: Programa -> Interpretación
-- tal que (baseDeHerbrand p) es la base de Herbrand del programa p. Por
-- ejemplo,
--    baseDeHerbrand programa1  ==>  [p,q,r,s,p1,p2]
-- ------------------------------------------------------------------

baseDeHerbrand :: Programa -> Interpretación
baseDeHerbrand p =
    símbolosProposicionalesConjuntoCláusula [secuente2Cláusula s | s <- p]

-- ------------------------------------------------------------------
```

```
-- Ejercicio 8: Definir la función
--    modelosDeHerbrand :: Programa -> [Interpretación]
-- tal que (modelosDeHerbrand p) es el conjunto de los modelos de
-- Herbrand del programa p. Por ejemplo,
--    modelosDeHerbrand programa1
--    ==> [[p,q,r,s,p1,p2],[p,q,r,p1,p2],[p,q,r,p1],[p,q,r,p2],[p,q,r]]
-- ---------------------------------------------------------------------

modelosDeHerbrand :: Programa -> [Interpretación]
modelosDeHerbrand p =
    modelosConjuntoCláusulas [secuente2Cláusula s | s <- p]

-- ---------------------------------------------------------------------
-- Ejercicio 9: Definir la función
--    menorModeloDeHerbrand :: Programa -> Interpretación
-- tal que (menorModeloDeHerbrand p) es el menor modelo de Herbrand del
-- programa p. Por ejemplo,
--    menorModeloDeHerbrand programa1  ==>  [p,q,r]
-- ---------------------------------------------------------------------

menorModeloDeHerbrand :: Programa -> Interpretación
menorModeloDeHerbrand p =
    intersecciónGeneral (modelosDeHerbrand p)
    where intersecciónGeneral [x]      = x
          intersecciónGeneral (x:y:zs) =
              intersecciónGeneral ((x `intersect` y):zs)

-- ---------------------------------------------------------------------
-- Semántica de punto fijo                                             --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 10: Definir la función
--    subconjunto :: Eq a => [a] -> [a] -> Bool
-- tal que (subconjunto x y) se verifica si x es subconjunto de y. Por
-- ejemplo,
--    subconjunto [1,3] [3,2,1]    ==>  True
--    subconjunto [1,3,5] [3,2,1]  ==> False
-- ---------------------------------------------------------------------
```

```
subconjunto :: Eq a => [a] -> [a] -> Bool
subconjunto xs ys =
    and [elem x ys | x <- xs]

-- ---------------------------------------------------------------------
-- Ejercicio 11: Definir la función
--    consecuenciasInmediatas :: Programa -> Interpretación ->
--                               Interpretación
-- tal que (consecuenciasInmediatas p i) es el conjunto de átomos que
-- son consecuencia inmediata del programa p y la interpretación i. Por
-- ejemplo,
--    consecuenciasInmediatas programa1 []       ==>  [r,q]
--    consecuenciasInmediatas programa1 [r,q]    ==>  [p,r,q]
--    consecuenciasInmediatas programa1 [p,r,q]  ==>  [p,r,q]
-- ---------------------------------------------------------------------

consecuenciasInmediatas :: Programa -> Interpretación -> Interpretación
consecuenciasInmediatas p i =
    [cabeza s | s <- p, subconjunto (cuerpo s) i]

-- ---------------------------------------------------------------------
-- Ejercicio 12: Definir la función
--    menorPuntoFijo :: Programa -> Interpretación
-- tal que (menorPuntoFijo p) es el menor punto fijo del programa p. Por
-- ejemplo,
--    menorPuntoFijo programa1  ==>  [p,r,q]
-- ---------------------------------------------------------------------

menorPuntoFijo :: Programa -> Interpretación
menorPuntoFijo p =
    menorPuntoFijo' p []

menorPuntoFijo' :: Programa -> Interpretación -> Interpretación
menorPuntoFijo' p i
    | consecuencias == i = i
    | otherwise          = menorPuntoFijo' p consecuencias
    where consecuencias = consecuenciasInmediatas p i

-- ---------------------------------------------------------------------
-- Resolución SLD                                                    --
```

```
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 13: Definir el tipo Objetivo para representar las listas de
-- literales.
-- ---------------------------------------------------------------------

type Objetivo = [Literal]

-- ---------------------------------------------------------------------
-- Ejercicio 14: Definir la función
--    sldResolvente :: Objetivo -> Secuente -> Objetivo
-- tal que (sldResolvente o s) es la resolvente SLD del objetivo o con
-- el secuente s. Por ejemplo,
--    sldResolvente [q,r] (q,[p,s])  ==>  [p,s,r]
-- ---------------------------------------------------------------------

sldResolvente :: Objetivo -> Secuente -> Objetivo
-- sldResolvente o s
--     | trace (" ==> " ++ show o ++ "\t" ++ show s) False = undefined
sldResolvente o s =
    (cuerpo s) ++ (tail o)

-- ---------------------------------------------------------------------
-- Ejercicio 15: Definir la función
--    pruebaPorSLD_Resolución :: Programa -> Objetivo -> Bool
-- tal que (pruebaPorSLD_Resolución p o) que se verifique si el objetivo
-- o es demostrable por resolución SLD a partir del programa p. Además,
-- si la primera ecuación de sldResolvente no está comentada, escribe la
-- prueba. Por ejemplo,
--    PLP> pruebaPorSLD_Resolución programa1 [p]
--     ==> [p]    (p,[q,r])
--     ==> [q,r]  (q,[])
--     ==> [r]          (r,[])
--    True
--    PLP> pruebaPorSLD_Resolución ((p,[s]):programa1) [p]
--     ==> [p]         (p,[s])
--     ==> [p]         (p,[q,r])
--     ==> [q,r] (q,[])
--     ==> [r]         (r,[])
```

