LOGICA COMPUTACIONAL

Practica 02

11 de noviembre de 2018

1. Integrantes del equipo

- Carmona Mendoza Martin
- Rivera Mercado Sergio
- Zaldivar Rico William Oceloth

2. Implementacion de la practica

A continuacion se presenta el codigo de los ejercicios correspondientes a la practica 02 (Deduccion Natural):

```
--Practica 02
module DeduccionNatural (ReglaDN(..), showCheckDedNat)
--Verifica que los pasos de una deduccion natural sean correctos.
where
import Data.List as L (delete,(\\))
import SintaxisPL
--
--
-------------------------------------------------------------------------------------------------
-- Deduccion Natural:
--
data ReglaDN =
             Icon  Int Int | Econ1 Int | Econ2 Int  -- reglas para la conjuncion
```

```
          | Iimp Int Int | Eimp Int Int |Simp Int -- reglas para la
              implicacion
          | Ineg Int Int | Eneg Int Int           -- reglas para la negacion
          | Idis1 Int | Idis2 Int | Edis Int Int Int -- reglas para la
              disyuncion
          | Ebot Int -- regla para bottom (no hay introduccion de bottom)
          | Isup     -- regla para suposiciones (Assumptions). Las
              suposiciones se introducen con cajas en una prueba.
          | Prem     -- ragla para premisas (premises). Las premisas se
              consideran validas en una prueba.
          | E2neg Int -- regla para eliminacion de doble negacion
          | Itop     -- regla para top (no hay eliminacion de top). Esta regla
              no se usa.
          | Copy Int -- Esta regla permite repetir una formula previa.
              Huth-Ryan p.20:
              -- The rule âĂŸcopyâĂŹ allows us to repeat something that we
                  know already.
              -- We need to do this in this example, because the rule âĘŠi
                  requires that we end the inner box with p.
              -- The copy rule entitles us to copy formulas that appeared
                  before,
              -- unless they depend on temporary assumptions whose box has
                  already been closed.
              -- Though a little inelegant, this additional rule is a small
                  price to pay
              -- for the freedom of being able to use premises, or any other
                  âĂŸvisibleâĂŹ formulas, more than once.
          deriving (Eq,Show)
--
type Caja  = (Int,Int) -- Caja de suposiciones. Huth-Ryan p.12.
                    -- (i,j), 0<i<=j: caja cerrada de i a j
                    -- (i,j), 0=j<i : caja abierta de i a ...
          -- Proofs may nest boxes or open new boxes after old ones have been
              closed.
          -- There are rules about which formulas can be used at which points
              in the proof.
          -- Generally, we can only use a formula ÏĘ in a proof at a given
              point if:
          --      (1) that formula occurs prior to that point and
          --      (2) no box which encloses that occurrence of ÏĘ has been
              closed already.
          -- The line immediately following a closed box has to match
          -- the pattern of the conclusion of the rule that uses the box.
          -- For implies-introduction, this means that we have to continue
              after the box with ÏĘ âĘŠ ÏĹ,
          -- where ÏĘ was the first and ÏĹ the last formula of that box.
--
type Paso  = (PL,ReglaDN,[Caja]) -- Un paso de una prueba,
                                 -- (formula,regla_aplicada,listaDeCajas)
```

```
type NumPaso= (Int,Paso)        -- Un paso numerado, (numero, paso)
--
phiPasoNum :: Int->[NumPaso] -> PL
--formula del paso numero i en lpasos
phiPasoNum i lpasos = case mpi of
                  Just (fi,_,_) -> fi
                  _             -> error $ "phiPasoNum: i fuera de rango,
                      (i,lpasos)="++show (i,lpasos)
    where
    mpi = lookup i lpasos
--
ultimoPaso :: [NumPaso] -> NumPaso
ultimoPaso lpasos
    | lpasos /= [] = (n,p)
--     | otherwise = error $ "ultimoPaso: no hay pasos, lpasos="++show lpasos
    | otherwise = (0,(Top,Itop,[])) -- (nN,(fN,r,lcN))
    where
    (n,p)  = last lpasos
--
eqCon1 :: PL -> PL -> Bool
-- True si g es el conyunto 1 de f
eqCon1 g f = case f of
            f1 `Oand` _  -> g == f1
            _            -> False
--
eqCon2 :: PL -> PL -> Bool
-- True si g es el conyunto 2 de f
eqCon2 g f = case f of
            _ `Oand` f2  -> g == f2
            _            -> False
--
usableP :: Int->[Caja]->Int -> Bool
-- True si el paso j es usable. Es decir, si 0<j<=nN y j no esta en ninguna
    caja cerrada.
usableP j lcajas nN = 0<j && j<=nN  -- j>0 y j es menor o igual que el ultimo
    paso previo
                  && and [not (k<=j && j<=l) | (k,l) <- cajasCerradas] -- j no
                      estÃą en ninguna caja cerrada.
                  where
                  cajasCerradas= [(k,l) | (k,l) <- lcajas, l/=0]
--
cerrarCaja :: [Caja]->Int->Int -> [Caja]
-- cierra correctamente la caja (i,0) de lcajas.
cerrarCaja lcajas i j
    | (i,0) `notElem` cajasAbiertas  = error laCajaNoEstaAbierta
    | cajasInternasAbiertas /= []    = error hayUnaCajaInternaAbierta
    | j <= 0                         = error jDebeSerPositivo
    | otherwise                      = (i,j): (L.delete (i,0) lcajas)
    where
```

```
    laCajaNoEstaAbierta   = "\n cerrarCaja: la caja "++show (i,j) ++" no esta
        abierta."
    hayUnaCajaInternaAbierta= "\n cerrarCaja: hay al menos una caja interna
        abierta: "++show (head cajasInternasAbiertas)
    jDebeSerPositivo      = "\n cerrarCaja: el final de la caja debe se
        positivo, j= "++show j
    cajasAbiertas         = [(k,l) | (k,l) <- lcajas, l==0]
    cajasInternasAbiertas = [(k,l) | (k,l) <- cajasAbiertas, i<k]
--
esDisyuncion :: PL-> (Bool,PL,PL)
--Regresa (True,g,h) si f= g v h.
esDisyuncion f = case f of
                    g `Oor`h -> (True,g,h)
                    _         -> (False,Bot,Bot)
--
checkPaso :: [PL]->[NumPaso]->NumPaso -> Bool
checkPaso lprem lpp p = -- listaDePremisas listaDePasosPrevios pasoActual
    case p of
        --Reglas para la conjuncion:
        (m,(g `Oand` h,Icon i j,lc)) -> lpp/=[]     -- hay pasos previos
                                     && m==nN+1        -- m se incrementa en 1.
                                     && lc== lcN       -- las cajas no cambiaron
                                     && usableP i lc nN -- i es usable, i<nN &&
                                         i no esta en una caja cerrada
                                     && usableP j lc nN -- j es usable, j<nN &&
                                         j no esta en una caja cerrada
                                     && g==fi && h==fj -- introduccion de la
                                         conjuncion: fi,fj |- fi & fj
                                         where
                                         fi= phiPasoNum i lpp -- paso i
                                         fj= phiPasoNum j lpp -- paso j
        (m,(g,Econ1 i,lc))          -> lpp/=[]         -- hay pasos previos
                                     && m==nN+1        -- m se incrementa en 1.