```
--    True
--    PLP> pruebaPorSLD_Resolución programa2 [p]
--     ==> [p]        (p,[q])
--     ==> [q]        (q,[])
--    True
--    PLP> pruebaPorSLD_Resolución programa2 [r]
--    False
--    PLP> pruebaPorSLD_Resolución programa3 [p]
--     ==> [p]        (p,[r])
--     ==> [p]        (p,[])
--    True
-- ----------------------------------------------------------------------

pruebaPorSLD_Resolución :: Programa -> Objetivo -> Bool
pruebaPorSLD_Resolución p o
    | null o    = True
    | otherwise = or [pruebaPorSLD_Resolución p (sldResolvente o s)
                      | s <- p
                      , cabeza s == head o]
-- ----------------------------------------------------------------------
-- Verificación                                                         --
-- ----------------------------------------------------------------------

ejemplosProgramacionLogicaProposicional :: [Test]
ejemplosProgramacionLogicaProposicional =
    ["cabeza ej 1" ~:
     cabeza (p,[q,r])
     ==> p,
     "cuerpo ej 1" ~:
     cuerpo (p,[q,r])
     ==> [q,r],
     "creaSecuente ej 1" ~:
     creaSecuente p [q,r]
     ==> (p,[q,r]),
     "secuente2Cláusula ej 1" ~:
     secuente2Cláusula (p,[q,no r])
     ==> [p,no q,r],
     "secuente2Cláusula ej 2" ~:
     secuente2Cláusula (p,[q,r])
     ==> [p,no q,no r],
```

```
"secuente2Cláusula ej 3" ~:
secuente2Cláusula (p,[])
==> [p],
"baseDeHerbrand ej 1" ~:
baseDeHerbrand programa1
==> [p,q,r,s,p1,p2],
"modelosDeHerbrand ej 1" ~:
modelosDeHerbrand programa1
==> [[p,q,r,s,p1,p2],[p,q,r,p1,p2],[p,q,r,p1],[p,q,r,p2],[p,q,r]],
"menorModeloDeHerbrand ej 1" ~:
menorModeloDeHerbrand programa1
==> [p,q,r],
"subconjunto ej 1" ~:
subconjunto [1,3] [3,2,1]
==> True,
"subconjunto ej 2" ~:
subconjunto [1,3,5] [3,2,1]
==> False,
"consecuenciasInmediatas ej 1" ~:
consecuenciasInmediatas programa1 []
==> [r,q],
"consecuenciasInmediatas ej 2" ~:
consecuenciasInmediatas programa1 [r,q]
==> [p,r,q],
"consecuenciasInmediatas ej 3" ~:
consecuenciasInmediatas programa1 [p,r,q]
==> [p,r,q],
"menorPuntoFijo ej 1" ~:
menorPuntoFijo programa1
==> [p,r,q],
"sldResolvente ej 1" ~:
sldResolvente [q,r] (q,[p,s])
==> [p,s,r],
"pruebaPorSLD_Resolución ej 1" ~:
pruebaPorSLD_Resolución programa1 [p]
==> True,
"pruebaPorSLD_Resolución ej 2" ~:
pruebaPorSLD_Resolución ((p,[s]):programa1) [p]
==> True,
"pruebaPorSLD_Resolución ej 3" ~:
```

```
     pruebaPorSLD_Resolución programa2 [p]
     ==> True,
     "pruebaPorSLD_Resolución ej 4" ~:
     pruebaPorSLD_Resolución programa2 [r]
     ==> False,
     "pruebaPorSLD_Resolución ej 5" ~:
     pruebaPorSLD_Resolución programa3 [p]
     ==> True
    ]

verificaProgramacionLogicaProposicional :: IO Counts
verificaProgramacionLogicaProposicional =
    runTestTT (test ejemplosProgramacionLogicaProposicional)

-- ProgramacionLogicaProposicional> verificaProgramacionLogicaProposicional
-- Cases: 21  Tried: 21  Errors: 0  Failures: 0
```

Capítulo 10

Unificación de términos de primer orden

```
module Unificacion where

import Data.List
import Test.HUnit
import Verificacion

-- ----------------------------------------------------------------------
-- Variables, términos y átomos                                         --
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Ejercicio 1: Definir el tipo de datos Nombre para representar las
-- cadenas.
-- ----------------------------------------------------------------------

type Nombre   = String

-- ----------------------------------------------------------------------
-- Ejercicio 2: Definir el tipo de datos Términos para representar los
-- términos como una variable o un símbolo de función seguido de una
-- lista de términos.
-- ----------------------------------------------------------------------

data Término = V Nombre
             | T Nombre [Término]
             deriving Eq
```

```
instance Show Término where
    show (V nombre)    = nombre
    show (T nombre []) = nombre
    show (T nombre ts) = nombre ++ concat [show ts]

-- ---------------------------------------------------------------------
-- Ejercicio 3: Definir el tipo de datos Atomo para representar los
-- átomos como un símbolo de predicado seguido de una lista de términos.
-- ---------------------------------------------------------------------

data Atomo = A Nombre [Término]
             deriving Eq

instance Show Atomo where
    show (A nombre []) = nombre
    show (A nombre ts) = nombre ++ concat [show ts]

-- ---------------------------------------------------------------------
-- Ejercicio 4: Definir el tipo de datos Variable.
-- ---------------------------------------------------------------------

type Variable = Término

-- Ejemplos de variables:
x = V "x"
y = V "y"
z = V "z"
u = V "u"
w = V "w"

-- Ejemplos de constantes:
a = T "a" []
b = T "b" []
c = T "c" []