                                     && lc== lcN       -- las cajas no cambiaron
                                     && usableP i lc nN -- i es usable, i<nN &&
                                         i no esta en una caja cerrada
                                     && g `eqCon1` fi  -- g es el conyunto 1 de
                                         fi: gi & hi |- gi
                                         where
                                         fi = phiPasoNum i lpp -- paso i, fi= gi
                                             & hi
        (m,(h,Econ2 i,lc))          -> lpp/=[]         -- hay pasos previos
                                     && m==nN+1        -- m se incrementa en 1.
                                     && lc== lcN       -- las cajas no cambiaron
                                     && usableP i lc nN -- i es usable, i<nN &&
                                         i no esta en una caja cerrada
                                     && h `eqCon2` fi  -- h es el conyunto 2 de
                                         fi: gi & hi |- hi
                                         where
```

```
                              fi = phiPasoNum i lpp -- paso i, fi= gi
                                  & hi
 --Reglas para la disyuncion:
(m,(g `Oor` _,Idis1 i,lc)) -> lpp/=[]               -- hay pasos previos
                             && m==nN+1        -- m se incrementa en 1.
                             && lc== lcN       -- las cajas no cambiaron
                             && usableP i lc nN -- i es usable, i<nN &&
                                 i no esta en una caja cerrada
                             && g==fi   -- introduccion de la
                                 conjuncion: fi |- fi || fj
                             where
                                fi= phiPasoNum i lpp -- paso i
(m,(_ `Oor` h,Idis2 j,lc)) -> lpp/=[]               -- hay pasos previos
                             && m==nN+1        -- m se incrementa en 1.
                             && lc== lcN       -- las cajas no cambiaron
                             && usableP j lc nN -- j es usable, j<nN &&
                                 j no esta en una caja cerrada
                             && h==fj   -- introduccion de la
                                 conjuncion: fj |- fi | fj
                             where
                                fj= phiPasoNum j lpp -- paso i
(m,(h,Edis i j k,lc)) ->     lpp/=[]                -- hay pasos previos
                             && m==nN+1        -- m se incrementa en 1.
                             && usableP i lcN nN -- i es usable, i<=nN
                                 && i no esta en una caja cerrada
                             && usableP j lcN nN -- j es usable, j<=nN
                                 && j no esta en una caja cerrada
                             && h==fj          -- introduccion de la
                                 implicacion: i->k j->k |- k->fj
                                 where
                                 fj= phiPasoNum k lpp
--Reglas para la implicacion:
(m,(_ `Oimp` h,Iimp i j,lc)) -> lpp/=[]             -- hay pasos previos
                             && m==nN+1             -- m se
                                 incrementa en 1.
                             && j==nN && h==fN        -- h debe ser la
                                 del paso inmediato anterior.Huth-Ryan
                             && lc L.\\ lcNijCerrada==[] -- se cerro la
                                 caja (i,j)
                             && usableP i lcN nN -- i es usable, i<=nN
                                 && i no esta en una caja cerrada
                             && usableP j lcN nN -- j es usable, j<=nN
                                 && j no esta en una caja cerrada
                             && h==fj          -- introduccion de la
                                 implicacion: ...fj |- g->fj
                                 where
                                 lcNijCerrada= cerrarCaja lcN i j
                                 fj= phiPasoNum j lpp -- formula del
                                     paso j.
```

```
(m,(h,Eimp i j,lc))        -> lpp/=[]        -- hay pasos previos
                              && m==nN+1        -- m se incrementa en 1.
                              && lc== lcN       -- las cajas no cambiaron
                              && usableP i lc nN -- i es usable, i<nN &&
                                  i no esta en una caja cerrada
                              && usableP j lc nN -- j es usable, j<nN &&
                                  j no esta en una caja cerrada
                              && fj==fi `Oimp` h -- eliminacion de la
                                  implicacion: fi,fi->h |- h
                                  where
                                  fi= phiPasoNum i lpp -- paso i
                                  fj= phiPasoNum j lpp -- paso j
--Reglas para la negacion (Âňg = g -> Bot):
(m,(Oneg _,Ineg i j,lc))  -> lpp/=[]               -- hay pasos previos
                              && m==nN+1               -- m se
                                  incrementa en 1.
                              && j==nN && Bot==fN      -- Bot debe ser
                                  la del paso inmediato
                                  anterior.Huth-Ryan
                              && lc L.\\ lcNijCerrada==[] -- se cerro la
                                  caja (i,j)
                              && usableP i lcN nN -- i es usable, i<=nN
                                  && i no esta en una caja cerrada
                              && usableP j lcN nN -- j es usable, j<=nN
                                  && j no esta en una caja cerrada
                              && Bot==fj -- introduccion de la negacion:
                                  g...Bot |- g->Bot = Âňg
                                  where
                                  lcNijCerrada= cerrarCaja lcN i j
                                  fj= phiPasoNum j lpp -- formula del
                                      paso j.
(m,(Bot,Eneg i j,lc))      -> lpp/=[]           -- hay pasos previos
                              && m==nN+1            -- m se incrementa en
                                  1.
                              && lc== lcN           -- las cajas no
                                  cambiaron
                              && usableP i lc nN    -- i es usable, i<nN
                                  && i no esta en una caja cerrada
                              && usableP j lc nN    -- j es usable, j<nN
                                  && j no esta en una caja cerrada
                              && fj==fi `Oimp` Bot  -- eliminacion de la
                                  negacion: fi,fi->Bot |- Bot
                                  where
                                  fi= phiPasoNum i lpp -- paso i
                                  fj= phiPasoNum j lpp -- paso j
(m,(g,E2neg i,lc))         -> lpp/=[]           -- hay pasos previos
                              && m==nN+1            -- m se incrementa en
                                  1.
                              && lc== lcN           -- las cajas no
```

```
                                    cambiaron
                                && usableP i lc nN    -- i es usable, i<nN
                                    && i no esta en una caja cerrada
                                && fi==fi    -- eliminacion de la doble
                                    negacion: ÂňÂňfi|- fi
                                    where
                                    fi= phiPasoNum i lpp -- paso i
        -- Regla para suposiciones (Assumptions):
        (m,(_,Isup,lc))           -> m==nN+1                      -- m se
            incrementa en 1.
                                && lc== lcN ++ [(nN+1,0)]        -- la caja
                                    (nN+1,0) se agrego a las cajas
        -- Regla para premisas (Premises):
        (m,(f,Prem,_))             -> f `elem` lprem -- basta que f este en la
            lista de premisas
                                && m==nN+1        -- m se incrementa en 1.
        -- Regla para Bot (no hay introduccion de Bot):
        (m,(_,Ebot i,lc))         -> lpp/=[]        -- hay pasos previos
                                && m==nN+1        -- m se incrementa en 1.
                                && lc== lcN       -- las cajas no cambiaron
                                && usableP i lc nN -- i es usable, i<nN &&
                                    i no esta en una caja cerrada
                                && fi==Bot        -- eliminacion de Bot:
                                    Bot |- f
                                    where
                                    fi= phiPasoNum i lpp -- paso i
        -- Regla para Top:
        (m,(Top,Itop,_))           -> True -- Top se puede derivar sin
            restricciones
                                && m==nN+1        -- m se incrementa en 1.
        -- Regla para usar formulas previas:
        (m,(f,Copy i,lc))         -> lpp/=[]        -- hay pasos previos
                                && m==nN+1        -- m se incrementa en 1.
                                && lc== lcN       -- las cajas no cambiaron
                                && usableP i lcN nN -- i es usable, i<=nN
                                    && i no esta en una caja cerrada
                                && f== fi         -- f es la formula del
                                    paso i
                                    where
                                    fi= phiPasoNum i lpp -- formula del
                                        paso i.
        _                         -> error $ "checkPaso: caso no
            implementado aun, p="++show p
        where
        (nN,(fN,_,lcN))= ultimoPaso lpp
--
checkPrueba :: [PL]->[NumPaso] -> Bool
-- True sii todos los pasos de lpasos son pasos vÃąlidos mediante alguna regla
    de deduccion natural.
```

```
checkPrueba lprem lpasos= -- listaDePremisas listaDePasos
   case lpasos of
        []    -> True -- la lista de pasos vacia es valida
        _:_   -> checkPrueba lprem lpp && checkPaso lprem lpp p
        where
        n  = length lpasos
        lpp= Prelude.take (n-1) lpasos
        p  = last lpasos
--
-----------------------------------------------------------------------------------------------------
--
showRegla :: ReglaDN->String
showRegla r= case r of
           -- reglas para la conjuncion:
           Icon  i j   -> "iCon "++show i++","++show j
           Econ1 i     -> "eCon1 "++show i
           Econ2 i     -> "eCon2 "++show i
           -- reglas para la implicacion:
           Iimp  i j   -> "iImp "++show i++"-"++show j
           Eimp i j    -> "eImp "++show i++","++show j
           -- reglas para la negacion:
           Ineg  i j   -> "iNeg "++show i++"-"++show j
           Eneg i j    -> "eNeg "++show i++","++show j
           -- reglas para la disyuncion:
           Idis1 i     -> "iDis1 "++show i
           Idis2 i     -> "iDis2 "++show i
           Edis i j k -> "eDis "++show i++","++show j++","++show k
           -- regla para bottom (no hay introduccion de bottom):
           Ebot  i     -> "eBot "++show i
           -- regla para suposiciones (Assumptions):
           Isup        -> "suposicion"
           -- regla para premisas (Premises):
           Prem        -> "premisa"
           -- regla para eliminacion de la doble negacion:
           E2neg i     -> "EÂňÂň "++show i
           -- regla para top (no hay eliminacion de top). Esta regla no se usa:
           Itop        -> "iTop"
           -- La siguiente regla permite repetir una formula previa. (***):
           Copy i      -> "copy "++show i
--            _           ->  show r
           --
--
showLphi :: [PL] -> String
--Muestra una lista de formulas.
showLphi lphi= case lphi of
                  [f]     -> showPL f
                  f:lf    -> showPL f ++","++ showLphi lf
                  []      -> ""
--
```

```
showCaja :: Caja -> String
showCaja (k,l) = showN k++"-"++ showN l
    where
    showN n= if n==0
                then "?"
                else show n
--
--
showLcajas :: [Caja] -> String
--Muestra una lista de cajas.
showLcajas lcajas= case lcajas of
                    [(i,j)] -> showCaja (i,j)
                    c:lc    -> showCaja c ++","++ showLcajas lc
                    []      -> ""
--
--
showNumPasoCheck :: Int->NumPaso->Bool -> String
-- Muestra un paso indicando, mediante b, si es correcto, o no.
showNumPasoCheck fSize (n,(f,r,lc)) b = "\t" ++ (show n) ++". "++ fS++
    spaceAfterPhi++ rS ++ lcS ++ checkS
    where
    fS              = showPL f
    spaceAfterPhi   = " " ++ Prelude.take (fSize-(length fS)) (repeat ' ')
    rS              = "\t " ++ (showRegla r)
    lcS             = ". Cajas=["++ showLcajas lc ++"]"
    checkS          = if b
                        then ". Correcto"
                        else ". Incorrecto"
--
showLpasos :: Int->[PL]->[NumPaso]->[NumPaso] -> IO ()
-- Muestra los pasos de lpasos indicando si son correctos, o no.
-- Initial call: showLpasos fSize lprem [] lpasos
showLpasos fSize lprem prevLp lpasos =
    case lpasos of
            []      -> putStr ""
            p:lps   ->  do
                        putStrLn $ showNumPasoCheck fSize p (checkPaso lprem
                            prevLp p)
                        showLpasos fSize lprem (prevLp++[p]) lps
--
--
showCheckConclusion :: [PL]->[NumPaso]->PL -> IO ()
showCheckConclusion lpremisas lpasos phi =
    do
    putStrLn mensaje
    putStrLn ""
    where
    mensaje
        | not pruebaOK = "\t*** Hay pasos incorrectos. ***"
```

```
        | lcAbiertas/=[]= "\t*** Hay cajas de suposiciones que no se cerraron
            ***: "++ showLcajas lcAbiertas
        | phi/=fN      = "\t*** La ultima fÃşrmula no es el objetivo ***: "++
            (showPL phi) ++" /= "++ (showPL fN)
        | otherwise    = "\tCorrecto. Mediante deduccion natural: "++ lpremS ++
            " |- " ++ showPL fN
    pruebaOK           = checkPrueba lpremisas lpasos
    (_,(fN,_,lc))      = ultimoPaso lpasos
    lpremS             = if lpremisas /= []
                            then "{" ++ showLphi lpremisas ++"}"
                            else ""
    lcAbiertas         = [(i,j) | (i,j)<-lc, j==0]
--
maxL :: Ord a => [a] -> a
maxL = foldr1 (\x y ->if x >= y then x else y)
--
showCheckDedNat :: [PL]->[NumPaso]->PL -> IO ()
--Muestra y verifica que lpasos sea una deduccion natural correcta de:
    lpremisas |- phi.
--Es decir, muestra y verifica que lpasos es una prueba, con deduccion
    natural, de phi a partir de lpremisas.
showCheckDedNat lpremisas lpasos phi = --listaDePremisas listaDePasos
    do
    showLpasos fSize lpremisas [] lpasos
    showCheckConclusion lpremisas lpasos phi
    where
    --fSize= 50
    fSize= maxL [length (showPL f) | (_,(f,_,_)) <- lpasos]
```

Ahora presentamos el codigo para los Ejemplos de Deduccion Natural:

```
module DeduccionNaturalEjemplos
--Muestra ejemplos de la verificacion de deducciones naturales mediante
    showCheckDedNat.
where
import SintaxisPL
import DeduccionNatural (ReglaDN(..),showCheckDedNat)
--
--
--
--
todosLosEjemplos :: IO ()
-- muestra todos los ejemplos.