-- Ejemplos de símbolos de función:
f = T "f"
g = T "g"
h = T "h"
```

```
-- Ejemplo de símbolos de predicado:
p = A "p"

-- ---------------------------------------------------------------------
-- Ejercicio 5: Definir la función
--    esVariable :: Término -> Bool
-- tal que (esVariable t) que se verifica si t es una variable. Por
-- ejemplo,
--    esVariable x  ==>  True
--    esVariable a  ==>  False
-- ---------------------------------------------------------------------

esVariable :: Término -> Bool
esVariable (V _) = True
esVariable _     = False

-- ---------------------------------------------------------------------
-- Ejercicio 6: Definir la función
--    variables :: Término -> [Variable]
-- tal que (variables t) es la lista de variables de t. Por ejemplo,
--    variables (g[f[x,y],z])  ==>  [x,y,z]
-- ---------------------------------------------------------------------

variables :: Término -> [Variable]
variables (V v)    = [V v]
variables (T n ts) = variablesEnLista ts

-- ---------------------------------------------------------------------
-- Ejercicio 7: Definir la función
--    variablesEnLista :: [Término] -> [Variable]
-- tal que (variablesEnLista ts) es la lista de variables de la lista de
-- términos ts. Por ejemplo,
--    variablesEnLista [f[x,y], g[f[x,y],z]]  ==>  [x,y,z]
-- ---------------------------------------------------------------------

variablesEnLista :: [Término] -> [Variable]
variablesEnLista = nub . concat . map variables

-- ---------------------------------------------------------------------
-- Sustituciones                                                     --
```

```
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 8: Definir el tipo de datos Sustitución para representar
-- las listas de pares formados por una variable y un término.
-- ---------------------------------------------------------------------

type Sustitución = [(Variable,Término)]

-- Ejemplos de sustitución:
sigma1 :: Sustitución
sigma1 = [(x,a),(z,f[x,y])]
sigma2 = [(x,z),(y,u)]
sigma3 = [(z,x),(x,b),(u,c)]
sigma4 = [(u,f[x]),(y,a)]
sigma5 = [(x,h[z]),(y,g[b])]

-- ---------------------------------------------------------------------
-- Ejercicio 9: Definir la función
--    epsilon :: Sustitución
-- tal que (epsilon s) es la sustitución identidad.
-- ---------------------------------------------------------------------

epsilon :: Sustitución
epsilon = []

-- ---------------------------------------------------------------------
-- Ejercicio 10: Definir la función
--    dominio :: Sustitución -> [Variable]
-- tal que (dominio s) es el domiinio de la sustitución s. Por ejemplo,
--    dominio sigma1  ==>  [x,z]
-- ---------------------------------------------------------------------

dominio :: Sustitución -> [Variable]
dominio = map fst

-- ---------------------------------------------------------------------
-- Ejercicio 11: Definir la función
--    aplicaVar :: Sustitución -> Variable -> Término
-- tal que (aplicaVar s x) es el resultado de aplicar la sustitución s a
```

```
-- la variable x. Por ejemplo,
--    sigma1              ==>  [(x,a),(z,f[x,y])]
--    aplicaVar sigma1 x  ==>  a
--    aplicaVar sigma1 y  ==>  y
--    aplicaVar sigma1 z  ==>  f[x,y]
-- ---------------------------------------------------------------------

aplicaVar :: Sustitución -> Variable -> Término
aplicaVar []         (V y) = V y
aplicaVar ((x,t):xs) (V y)
    | x == (V y) = t
    | otherwise  = aplicaVar xs (V y)

-- ---------------------------------------------------------------------
-- Ejercicio 12: Definir la función
--    aplicaT :: Sustitución -> Término -> Término
-- tal que (aplicaT s t) es el resultado de aplicar la sustitución s al
-- término t. Por ejemplo,
--    sigma1                         ==>  [(x,a),(z,f[x,y])]
--    aplicaT sigma1 (g[f[x,y],z])  ==>  g[f[a,y],f[x,y]]
-- ---------------------------------------------------------------------

aplicaT :: Sustitución -> Término -> Término
aplicaT s (V x)    = aplicaVar s (V x)
aplicaT s (T n ts) = T n [aplicaT s t | t <- ts]

-- ---------------------------------------------------------------------
-- Ejercicio 13: Definir la función
--    reduce :: Sustitución -> Sustitución
-- tal que (reduce s) es la sustitución obtenida eliminando los pares de
-- la sustitución s cuyos elementos son iguales. Por ejemplo,
--    reduce [(x,a),(y,y),(z,f[x,y])]  ==>  [(x,a),(z,f[x,y])]
-- ---------------------------------------------------------------------

reduce :: Sustitución -> Sustitución
reduce s =
    [(x,t) | (x,t) <- s, x /= t]

-- ---------------------------------------------------------------------
-- Ejercicio 14: Definir la función
```

```
--    composición :: Sustitución -> Sustitución -> Sustitución
-- tal que (composición s1 s2) es la composición de las sustituciones s1
-- y s2. Por ejemplo,
--    (sigma2,sigma3)           => ([(x,z),(y,u)],[(z,x),(x,b),(u,c)])
--    composición sigma2 sigma3 => [(y,c),(z,x),(u,c)]
--    (sigma4, sigma5)          => ([(u,f[x]),(y,a)],[(x,h[z]),(y,g[b])])
--    composición sigma4 sigma5 => [(u,f[h[z]]),(y,a),(x,h[z])]
-- ---------------------------------------------------------------------

composición :: Sustitución -> Sustitución -> Sustitución
composición xs ys =
    (reduce [ (y,(aplicaT ys t)) | (y,t) <-  xs ])
    ++
    [ (x,t) | (x,t) <- ys, x `notElem` (dominio xs) ]