todosLosEjemplos =
    do
    putStrLn ""
    putStrLn "Ejemplo thompsonP10:"
```

```
    thompsonP10
    --
    putStrLn "Ejemplo thompsonP12a:"
    thompsonP12a
    --
    putStrLn "Ejemplo thompsonP12b:"
    thompsonP12b
    --
    putStrLn "Ejemplo thompsonP12c1:"
    thompsonP12c1
    --
    putStrLn "Ejemplo huthRyanP20:"
    huthRyanP20
    --
    putStrLn "Ejemplo huthRyanP8Ej6:"
    huthRyanP8Ej6
    --
    putStrLn "Ejemplo thompsonP10:"
    thompsonP10
    --
--
ejerc1 :: IO()
ejerc1 = -- |- ((v1 -> v2) & (v2 -> v3)) -> ((v1 | v2 ) -> v3)
    let v1= Var 1
        v2= Var 2
        v3= Var 3
        gamma= []
        (âĹğ) :: PL->PL->PL
        fâĹğg= Oand f g
        (âĞŠ) :: PL->PL->PL
        fâĞŠg= Oimp f g
        (âĹÍ) :: PL->PL->PL
        fâĹÍg= Oor f g
        lpasos= [  (1,((v1âĞŠv2)âĹğ(v2âĞŠv3),         Isup,      [(1,0)])),
                (2,(v1,                              Isup,      [(1,0),(2,0)])),
                (3,((v1âĹÍv2),                        Idis1 2, [(1,0),(2,0)])),
                (4,((v1âĞŠv2),                        Econ1 1,   [(1,0),(2,0)])),
                (5,(v2,                              Eimp 2 4,  [(1,0),(2,0)])),
                (6,((v2âĞŠv3),                        Econ2 1,   [(1,0),(2,0)])),
                (7,(v3,                              Eimp 5 6,  [(1,0),(2,0)])),
                (8,(((v1âĹÍv2)âĞŠv3),                      Iimp 2 7,  [(1,0),(2,7)])),
                (9,(((v1âĞŠv2)âĹğ(v2âĞŠv3)âĞŠ((v1âĹÍv2)âĞŠv3), Iimp 1 8,
                    [(1,8),(2,7)])))
                ]
        phi= ((v1âĞŠv2)âĹğ(v2âĞŠv3))âĞŠ((v1âĹÍv2)âĞŠv3)
    in showCheckDedNat gamma lpasos phi
--
ejerc2 :: IO()
ejerc2 = -- (v1 -> v2), (v2 -> v3)) |- (v1 -> v3)
```

```
    let v1= Var 1
        v2= Var 2
        v3= Var 3
        gamma= [(v1âĞŠv2),(v2âĞŠv3)]
        (âĹğ) :: PL->PL->PL
        fâĹğg= Oand f g
        (âĞŠ) :: PL->PL->PL
        fâĞŠg= Oimp f g
        lpasos= [ (1,(v1,                          Isup,      [(1,0)])),
                (2,((v1âĞŠv2),                       Prem,      [(1,0)])),
                (3,(v2,                            Eimp 1 2,  [(1,0)])),
                (4,((v2âĞŠv3),                       Prem,      [(1,0)])),
                (5,(v3,                            Eimp 3 4,  [(1,0)])),
                (6,((v1âĞŠv3),                       Iimp 1 5,  [(1,5)]))
                ]
        phi= (v1âĞŠv3)
    in showCheckDedNat gamma lpasos phi
--
ejerc3 :: IO()
ejerc3 = -- (v1 | v2)-> v3) -> ((v1 -> v3) & (v2 -> v3))
    let v1= Var 1
        v2= Var 2
        v3= Var 3
        gamma= []
        (âĹğ) :: PL->PL->PL
        fâĹğg= Oand f g
        (âĞŠ) :: PL->PL->PL
        fâĞŠg= Oimp f g
        (âĹÍ) :: PL->PL->PL
        fâĹÍg= Oor f g
        lpasos= [ (1,((v1 âĹÍ v2)âĞŠv3 ,                  Isup,      [(1,0)])),
                (2,(v1,                                Isup,
                    [(1,0),(2,0)])),
                (3,((v1 âĹÍ v2),                          Idis1 2,
                    [(1,0),(2,0)])),
                (4,(v3,                                Eimp 3 1, [(1,0),(2,0)])),
                (5,((v1âĞŠv3),                           Iimp 2 4,
                    [(1,0),(2,4)])),
                (6,(v2,                                Isup,
                    [(1,0),(2,4),(6,0)])),
                (7,((v1 âĹÍ v2),                            Idis2 6,
                    [(1,0),(2,4),(6,0)])),
                (8,(v3,                                Eimp 7 1,
                    [(1,0),(2,4),(6,0)])),
                (9,((v2âĞŠv3),                           Iimp 6 8,
                    [(1,0),(2,4),(6,8)])),
                (10,((v1âĞŠv3)âĹğ(v2âĞŠv3),                  Icon 5 9,
                    [(1,0),(2,4),(6,8)])),
                (11,(((v1âĹÍv2)âĞŠv3) âĞŠ ((v1âĞŠv3) âĹğ (v2âĞŠv3)), Iimp 1 10,
```

```
                    [(1,10),(2,4),(6,8)]))
                ]
        phi= ((v1âĹÎv2)âĞŠv3) âĞŠ ((v1âĞŠv3) âĹğ (v2âĞŠv3))
    in showCheckDedNat gamma lpasos phi
--
ejerc4 :: IO()
ejerc4 = -- (v1 -> (v2 -> v3)) -> (v1 & v2) -> v3)
    let v1= Var 1
        v2= Var 2
        v3= Var 3
        gamma= []
        (âĹğ) :: PL->PL->PL
        fâĹğg= Oand f g
        (âĞŠ) :: PL->PL->PL
        fâĞŠg= Oimp f g
        lpasos= [ (1,((v1âĞŠ(v2âĞŠv3)),              Isup,      [(1,0)])),
                (2,((v1 âĹğ v2),                    Isup,      [(1,0),(2,0)])),
                (3,(v1,                           Econ1 2,  [(1,0),(2,0)])),
                (4,(v2,                           Econ2 2,      [(1,0),(2,0)])),
                (5,((v2âĞŠv3),                       Eimp 3 1,  [(1,0),(2,0)])),
                (6,(v3,                           Eimp 4 5,  [(1,0),(2,0)])),
                (7,(((v1 âĹğ v2)âĞŠv3),               Iimp 2 6,  [(1,0),(2,6)])),
                (8,((v1âĞŠ(v2âĞŠv3))âĞŠ((v1 âĹğ v2)âĞŠv3), Iimp 1 7,
                    [(1,7),(2,6)]))
                ]
        phi= (v1âĞŠ(v2âĞŠv3))âĞŠ((v1 âĹğ v2)âĞŠv3)
    in showCheckDedNat gamma lpasos phi
--
ejerc5 :: IO()
ejerc5 = -- (v1 -> v2) -> (v2 -> v1)
    error "No se puede demostrar"
--
ejerc6 :: IO()
ejerc6 = -- v1 -> ÂňÂňv1
    let v1= Var 1
        v2= Var 2
        v3= Var 3
        gamma= [v1]
        (âĞŠ) :: PL->PL->PL
        fâĞŠg= Oimp f g
        (Âň)f= Oneg f
        lpasos= [ (1,((Âň)v1,                        Isup,      [(1,0)])),
                (2,(v1,                          Prem,      [(1,0)])),
                (3,(Bot,                          Eneg 1 2,  [(1,0)])),
                (4,((Âň)((Âň)v1),                      Ebot 3,      [(1,0)])),
                (5,(v1 âĞŠ (Âň)((Âň)v1),                 Iimp 1 4,  [(1,4)]))
                ]
        phi= v1 âĞŠ (Âň)((Âň)v1)
    in showCheckDedNat gamma lpasos phi
```

```
--
ejerc7 :: IO()
ejerc7 = --
  let v1= Var 1
      v2= Var 2
      gamma= []
      (âĹğ) :: PL->PL->PL
      fâĹğg= Oand f g
      (âĞŠ) :: PL->PL->PL
      fâĞŠg= Oimp f g
      (Âň)f= Oneg f
      (âĹĺ) :: PL->PL->PL
      fâĹĺg= Oor f g
      lpasos= [ (1,((v1âĞŠv2)âĞŠv1,              Isup,      [(1,0)])),
              (2,((Âň)((Âň)(v1âĞŠv2)âĹĺ v1),        Isup,      [(1,0),(2,0)])),
              (3,((Âň)(v1âĞŠv2),                  Isup,
                  [(1,0),(2,0),(3,0)])),
              (4,((Âň)(v1âĞŠv2)âĹĺ v1,             Idis1 3,
                  [(1,0),(2,0),(3,0)])),
              (5,(Bot,                           Eneg 2 4,  [(1,0),(2,0),(3,0)])),
              (6,( (Âň)((Âň)(v1âĞŠv2)),           Ineg 3 5,
                  [(1,0),(2,0),(3,5)])),
              (7,((v1âĞŠv2),                      E2neg 6,
                  [(1,0),(2,0),(3,5)])),
              (8,((v1,                           Isup,
                  [(1,0),(2,0),(3,5),(8,0)]))),
              (9,(((Âň)((v1âĞŠv2))âĹĺ v1,          Idis2 8,
                  [(1,0),(2,0),(3,5),(8,0)]))),
              (10,(Bot,                          Eneg 2 9,
                  [(1,0),(2,0),(3,5),(8,0)])),
              (11,((Âň)v1,                        Ineg 8 10,
                  [(1,0),(2,0),(3,5),(8,10)])),
              (12,(v1,                           Eimp 7 1,
                  [(1,0),(2,0),(3,5),(8,10)])),
              (13,(Bot,                          Eneg 11 12,
                  [(1,0),(2,0),(3,5),(8,10)])),
              (14,((Âň)((Âň)((Âň)(v1âĞŠv2)âĹĺ v1)), Ineg 2 13,
                  [(1,0),(2,13),(3,5),(8,10)])),
              (15,((Âň)(v1âĞŠv2)âĹĺ v1,            E2neg 14,
                  [(1,0),(2,13),(3,5),(8,10)])),
              (16,((Âň)(v1âĞŠv2),                 Isup,
                  [(1,0),(2,13),(3,5),(8,10),(16,0)])),
              (17,((Âň)(v1),                      Isup,
                  [(1,0),(2,13),(3,5),(8,10),(16,0),(17,0)])),
              (18,((v1),                         Isup,
                  [(1,0),(2,13),(3,5),(8,10),(16,0),(17,0),(18,0)])),
              (19,(Bot,                          Eneg 17 18,
                  [(1,0),(2,13),(3,5),(8,10),(16,0),(17,0),(18,0)])),
              (20,(v2,                           Ebot 19,
```

```
                  [(1,0),(2,13),(3,5),(8,10),(16,0),(17,0),(18,0)])),
              (21,((v1âĞŠv2),                  Iimp 18 20,
                  [(1,0),(2,13),(3,5),(8,10),(16,0),(17,0),(18,20)])),
              (22,(Bot,                       Eneg 16 21,
                  [(1,0),(2,13),(3,5),(8,10),(16,0),(17,0),(18,20)])),
              (23,((Âň)((Âň)v1),               Ineg 17 22,
                  [(1,0),(2,13),(3,5),(8,10),(16,0),(17,22),(18,20)])),
              (24,( v1,                       E2neg 23,
                  [(1,0),(2,13),(3,5),(8,10),(16,0),(17,22),(18,20)])),
              (25,((Âň)(v1âĞŠv2)âĞŠv1,          Iimp 16 24,
                  [(1,0),(2,13),(3,5),(8,10),(16,24),(17,22),(18,20)])),
              (26,(v1,                        Isup,
                  [(1,0),(2,13),(3,5),(8,10),(16,24),(17,22),(18,20),(26,0)])),
              (27,(v1âĞŠv1,                    Iimp 26 26,
                  [(1,0),(2,13),(3,5),(8,10),(16,24),(17,22),(18,20),(26,26)])),
              (28,(v1,                        Eimp 27 27,
                  [(1,0),(2,13),(3,5),(8,10),(16,24),(17,22),(18,20),(26,26)])),
              (29,(((v1âĞŠv2)âĞŠv1)âĞŠv1,          Iimp 1 28,
                  [(1,28),(2,13),(3,5),(8,10),(16,24),(17,22),(18,20),(26,26)]))
              ]
       phi= ((v1âĞŠv2)âĞŠv1)âĞŠv1
   in showCheckDedNat gamma lpasos phi
--
thompsonP10 :: IO ()
thompsonP10 = -- |- ((v1&v2)&v3) -> (v1&(v2&v3))
   let v1= Var 1
       v2= Var 2
       v3= Var 3
       gamma= []
       (âĹğ) :: PL->PL->PL
       fâĹğg= Oand f g
       (âĞŠ) :: PL->PL->PL
       fâĞŠg= Oimp f g
       lpasos= [  (1,((v1âĹğv2)âĹğv3,          Isup,      [(1,0)])),
              (2,((v1âĹğv2),                Econ1 1,   [(1,0)])),
              (3,(v1,                      Econ1 2,   [(1,0)])),
              (4,(v2,                      Econ2 2,   [(1,0)])),
              (5,(v3,                      Econ2 1,   [(1,0)])),
              (6,(v2âĹğv3,                   Icon 4 5,  [(1,0)])),
              (7,(v1âĹğ(v2âĹğv3),             Icon 3 6,  [(1,0)])),
              (8,(((v1âĹğv2)âĹğv3)âĞŠ(v1âĹğ(v2âĹğv3)), Iimp 1 7, [(1,7)]))
              ]
       phi= ((v1âĹğv2)âĹğv3)âĞŠ(v1âĹğ(v2âĹğv3))
   in showCheckDedNat gamma lpasos phi
--
thompsonP12a :: IO ()
thompsonP12a = -- |- ((v1 âĹğ v2) âĞŠ v3) âĞŠ (v1 âĞŠ (v2 âĞŠ v3))
   let v1= Var 1
       v2= Var 2
```

```
        v3= Var 3
        gamma= []
        (âĹğ) :: PL->PL->PL
        fâĹğg= Oand f g
        (âĞŠ) :: PL->PL->PL
        fâĞŠg= Oimp f g
        lpasos= [  (1,((v1âĹğv2)âĞŠv3,              Isup,      [(1,0)])),
                   (2,(v1,                      Isup,      [(1,0),(2,0)])),
                   (3,(v2,                      Isup,      [(1,0),(2,0),(3,0)])),
                   (4,(v1âĹğv2,                   Icon 2 3,
                       [(1,0),(2,0),(3,0)])),
                   (5,((v1âĹğv2)âĞŠv3,              Copy 1,
                       [(1,0),(2,0),(3,0)])),
                   (6,(v3,                      Eimp 4 5,  [(1,0),(2,0),(3,0)])),
                   (7,(v2 âĞŠ v3,                 Iimp 3 6,
                       [(1,0),(2,0),(3,6)])),
                   (8,(v1 âĞŠ(v2 âĞŠ v3),            Iimp 2 7,
                       [(1,0),(2,7),(3,6)])),
                   (9,(((v1âĹğv2)âĞŠv3)âĞŠ(v1âĞŠ(v2âĞŠv3)), Iimp 1 8,
                       [(1,8),(2,7),(3,6)]))
                   ]
        phi= ((v1âĹğv2)âĞŠv3)âĞŠ(v1âĞŠ(v2âĞŠv3))
    in showCheckDedNat gamma lpasos phi
--
thompsonP12b :: IO ()
--  1. v1 Sup; 2. v1->v1 iImp 1-1.