-- ---------------------------------------------------------------------
-- Unificación                                                         --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 15: Definir la función
--    unifica :: Término -> Término -> [Sustitución]
-- tal que (unifica t1 t2) es la lista formada por un unificador de
-- máxima generalidad de los términos t1 y t2, si son unificables y es
-- la lista vacía en caso contrario, Por ejemplo,
--    unifica a a                  ==>  [[]]
--    unifica x a                  ==>  [[(x,a)]]
--    unifica x (f[y])             ==>  [[(x,f[y])]]
--    unifica x (f[x])             ==>  []
--    unifica (f[y]) x             ==>  [[(x,f[y])]]
--    unifica (f[x]) x             ==>  []
--    unifica a b                  ==>  []
--    unifica (f[x,b]) (f[a,y])    ==>  [[(y,b),(x,a)]]
--    unifica (f[x,x]) (f[a,b])    ==>  []
--    unifica (f[x,g[y]]) (f[y,x]) ==>  []
-- ---------------------------------------------------------------------

unifica :: Término -> Término -> [Sustitución]
unifica (V x) (V y)
    | x==y      = [epsilon]
```

```
    | otherwise = [[(V x,V y)]]
unifica (V x) t2 =
    [ [(V x,t2)] | (V x) `notElem` variables t2 ]
unifica t1 (V y) =
    [ [(V y,t1)] | (V y) `notElem` variables t1 ]
unifica (T f ts) (T g rs) =
    [ u | f==g, u <- unificaListas ts rs ]

-- ---------------------------------------------------------------------
-- Ejercicio 16: Definir la función
--    unificaListas :: [Término] -> [Término] -> [Sustitución]
-- tal que (unificaListas (unificaListas ts1 ts2) es la lista formada
-- por un unificador de máxima generalidad de las ecuaciones
-- correspondientes a las listas de términos ts1 y ts2, si son
-- unificables y es la lista vacía en caso contrario. Por ejemplo,
--    unificaListas [x,f[x],y] [a,y,z] => [[(z,f[a]),(y,f[a]),(x,a)]]
--    unificaListas [x,f[x]] [y,y]     => []
-- ---------------------------------------------------------------------

unificaListas :: [Término] -> [Término] -> [Sustitución]
unificaListas []     []     = [epsilon]
unificaListas []     (r:rs) = []
unificaListas (t:ts) []     = []
unificaListas (t:ts) (r:rs) =
   [ composición sigma2 sigma1
     | sigma1 <- unifica t r,
       sigma2 <- unificaListas [aplicaT sigma1 t | t <- ts]
                               [aplicaT sigma1 r | r <- rs] ]

-- ---------------------------------------------------------------------
-- Ejercicio 17: Definir la función
--    unificaA :: Atomo -> Atomo -> [Sustitución]
-- tal que (unificaA a1 a2) es la lista formada por un unificador de
-- máxima generalidad de los átomos a1 y a2, si son unificables y es la
-- lista vacía en caso contrario, Por ejemplo,
--    unificaA (p[w,a,h[w]]) (p[f[x,y],x,z])
--    ==> [[(z,h[f[a,y]]),(x,a),(w,f[x,y])]]
--    unificaA (p[w,a,h[w]]) (p[f[x,y],x,y])
--    ==> []
-- ---------------------------------------------------------------------
```

```
unificaA :: Atomo -> Atomo -> [Sustitución]
unificaA (A n1 ts1) (A n2 ts2)
    | n1==n2    = unificaListas ts1 ts2
    | otherwise = []

-- ------------------------------------------------------------------
-- Verificación                                                     --
-- ------------------------------------------------------------------

ejemplosUnificacion :: [Test]
ejemplosUnificacion =
    ["esVariable ej 1" ~:
     esVariable x
     ==> True,
     "esVariable ej 1" ~:
     esVariable a
     ==> False,
     "variables ej 1" ~:
     variables (g[f[x,y],z])
     ==> [x,y,z],
     "variablesEnLista ej 1" ~:
     variablesEnLista [f[x,y], g[f[x,y],z]]
     ==> [x,y,z],
     "dominio ej 1" ~:
     dominio sigma1
     ==> [x,z],
     "aplicaVar ej 1" ~:
     aplicaVar sigma1 x
     ==> a,
     "aplicaVar ej 1" ~:
     aplicaVar sigma1 y
     ==> y,
     "aplicaVar ej 1" ~:
     aplicaVar sigma1 z
     ==> f[x,y],
     "aplicaT ej 1" ~:
     aplicaT sigma1 (g[f[x,y],z])
     ==> g[f[a,y],f[x,y]],
     "reduce ej 1" ~:
```

```
reduce [(x,a),(y,y),(z,f[x,y])]
==> [(x,a),(z,f[x,y])],
"composición ej 1" ~:
composición sigma2 sigma3
==> [(y,c),(z,x),(u,c)],
"composición ej 1" ~:
composición sigma4 sigma5
==> [(u,f[h[z]]),(y,a),(x,h[z])],
"unifica ej 1" ~:
unifica a a
==> [[]],
"unifica ej 1" ~:
unifica x a
==> [[(x,a)]],
"unifica ej 1" ~:
unifica x (f[y])
==> [[(x,f[y])]],
"unifica ej 1" ~:
unifica x (f[x])
==> [],
"unifica ej 1" ~:
unifica (f[y]) x
==> [[(x,f[y])]],
"unifica ej 1" ~:
unifica (f[x]) x
==> [],
"unifica ej 1" ~:
unifica a b
==> [],
"unifica ej 1" ~:
unifica (f[x,b]) (f[a,y])
==> [[(y,b),(x,a)]],
"unifica ej 1" ~:
unifica (f[x,x]) (f[a,b])
==> [],
"unifica ej 1" ~:
unifica (f[x,g[y]]) (f[y,x])
==> [],
"unificaListas ej 1" ~:
unificaListas [x,f[x],y] [a,y,z]
```