-- Huth-Ryan p.13:
-- The rule âĘŠi (with conclusion ÏĘ âĘŠ ÏĹ) does not prohibit the possibility
    that ÏĘ and ÏĹ coincide.
-- They could both be instantiated to p.
thompsonP12b = -- |- v1->v1
    let
        gamma  = []
        v1     = Var 1
        (âĞŠ) :: PL->PL->PL
        fâĞŠg    = Oimp f g
        lpasos = [ (1,(v1,   Isup,      [(1,0)])),
                   (2,(v1âĞŠv1, Iimp 1 1,  [(1,1)]))
                   ]
        phi    = v1âĞŠv1
    in showCheckDedNat gamma lpasos phi
--
thompsonP12c1 :: IO ()
-- 1. v2 Sup; 2. v1->v2 iImp 1-1; 3. v2->(v1->v2)
thompsonP12c1 = -- |- v2->(v1->v2) Incorrecta
    let v1= Var 1
        v2= Var 2
        gamma= []
        (âĞŠ) :: PL->PL->PL
```

```
        fâĞŠg= Oimp f g
        lpasos = [ (1,(v2,           Isup,      [(1,0)])),
                   (2,(v1âĞŠv2,        Iimp 1 1,  [(1,0)])),
                   (3,(v2âĞŠ(v1âĞŠv2),   Iimp 1 2,  [(1,1)]))
                   ]
        phi = v2âĞŠ(v1âĞŠv2)
    in showCheckDedNat gamma lpasos phi
--
huthRyanP20 :: IO ()
huthRyanP20 = -- |- v2->(v1->v2) Correcta
    let v1= Var 1
        v2= Var 2
        gamma= []
        (âĞŠ) :: PL->PL->PL
        fâĞŠg= Oimp f g
        lpasos = [ (1,(v2,           Isup,      [(1,0)])),
                   (2,(v1,           Isup,      [(1,0),(2,0)])),
                   (3,(v2,           Copy 1,    [(1,0),(2,0)])),
                   (4,(v1âĞŠv2,        Iimp 2 3,  [(1,0),(2,3)])),
                   (5,(v2âĞŠ(v1âĞŠv2),   Iimp 1 4,  [(1,4),(2,3)]))
                   ]
        phi = v2âĞŠ(v1âĞŠv2)
    in showCheckDedNat gamma lpasos phi
--
--
huthRyanP8Ej6 :: IO ()
huthRyanP8Ej6 = -- {(v1 âĹğ v2) âĹğ v3, v4 âĹğ v5} |âĹŠ v2 âĹğ v4
    let v1= Var 1
        v2= Var 2
        v3= Var 3
        v4= Var 4
        v5= Var 5
        gamma= [(v1âĹğv2)âĹğv3, v4âĹğv5]
        (âĹğ) :: PL->PL->PL
        fâĹğg= Oand f g
        lpasos = [ (1,((v1âĹğv2)âĹğv3,   Prem,      [])),
                   (2,(v4âĹğv5,        Prem,      [])),
                   (3,(v1âĹğv2,        Econ1 1,   [])),
                   (4,(v2,           Econ2 3,   [])),
                   (5,(v4,           Econ1 2,   [])),
                   (6,(v2âĹğv4,        Icon 4 5,  []))
                   ]
        phi = v2âĹğv4
    in showCheckDedNat gamma lpasos phi
```

Las unicas complicaciones a mencionar en este punto fue para el ejercicio 7 donde al final no poseemos una regla de simplificacion tal que v1->v1-=v1 por lo cual se puso Eimp con el mismo modulo para simularlo.

En el uso de la regla de Eneg no pudimos descifrar el uso correcto de Bot por lo cual su utilizacion en los ejercicios nos muestra como un paso incorrecto.

A continuacion se muestra la implementacion de tableau comenzando por la clase de tableau y luego por las pruebas:

```
module DeduccionTableaus
--Verifica que una deduccion mediante un tableau sea correcta.
--mcb
where
import Data.List as L (delete,(\\)) -- (nub,union)
--import Data.Set as S
import SintaxisPL
--
--
------------------------------------------------------------------------------------------------------
-- Deduccion con tableaus:
-- Referencia: vanBenthem-vanEijck. Logic in Action, CapÃŋtulo 8. 2016
    (disponible en el grupo)
--
--
-- Un tableau es un arbol.
-- Each node in the tree is called a sequent.
-- A tree of sequents is called a tableau.
-- A branch of such a tableau is closed if its end node contains a sequent
-- with a formula which appears both on the left (true) and on the right
    (false) part of the sequent.
--
-- As to distinguish open and closed branches
-- we replace the truth-falsity separation symbol âŮę by and âĂć respectively.
--
data Tsequent  = Sep | Closed | Open --Tipo de "sequent": separacion, cerrado,
    abierto
                deriving (Eq,Show)
type Sequent   = ([PL],Tsequent,[PL])

data ReglaT = --Reglas de reduccion para tableaus
             ConI | ConD  -- reglas para Conjuncion, izquierda y Derecha
           | DisI | DisD  -- reglas para Disyuncion, izquierda y Derecha
           | ImpI | ImpD  -- reglas para Implicacion, izquierda y Derecha
           | NegI | NegD  -- reglas para Negacion, izquierda y Derecha
           | EquI | EquD  -- reglas para Equivalencia, izquierda y Derecha
           deriving (Eq,Show)

data Tableau   =  -- Un tableau es un arbol de "sequents"
                Hoja Sequent                          -- hoja de arbol (sin
                    descendientes)
              | UnaRama Sequent ReglaT Tableau        -- arbol de una rama
              | DosRamas Sequent ReglaT Tableau Tableau -- arbol de dos ramas
```

```
                  (izquierda y derecha)
              deriving (Eq,Show)

--
--
--
--Tableaus:
    ------------------------------------------------------------------------------
--
-- 1. Representar con una variable de tipo Tableau los dos tableaus de
-- la figura 8.1 (p.8-6) de vanBenthem-vanEijck. Logic in Action, CapÃŋtulo 8.
tFig8_1a :: Tableau
tFig8_1a =
    let
        v1= Var 1
        v2= Var 2
        v3= Var 3
    in
    Hoja ([Oand v1 (Oor v2 v3)],Sep,[Oor (Oand v1 v2) v3])
tFig8_1b :: Tableau
tFig8_1b =
    let
        v1= Var 1
        v2= Var 2
        v3= Var 3
    in
    Hoja ([Oor(Oand v1 v2) v3],Sep,[Oand v1 (Oor v2 v3)])
--
-- 2. Representar con una variable de tipo Tableau los dos tableaus de
-- la figura 8.2 (p.8-7) de vanBenthem-vanEijck. Logic in Action, CapÃŋtulo 8.
tFig8_2a :: Tableau
tFig8_2a =
    let
        v1= Var 1
        v2= Var 2
    in
    Hoja ([Oand(Oneg v1) (Oneg v2)],Sep,[Oneg(Oand v1 v2)])
-- tFig8_2b :: Tableau
tFig8_2b :: Tableau
tFig8_2b =
    let
        v1= Var 1
        v2= Var 2
    in
    Hoja ([Oneg(Oand v1 v2)],Sep,[Oand(Oneg v1) (Oneg v2)])
--
-- Definir una funcion, hojasDe, tal que: dado un tableau t, entregue una
    lista con las hojas de t.