```
     ==> [[(z,f[a]),(y,f[a]),(x,a)]],
     "unificaListas ej 1" ~:
     unificaListas [x,f[x]] [y,y]
     ==> [],
     "unificaA ej 1" ~:
     unificaA (p[w,a,h[w]]) (p[f[x,y],x,z])
     ==> [[(z,h[f[a,y]]),(x,a),(w,f[x,y])]],
     "unificaA ej 1" ~:
     unificaA (p[w,a,h[w]]) (p[f[x,y],x,y])
     ==> []
    ]

verificaUnificacion :: IO Counts
verificaUnificacion =
    runTestTT (test ejemplosUnificacion)

-- Unificacion> verificaUnificacion
-- Cases: 26  Tried: 26  Errors: 0  Failures: 0
```

Capítulo 11

Implementación de Prolog

```
module Prolog where

import Unificacion
import Data.List
import Debug.Trace
import Test.HUnit
import Verificacion

-- ---------------------------------------------------------------------
-- Cláusulas, objetivos y programas                                    --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 1: Definir el tipo de datos Cláusulas para representar las
-- cláusulas como pares formados por un átomo y una lista de átomos.
-- ---------------------------------------------------------------------

type Cláusula = (Atomo,[Atomo])

-- ---------------------------------------------------------------------
-- Ejercicio 2: Definir el tipo de datos Objetivos para representar los
-- objetivos como listas de átomos.
-- ---------------------------------------------------------------------

type Objetivo = [Atomo]

-- ---------------------------------------------------------------------
```

```
-- Ejercicio 3: Definir el tipo de datos Programa para representar los
-- programas como listas de cláusulas.
-- ---------------------------------------------------------------------

type Programa = [Cláusula]

-- ---------------------------------------------------------------------
-- Ejercicio 4: Definir el tipo de datos Nivel como sinónimo de número
-- enteros.
-- ---------------------------------------------------------------------

type Nivel = Int

-- Ejemplos.
q    = A "q"
r    = A "r"
suma = A "suma"
s    = T "s"
o    = T "o" []
programa1 = [(p[x], [q[x]]),
             (q[x], [])]
programa2 = [(suma [o,x,x],       []),
             (suma [s[x],y,s[z]], [suma[x,y,z]])]

-- ---------------------------------------------------------------------
-- Ejercicio 5: Definir la función
--    cabeza :: Cláusula -> Atomo
-- tal que (cabeza c) es la cabeza de la cláusula c. Por ejemplo,
--    cabeza (p[x], [q[x], r[x]])  ==>  p[x]
-- ---------------------------------------------------------------------

cabeza :: Cláusula -> Atomo
cabeza = fst

-- ---------------------------------------------------------------------
-- Ejercicio 6: Definir la función
--    cuerpo :: Cláusula -> [Atomo]
-- tal que (cuerpo c) es el cuerpo de la cláusula c. Por ejemplo,
--    cuerpo (p[x], [q[x], r[x]])  ==>  [q[x],r[x]]
-- ---------------------------------------------------------------------
```

```
cuerpo :: Cláusula -> [Atomo]
cuerpo = snd

-- ---------------------------------------------------------------------
-- Cálculo de respuestas                                               --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 7: Definir la función
--    renombraT :: Término -> Nivel -> Término
-- tal que (renombraT t n) es es el resultado de añadirle a las variables
-- del término t como subíndice el número n. Por ejemplo,
--    renombraT x 3         ==>  x_3
--    renombraT (f[x,y]) 3  ==>  f[x_3,y_3]
-- ---------------------------------------------------------------------

renombraT :: Término -> Nivel -> Término
renombraT (V x) n     = V (x ++ "_" ++ show n)
renombraT (T f ts) n  = T f [aplicaT s t | t <- ts]
                        where s = [(x,renombraT x n)
                                   | x <- variablesEnLista ts]

-- ---------------------------------------------------------------------
-- Ejercicio 8: Definir la función
--    renombraA :: Atomo -> Nivel -> Atomo
-- tal que (renombraA a n) es es el resultado de añadirle a las
-- variables del átomo a como subíndice el número n. Por ejemplo,
--    renombraA (p[x,y]) 3  ==>  p[x_3,y_3]
-- ---------------------------------------------------------------------

renombraA :: Atomo -> Nivel -> Atomo
renombraA (A p ts) n =
    A p [renombraT t n | t <- ts]

-- ---------------------------------------------------------------------
-- Ejercicio 9: Definir la función
--    renombra :: [Atomo] -> Nivel -> [Atomo]
-- tal que (renombra as n) es es el resultado de añadirle a las
-- variables de la lista de átomos as como subíndice el número n. Por
```

```
-- ejemplo,
--    renombra [p[x,y], q[f[x],z]] 3  ==>  [p[x_3,y_3],q[f[x_3],z_3]]
-- ----------------------------------------------------------------------

renombra :: [Atomo] -> Nivel -> [Atomo]
renombra as n =
    [renombraA a n | a <- as]

-- ----------------------------------------------------------------------
-- Ejercicio 10: Definir la función
--    resolvente :: Objetivo -> Cláusula -> Nivel -> Objetivo
-- tal que (resolvente o c n) es la resolvente del objetivo c y la
-- cláusula c en el nivel n. Por ejemplo,
--    resolvente [p[x],q[x]] (p[x],[r[x]]) 3  ==>  [r[x_3],q[x]]
-- ----------------------------------------------------------------------

resolvente :: Objetivo -> Cláusula -> Nivel -> Objetivo
-- resolvente o c n
--     | trace ("\n ==> " ++ show o ++ "\t" ++ show c) False = undefined
resolvente o c n =
    (renombra (cuerpo c) n) ++ (tail o)