--
```

```
hojasDe :: Tableau -> [Sequent]
hojasDe t = case t of
  Hoja s -> [s]
  UnaRama s r t -> hojasDe t
  DosRamas s r t1 t2 -> hojasDe t1 ++ hojasDe t2
-- Definir una funcion, rammasCerradas, tal que: dado un tableau t, regrese
    True sii todas las ramas de t estÃąn cerradas.
--
ramasCerradas :: Tableau -> Bool
ramasCerradas t = and [estaCerrado h | h <- hojasDe t]

-- FunciÃşn auxiliar que nos indica si el sequent esta cerrado.
estaCerrado :: Sequent -> Bool
estaCerrado (t,_,f) = or [elem v f | v <- obtenAtomicas t]

-- FunciÃşn que dada una lista de formulas nos regresa las formulas atomicas
obtenAtomicas :: [PL] -> [PL]
obtenAtomicas f = case f of
  [] -> []
  (l:ls) -> case l of
    Top -> [Top] ++ obtenAtomicas ls
    Bot -> [Bot] ++ obtenAtomicas ls
    Var n -> [Var n] ++ obtenAtomicas ls
    Oneg phi -> case phi of
      Var n -> [Oneg (Var n)] ++ obtenAtomicas ls
      _ -> obtenAtomicas ls
    _ -> obtenAtomicas ls
-- Definir una funcion, ramaAbierta, tal que: dado un tableau t, regrese True
    sii t tiene una rama abierta.
--
ramaAbierta :: Tableau -> Bool
ramaAbierta t = not (ramasCerradas t)
-- Definir una funcion, checkTableau, tal que: dado un tableau t, regresa True
    sii
-- todas las ramas de t estan construidas de acuerdo a la especificacion de la
    reglas de reduccion.
--
checkTableau :: Tableau -> Bool
checkTableau t = case t of
  -- Reglas para las hojas
  (Hoja s@(v, Closed, f)) -> estaCerrado s -- Si la hoja dice estar cerrada
      hay que verificarlo
  (Hoja s@(v, Open, f)) -> not (estaCerrado s) -- Si la hoja dice estar
      abierta hay que verificarlo
  -- Reglas para el conjunto de formulas de la izquierda
  -- Conjuncion Izquierda
  (UnaRama s@(v,ts,f) ConI t) -> ts == Sep && -- El seuquent no debe estar
      abierto o cerrado
                                 length v + 1 == length vh && -- La lista
```

```
                              resultante debe tener un elemento mÃąs ([Oand
                              v1 v2], [v1, v2])
                          conjFaltante cf avh && -- Verificamos que se
                              elimino de manera correcta la conjunciÃşn
                          checkTableau t -- Revisamos la rama
                          where
                            cf = elementosFaltantes v vh
                            avh = conjPosibles (obtenAtomicas vh)
                            vh = obtenListaIzquierda t
-- Conjuncion Derecha
(DosRamas s@(v,ts,f) ConD ti td) -> ts == Sep && -- El sequent no debe estar
    abierto o cerrado
                               length v == length vhi && -- Verificamos que
                                   sean del mismo tamaÃśo
                               length v == length vhd && -- Verificamos que
                                   sean del mismo tamaÃśo
                               dfi == dfd && -- Verificamos que falten los
                                   mismo elementos
                               elem a vhi && -- Verificamos que la primera
                                   parte de la conjuncion esta en la rama
                                   izquierda
                               elem b vhd && -- Verificamos que la segunda
                                   parte de la conjuncion esta en la rama
                                   derecha
                               checkTableau ti && -- Revisamos la rama
                                   izquierda
                               checkTableau td -- Revisamos la rama derecha
                               where
                                 vhi = obtenListaIzquierda ti
                                 vhd = obtenListaIzquierda td
                                 dfi = elementosFaltantes v vhi
                                 dfd = elementosFaltantes v vhd
                                 d = dfi !! 0
                                 (a,b) = elementosCon d
-- DisyunciÃşn Izquierda
(DosRamas s@(v,ts,f) DisI ti td) -> ts == Sep && -- El sequent no debe estar
    abierto o cerrado
                               length v == length vhi && -- Verificamos que
                                   sean del mismo tamaÃśo
                               length v == length vhd && -- Verificamos que
                                   sean del mismo tamaÃśo
                               dfi == dfd && -- Verificamos que falten los
                                   mismo elementos
                               elem a vhi && -- Verificamos que la primera
                                   parte de la disyuncion esta en la rama
                                   izquierda
                               elem b vhd && -- Verificamos que la segunda
                                   parte de la disyuncion esta en la rama
                                   derecha
```

```
                                   checkTableau ti && -- Revisamos la rama
                                       izquierda
                                   checkTableau td -- Revisamos la rama derecha
                                   where
                                     vhi = obtenListaIzquierda ti
                                     vhd = obtenListaIzquierda td
                                     dfi = elementosFaltantes v vhi
                                     dfd = elementosFaltantes v vhd
                                     d = dfi !! 0
                                     (a,b) = elementosDis d
-- Disyuncion Derecha
(UnaRama s@(v,ts,f) DisD t) -> ts == Sep && -- El seuquent no debe estar
    abierto o cerrado
                              length v + 1 == length vh && -- La lista
                                  resultante debe tener un elemento mÃąs ([Oor
                                  v1 v2], [v1, v2])
                              disjFaltante cf avh && -- Verificamos que se
                                  elimino de manera correcta la disyunciÃşn
                              checkTableau t -- Revisamos la rama
                              where
                                cf = elementosFaltantes v vh
                                avh = disjPosibles (obtenAtomicas vh)
                                vh = obtenListaDerecha t
-- Implicacion Izquierda
(DosRamas s@(v,ts,f) ImpI ti td) -> ts == Sep && -- El sequent no debe estar
    abierto o cerrado
                                   length v == length vhi && -- Verificamos que
                                       sean del mismo tamaÃśo
                                   length v == length vhd && -- Verificamos que
                                       sean del mismo tamaÃśo
                                   dfi == dfd && -- Verificamos que falten los
                                       mismo elementos
                                   elem a vhi && -- Verificamos que la primera
                                       parte de la implicacion esta en la rama
                                       izquierda
                                   elem b vhd && -- Verificamos que la segunda
                                       parte de la implicacion esta en la rama
                                       derecha
                                   checkTableau ti && -- Revisamos la rama
                                       izquierda
                                   checkTableau td -- Revisamos la rama derecha
                                   where
                                     vhi = obtenListaIzquierda ti
                                     vhd = obtenListaIzquierda td
                                     dfi = elementosFaltantes v vhi
                                     dfd = elementosFaltantes v vhd
                                     d = dfi !! 0
                                     (a,b) = elementosImp d
-- Implicacion Derecha
```

```
  (UnaRama s@(v,ts,f) ImpD t) -> ts == Sep && -- El seuquent no debe estar
      abierto o cerrado
                              length v + 1 == length vh && -- La lista
                                  resultante debe tener un elemento mÃąs ([Oimp
                                  v1 v2], [v1, v2])
                              impFaltante cf avh && -- Verificamos que se
                                  elimino de manera correcta la implicacion
                              checkTableau t -- Revisamos la rama
                              where
                                cf = elementosFaltantes v vh
                                avh = impPosibles (obtenAtomicas vh)
                                vh = obtenListaDerecha t
  --Negaciones
  (UnaRama s@(v,ts,f) NegI t) -> ts == Sep && -- El seuquent no debe estar
      abierto o cerrado
                              length v + 1 == length vh && -- La lista
                                  resultante debe tener un elemento mÃąs ([Oimp
                                  v1 v2], [v1, v2])
                              negFaltante cf avh && -- Verificamos que se
                                  elimino de manera correcta la negaciÃşn
                              checkTableau t -- Revisamos la rama
                              where
                                cf = elementosFaltantes v vh
                                avh = negPosibles (obtenAtomicas vh)
                                vh = obtenListaIzquierda t
  (UnaRama s@(v,ts,f) NegD t) -> ts == Sep && -- El seuquent no debe estar
      abierto o cerrado
                              length v + 1 == length vh && -- La lista
                                  resultante debe tener un elemento mÃąs ([Oimp
                                  v1 v2], [v1, v2])
                              negFaltante cf avh && -- Verificamos que se
                                  elimino de manera correcta la negaciÃşn
                              checkTableau t -- Revisamos la rama
                              where
                                cf = elementosFaltantes v vh
                                avh = negPosibles (obtenAtomicas vh)
                                vh = obtenListaDerecha t
-- Funciones auxiliares

-- FunciÃşn que nos regresa la lista con las formulas verdaderas
obtenListaIzquierda :: Tableau -> [PL]
obtenListaIzquierda t = case t of
  Hoja s@(v,ts,f) -> v
  UnaRama s@(v,ts,f) r st -> v
  DosRamas s@(v,ts,f) r ti td -> v
-- FunciÃşn que nos regresa la lista con las formulas verdaderas
obtenListaDerecha :: Tableau -> [PL]
obtenListaDerecha t = case t of
  Hoja s@(v,ts,f) -> v
```

```
  UnaRama s@(v,ts,f) r st -> v
  DosRamas s@(v,ts,f) r ti td -> v

-- FunciÃşn que nos regresa la lista con los elementos que estan en la primera
    y no en la segunda
elementosFaltantes :: [PL] -> [PL] -> [PL]
elementosFaltantes l1 l2 = l1 \\ l2

-- FunciÃşn que nos dice las conjunciones posibles de una lista
conjPosibles :: [PL] -> [PL]
conjPosibles f = [Oand alpha beta | alpha <- f, beta <- f, alpha /= beta]

-- FunciÃşn que nos dice si la conjunciÃşn fue eliminada correctamente
conjFaltante :: [PL] -> [PL] -> Bool
conjFaltante l1 l2 = length l1 == 1 &&
                     or [elem x l2 | x <- l1]
-- FunciÃşn que nos regresa los elementos de una conjunciÃşn
elementosCon :: PL -> (PL,PL)
elementosCon phi = case phi of
  Oand alpha beta -> (alpha, beta)
  _ -> error $ "No es una conjuncion"

-- FunciÃşn que nos dice las disjunciones posibles de una lista
disjPosibles :: [PL] -> [PL]
disjPosibles f = [Oor alpha beta | alpha <- f, beta <- f, alpha /= beta]

-- FunciÃşn que nos dice si la disjuncion fue eliminada correctamente
disjFaltante :: [PL] -> [PL] -> Bool
disjFaltante l1 l2 = length l1 == 1 &&
                     or [elem x l2 | x <- l1]
-- FunciÃşn que nos regresa los elementos de una disjuncion
elementosDis :: PL -> (PL,PL)
elementosDis phi = case phi of
  Oor alpha beta -> (alpha, beta)
  _ -> error $ "No es una disjuncion"

-- FunciÃşn que nos dice las implicaciones posibles de una lista
impPosibles :: [PL] -> [PL]
impPosibles f = [Oimp alpha beta | alpha <- f, beta <- f, alpha /= beta]

-- FunciÃşn que nos dice si la implicacion fue eliminada correctamente
impFaltante :: [PL] -> [PL] -> Bool
impFaltante l1 l2 = length l1 == 1 &&
                     or [elem x l2 | x <- l1]
-- FunciÃşn que nos regresa los elementos de una implicacion
elementosImp :: PL -> (PL,PL)
elementosImp phi = case phi of
  Oimp alpha beta -> (alpha, beta)
  _ -> error $ "No es una implicacion"
```

```
-- FunciÃşn que nos dice las negaciones posibles de una lista
negPosibles :: [PL] -> [PL]
negPosibles f = [Oneg alpha | alpha <- f]

-- FunciÃşn que nos dice si la implicacion fue eliminada correctamente
negFaltante :: [PL] -> [PL] -> Bool
negFaltante l1 l2 = length l1 == 1 &&
                   or [elem x l2 | x <- l1]
```

Ahora mostramos las pruebas unitarias:

```
module DeduccionTableausEjemplos
where
import SintaxisPL
import DeduccionTableaus


-- Ejemplos de Tableaus para verificar las reglas ya hechas

tEjem1 :: Bool
tEjem1 = -- ([v1 âĹğ v2],Sep,[v1]) --ConI--> ([v1,v2],Closed,[v1])
    let
        v1= Var 1
        v2= Var 2
    in
    checkTableau (UnaRama ([Oand v1 v2],Sep,[v1]) ConI (Hoja
        ([v1,v2],Closed,[v1])))

tEjem2 :: Bool
tEjem2 =
  let
    v1 = Var 1
    v2 = Var 2
    in
    checkTableau (DosRamas ([Oor v1 v2], Sep,[v1,v2]) DisI (Hoja ([v1], Closed,
        [v1,v2])) (Hoja ([v2], Closed, [v1,v2])))
--Implicacion
tEjem3 :: Bool
tEjem3 =
    let
        v1= Var 1
        v2= Var 2
    in
    checkTableau (UnaRama ([Oimp v1 v2],Sep,[v1]) ImpD (Hoja
        ([v1,v2],Closed,[v1])))

tEjem4 :: Bool
```

```
tEjem4 =
  let
    v1 = Var 1
    v2 = Var 2
    in
    checkTableau (DosRamas ([Oimp v1 v2], Sep,[v1,v2]) ImpI (Hoja ([v1],
        Closed, [v1,v2])) (Hoja ([v2], Closed, [v1,v2])))
--Conjuncion Derecha
tEjem5 :: Bool
tEjem5 =
  let
    v1 = Var 1
    v2 = Var 2
    in
    checkTableau (DosRamas ([Oand v1 v2], Sep,[v1,v2]) ConD (Hoja ([v1],
        Closed, [v1,v2])) (Hoja ([v2], Closed, [v1,v2])))
--Disyuncion Derecha
tEjem6 :: Bool
tEjem6 =
  let
    v1 = Var 1
    v2 = Var 2
    in
    checkTableau (UnaRama ([Oor v1 v2],Sep,[v1]) DisD (Hoja
        ([v1,v2],Closed,[v1])))
--Negacion
tEjem7 :: Bool
tEjem7 =
  let
    v1 = Var 1
    v2 = Var 2
    in
    checkTableau (UnaRama ([Oneg(Oor v1 v2)],Sep,[v1]) NegI (Hoja
        ([v1,v2],Closed,[v1])))

tEjem8 :: Bool
tEjem8 =
  let
    v1 = Var 1
    v2 = Var 2
    in
    checkTableau (UnaRama ([Oneg(Oor v1 v2)],Sep,[v1]) NegD (Hoja
        ([v1,v2],Closed,[v1])))
```

Para la implementacion de los tableau se tomo en cuenta las estructuras vistas en laboratorio y los ejemplos se implementaron con la estuctura Hoja sequent.