-- ----------------------------------------------------------------------
-- Ejercicio 11: Definir la función
--    aplicaA :: Sustitución -> Atomo -> Atomo
-- tal que (aplicaA s a) es el resultado de aplicar la sustitución s al
-- átomo a. Por ejemplo,
--    aplicaA [(x,z),(z,f[x,y])] (p[g[x],y])  ==>  p[g[z],y]
-- ----------------------------------------------------------------------

aplicaA :: Sustitución -> Atomo -> Atomo
aplicaA s (A n ts) =
    A n [aplicaT s t | t <- ts]

-- ----------------------------------------------------------------------
-- Ejercicio 12: Definir la función
--    aplica :: Sustitución -> Objetivo -> Objetivo
-- tal que (aplica s o) es el resultado de aplicar la sustitución s al
-- objetivo o. Por ejemplo,
--    aplica [(x,z),(z,f[x,y])] [p[g[x],y], p[z]]
```

```
--    ==> [p[g[z],y],p[f[x,y]]]
-- ---------------------------------------------------------------------

aplica :: Sustitución -> Objetivo -> Objetivo
aplica s o =
    [aplicaA s a | a <- o]

-- ---------------------------------------------------------------------
-- Ejercicio 13: Definir la función
--    respuestas :: Programa -> Objetivo -> [Sustitución]
-- tal que (respuestas p o) es la listas de respuestas al objetivo o
-- mediante el programa p. Por ejemplo,
--    respuestas programa1 [p[x]]  ==>  [[(x_0,x),(x_1,x)]]
-- ---------------------------------------------------------------------

respuestas :: Programa -> Objetivo -> [Sustitución]
respuestas p o =
    respuestas' p o 0 []

respuestas' :: Programa -> Objetivo -> Nivel -> Sustitución ->
               [Sustitución]
respuestas' p o n s
    | null o    = [s]
    | otherwise = concat [respuestas' p
                                      (aplica u (resolvente o c n))
                                      (n+1)
                                      (composición s u)
                          | c <- p
                          , let us = unificaA (renombraA (cabeza c) n)
                                              (head o)
                          , not (null us)
                          , let u = head us]

-- ---------------------------------------------------------------------
-- Respuestas reducidas                                                --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 14: Definir la función
--    unión General :: Eq a => [[a]] -> [a]
```

```
-- tal que (uniónGeneral x) es la unión de los conjuntos de la lista de
-- conjuntos x. Por ejemplo,
--    uniónGeneral []                 ==>  []
--    uniónGeneral [[1]]              ==>  [1]
--    uniónGeneral [[1],[1,2],[2,3]]  ==>  [1,2,3]
-- ------------------------------------------------------------------

uniónGeneral :: Eq a => [[a]] -> [a]
uniónGeneral []     = []
uniónGeneral (x:xs) = x `union` uniónGeneral xs

-- ------------------------------------------------------------------
-- Ejercicio 15: Definir la función
--    variablesÁtomo :: Atomo -> [Variable]
-- tal que (variablesÁtomo a) es la lista de variables del átomo a. Por
-- ejemplo,
--    variablesÁtomo (p[x,f[y,x]])  ==>  [x,y]
-- ------------------------------------------------------------------

variablesÁtomo :: Atomo -> [Variable]
variablesÁtomo (A _ ts) =
    variablesEnLista ts

-- ------------------------------------------------------------------
-- Ejercicio 16: Definir la función
--    variablesObjetivo :: Objetivo -> [Variable]
-- tal que (variablesObjetivo o) es la lista de variables del objetivo
-- o. Por ejemplo,
--    variablesObjetivo [p[z,f[x,y]], q[y,a,z]]  ==>  [z,x,y]
-- ------------------------------------------------------------------

variablesObjetivo :: Objetivo -> [Variable]
variablesObjetivo o =
    uniónGeneral [variablesÁtomo a | a <- o]

-- ------------------------------------------------------------------
-- Ejercicio 17: Definir la función
--    valor :: Término -> Sustitución -> Término
-- tal que (valor t s) es el valor del término t mediante la sustitución
-- s. Por ejemplo,
```

```
--    valor (f[x]) [(x,g[y]),(y,a)]  ==>  f[g[a]]
-- ---------------------------------------------------------------------

valor :: Término -> Sustitución -> Término
valor (V x) s
      | elem (V x) (dominio s) = valor (head [t | (y,t) <- s, y==V x]) s
      | otherwise              = V x
valor (T f ts) s               = T f [valor t s | t <- ts]

-- ---------------------------------------------------------------------
-- Ejercicio 18: Definir la función
--    calculaRespuesta :: [Variable] -> Sustitución -> Sustitución
-- tal que (calculaRespuesta xs s) es la lista de valores de las
-- variables de xs según la sustitución s. Por ejemplo,
--    calculaRespuesta [x,z] [(x,g[y]),(y,a),(z,f[x])]
--    ==> [(x,g[a]),(z,f[g[a]])]
-- ---------------------------------------------------------------------

calculaRespuesta :: [Variable] -> Sustitución -> Sustitución
calculaRespuesta xs s =
    [(x,valor x s) | x <- xs]

-- ---------------------------------------------------------------------
-- Ejercicio 19: Definir la función
--    respuestasReducidas :: Programa -> Objetivo -> [Sustitución]
-- tal que (respuestasReducidas p o) es la listas de respuestas
-- reducidas al objetivo o mediante el programa p. Por ejemplo,
--    respuestasReducidas programa2 [suma [x,y,s[o]]]
--    ==> [[(x,o),(y,s[o])],[(x,s[o]),(y,o)]]
--    take 3 (respuestasReducidas programa2 [suma [x,y,z]])
--    ==> [[(x,o),(y,z),(z,z)],
--         [(x,s[o]),(y,z_0),(z,s[z_0])],
--         [(x,s[s[o]]),(y,z_1),(z,s[s[z_1]])]]
-- ---------------------------------------------------------------------

respuestasReducidas :: Programa -> Objetivo -> [Sustitución]
respuestasReducidas p o =
    [calculaRespuesta var s | s <- res]
    where var = variablesObjetivo o
          res = respuestas p o
```

```
-- ---------------------------------------------------------------------
-- Escritura de respuestas                                             --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Ejercicio 20: Definir la función
--    escribeLigadura :: (Variable,Término) -> IO ()
-- tal que (escribeLigadura l) escribe la ligadura l. Por ejemplo,
--    Prolog> escribeLigadura (x,a)
--    x = a
-- ---------------------------------------------------------------------

escribeLigadura :: (Variable,Término) -> IO ()
escribeLigadura (x,t) =
    putStrLn ((show x)++" = "++(show t))

-- ---------------------------------------------------------------------
-- Ejercicio 21: Definir la función
--    escribeSustitución :: Sustitución -> IO ()
-- tal que (escribeSustitución s) escribe la sustitución s. Por ejemplo,
--    Prolog> escribeSustitución [(x,a),(y,b)]
--    x = a
--    y = b
-- ---------------------------------------------------------------------

escribeSustitución :: Sustitución -> IO ()
escribeSustitución [] =
    putStrLn ""
escribeSustitución (l:ls) =
    do escribeLigadura l
       escribeSustitución ls

-- ---------------------------------------------------------------------
-- Ejercicio 22: Definir la función
--    escribeRespuesta :: [Sustitución] -> IO ()
-- tal que (escribeRespuesta r) escribe la respuesta r. Por ejemplo,
--    Prolog> escribeRespuesta [[(x,o),(y,s[o])],[(x,s[o]),(y,o)]]
--    Respuesta 1:
--    x = o
```

```
--    y = s[o]
--
--    Respuesta 2:
--    x = s[o]
--    y = o
--
--    No hay más respuestas.
--
--    Prolog> escribeRespuesta
--            (take 2 (respuestasReducidas programa2 [suma [x,y,z]]))
--    Respuesta 1:
--    x = o
--    y = z
--    z = z
--
--    Respuesta 2:
--    x = s[o]
--    y = z0
--    z = s[z0]
--
--    No hay más respuestas.
-- ---------------------------------------------------------------------

escribeRespuesta :: [Sustitución] -> IO ()
escribeRespuesta rs =
    escribeRespuesta' (zip [1..] rs)

escribeRespuesta' :: [(Int,Sustitución)] -> IO ()
escribeRespuesta' [] =
    putStrLn "No hay más respuestas."
escribeRespuesta' ((n,r):nrs) =
    do putStrLn ("Respuesta " ++ (show n) ++ ":")
       escribeSustitución r
       escribeRespuesta' nrs

-- ---------------------------------------------------------------------
-- Prolog                                                              --
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
```

```
-- Ejercicio 23: Definir la función
--    prolog :: Programa -> Objetivo -> IO ()
-- tal que (prolog p o) escribe las respuestas reducidas al objetivo o
-- mediante el programa p. Por ejemplo,
--    Prolog> prolog programa2 [suma [x,y,s[s[o]]]]
--    Respuesta 1:
--    x = o
--    y = s[s[o]]
--
--    Respuesta 2:
--    x = s[o]
--    y = s[o]
--
--    Respuesta 3:
--    x = s[s[o]]
--    y = o
--
--    No hay más respuestas.
--
--    Prolog> prolog programa2 [suma [x,s[y],o]]
--    No hay respuesta.
-- ---------------------------------------------------------------------

prolog :: Programa -> Objetivo -> IO ()
prolog p o
    | null res  = putStrLn "No hay respuesta."
    | otherwise = escribeRespuesta res
    where res = respuestasReducidas p o
-- ---------------------------------------------------------------------
-- Verificación                                                        --
-- ---------------------------------------------------------------------

ejemplosProlog :: [Test]
ejemplosProlog =
    ["cabeza ej 1" ~:
     cabeza (p[x], [q[x], r[x]])
     ==> p[x],
     "cuerpo ej 1" ~:
     cuerpo (p[x], [q[x], r[x]])
     ==> [q[x],r[x]],
```

```
"renombraT ej 1" ~:
renombraT x 3
==> V "x_3",
"renombraT ej 1" ~:
renombraT (f[x,y]) 3
==> f[V "x_3",V "y_3"],
"renombraA ej 1" ~:
renombraA (p[x,y]) 3
==> p[V "x_3",V "y_3"],
"renombra ej 1" ~:
renombra [p[x,y], q[f[x],z]] 3
==> [p[V "x_3",V "y_3"],q[f[V "x_3"],V "z_3"]],
"resolvente ej 1" ~:
resolvente [p[x],q[x]] (p[x],[r[x]]) 3
==> [r[V "x_3"],q[V "x"]],
"aplicaA ej 1" ~:
aplicaA [(x,z),(z,f[x,y])] (p[g[x],y])
==> p[g[z],y],
"aplica ej 1" ~:
aplica [(x,z),(z,f[x,y])] [p[g[x],y], p[z]]
==> [p[g[z],y],p[f[x,y]]],
"respuestas ej 1" ~:
respuestas programa1 [p[x]]
==> [[(V "x_0", V "x"),(V "x_1",V "x")]],
"uniónGeneral ej 1" ~:
uniónGeneral [[1]]
==> [1],
"uniónGeneral ej 1" ~:
uniónGeneral [[1],[1,2],[2,3]]
==> [1,2,3],
"variablesÁtomo ej 1" ~:
variablesÁtomo (p[x,f[y,x]])
==> [x,y],
"variablesObjetivo ej 1" ~:
variablesObjetivo [p[z,f[x,y]], q[y,a,z]]
==> [z,x,y],
"valor ej 1" ~:
valor (f[x]) [(x,g[y]),(y,a)]
==> f[g[a]],
"calculaRespuesta ej 1" ~:
```

```
     calculaRespuesta [x,z] [(x,g[y]),(y,a),(z,f[x])]
     ==> [(x,g[a]),(z,f[g[a]])],
     "respuestasReducidas ej 1" ~:
     respuestasReducidas programa2 [suma [x,y,s[o]]]
     ==> [[(x,o),(y,s[o])],[(x,s[o]),(y,o)]],
     "respuestasReducidas ej 1" ~:
     take 3 (respuestasReducidas programa2 [suma [x,y,z]])
     ==> [[(x,o),(y,z),(z,z)],
          [(x,s[o]),(y,V "z_0"),(z,s[V "z_0"])],
          [(x,s[s[o]]),(y,V "z_1"),(z,s[s[V "z_1"]])]]
    ]

verificaProlog :: IO Counts
verificaProlog =
    runTestTT (test ejemplosProlog)

-- Prolog> verificaProlog
-- Cases: 18  Tried: 18  Errors: 0  Failures: 0
```

Apéndice A

Verificación de todos los programas

```
module VerificaTodo where

import SintaxisSemantica
import FormasNormales
import Clausulas
import TablerosSemanticos
import Secuentes
import DavisPutnam
import ResolucionProposicional
import RefinamientosResolucion
import ProgramacionLogicaProposicional
import Unificacion
import Prolog

import Data.List
import Test.HUnit
import Verificacion
import Debug.Trace

verificaTodo =
    runTestTT (test ((map (TestLabel "SintaxisSemantica")
                      ejemplosSintaxisSemantica) ++
                     (map (TestLabel "FormasNormales")
                      ejemplosFormasNormales) ++
                     (map (TestLabel "Clausulas")
                      ejemplosClausulas) ++
                     (map (TestLabel "TablerosSemanticos")
```

```
                        ejemplosTablerosSemanticos) ++
                       (map (TestLabel "Secuentes")
                        ejemplosSecuentes) ++
                       (map (TestLabel "DavisPutnam")
                        ejemplosDavisPutnam) ++
                       (map (TestLabel "ResolucionProposicional")
                        ejemplosResolucionProposicional) ++
                       (map (TestLabel "RefinamientosResolucion")
                        ejemplosRefinamientosResolucion) ++
                       (map (TestLabel "ProgramacionLogicaProposicional")
                        ejemplosProgramacionLogicaProposicional) ++
                       (map (TestLabel "Unificacion")
                        ejemplosUnificacion) ++
                       (map (TestLabel "Prolog")
                        ejemplosProlog)))

-- VerificaTodo> verificaTodo
-- Cases: 330  Tried: 330  Errors: 0  Failures: 0
```

Apéndice B

Resumen de funciones predefinidas de Haskell

1. `(x == y)` se verifica si `x` es igual a `y`.
2. `(x /= y)` se verifica si `x` es distinto de `y`.
3. `(x && y)` es la conjunción de `x` e `y`.
4. `(x || y)` es la disyunción de `x` e `y`.
5. `(x:ys)` es la lista obtenida añadiendo `x` al principio de `ys`.
6. `(xs ++ ys)` es la concatenación de `xs` e `ys`.
7. `[]` es la lista vacía.
8. `[x1,...,xn]` es la lista cuyos elementos son `x1`, ..., `xn`.
9. `[1..]` s la lista cuyos elementos son los números naturales.
10. `[x | x <- ys, p x]` es la lista de los elementos de `ys` que verifican `p`.
11. `[f x | x <- ys, p x]` es la lista de los valores por `f` de los elementos de `ys` que verifican `p`.
12. `(and xs)` es la conjunción de la lista de booleanos `xs`.
13. `(concat xss)` es la concatenación de la lista de listas `xss`.
14. `(delete x ys)` es el resultado de eliminar la primera ocurrencia de `x` en `ys`.
15. `(elem x ys)` se verifica si `x` pertenece a `ys`.
16. `(fst p)` es el primer elemento del par `p`.

17. (`head xs`) es el primer elemento de la lista `xs`.
18. (`intersect xs ys`) es la intersección de `xs` e `ys`.
19. (`map f xs`) es la lista obtenida aplicado `f` a cada elemento de `xs`.
20. (`not x`) es la negación lógica del booleano `x`.
21. (`noElem x ys`) se verifica si `x` no pertenece a `ys`.
22. (`nub xs`) es la lista obtenida eliminando las repeticiones de elementos en `xs`.
23. (`null xs`) se verifica si `xs` es la lista vacía.
24. (`or xs`) es la disyunción de la lista de booleanos `xs`.
25. (`putStrLn c`) imprime la cadena `c`.
26. (`show x`) es la representación de `x` como cadena.
27. (`snd p`) es el segundo elemento del par `p`.
28. (`sort xs`) es el resultado de ordenar `xs`.
29. (`union xs ys`) es la unión de `xs` e `ys`.
30. (`zip xs ys`) es la lista de pares formado por los correspondientes elementos de `xs` e `ys`.