

Tema 17: El TAD de los conjuntos

Informática (2017–18)

José A. Alonso Jiménez

Grupo de Lógica Computacional
Departamento de Ciencias de la Computación e I.A.
Universidad de Sevilla

Tema 17: El TAD de los conjuntos

1. Especificación del TAD de los conjuntos

Signatura del TAD de los conjuntos

Propiedades del TAD de los conjuntos

2. Implementaciones del TAD de los conjuntos

Los conjuntos como listas no ordenadas con duplicados

Los conjuntos como listas no ordenadas sin duplicados

Los conjuntos como listas ordenadas sin duplicados

Los conjuntos de números enteros mediante números binarios

3. Comprobación de las implementaciones con QuickCheck

Librerías auxiliares

Generador de conjuntos

Especificación de las propiedades de los conjuntos

Comprobación de las propiedades

Tema 17: El TAD de los conjuntos

1. Especificación del TAD de los conjuntos
 - Signatura del TAD de los conjuntos
 - Propiedades del TAD de los conjuntos
2. Implementaciones del TAD de los conjuntos
3. Comprobación de las implementaciones con QuickCheck

Signatura del TAD de los conjuntos

► Signatura:

```
vacio,      :: Conj a
inserta     :: Eq a => a -> Conj a -> Conj a
elimina     :: Eq a => a -> Conj a -> Conj a
pertenece  :: Eq a => a -> Conj a -> Bool
esVacio    :: Conj a -> Bool
```

► Descripción de las operaciones:

- `vacio` es el conjunto vacío.
- `(inserta x c)` es el conjunto obtenido añadiendo el elemento `x` al conjunto `c`.
- `(elimina x c)` es el conjunto obtenido eliminando el elemento `x` del conjunto `c`.
- `(pertenece x c)` se verifica si `x` pertenece al conjunto `c`.
- `(esVacio c)` se verifica si `c` es el conjunto vacío.

Tema 17: El TAD de los conjuntos

1. Especificación del TAD de los conjuntos

Signatura del TAD de los conjuntos

Propiedades del TAD de los conjuntos

2. Implementaciones del TAD de los conjuntos

3. Comprobación de las implementaciones con QuickCheck

Propiedades del TAD de los conjuntos

1. `inserta x (inserta x c) == inserta x c`
2. `inserta x (inserta y c) == inserta y (inserta x c)`
3. `not (pertenece x vacio)`
4. `pertenece y (inserta x c) == (x==y) || pertenece y c`
5. `elimina x vacio == vacio`
6. Si `x == y`, entonces
`elimina x (inserta y c) == elimina x c`
7. Si `x /= y`, entonces
`elimina x (inserta y c) == inserta y (elimina x c)`
8. `esVacio vacio`
9. `not (esVacio (inserta x c))`

Tema 17: El TAD de los conjuntos

1. Especificación del TAD de los conjuntos

2. Implementaciones del TAD de los conjuntos

Los conjuntos como listas no ordenadas con duplicados

Los conjuntos como listas no ordenadas sin duplicados

Los conjuntos como listas ordenadas sin duplicados

Los conjuntos de números enteros mediante números binarios

3. Comprobación de las implementaciones con QuickCheck

Los conjuntos como listas no ordenadas con duplicados

- Cabecera del módulo:

```
module ConjuntoConListasNoOrdenadasConDuplicados
  (Conj,
   vacio,      -- Conj a
   inserta,   -- Eq a => a -> Conj a -> Conj a
   elimina,   -- Eq a => a -> Conj a -> Conj a
   pertenece, -- Eq a => a -> Conj a -> Bool
   esVacio,   -- Conj a -> Bool
  ) where
```

Los conjuntos como listas no ordenadas con duplicados

- El tipo de los conjuntos.

```
newtype Conj a = Cj [a]
```

- Procedimiento de escritura de los conjuntos.

```
instance Show a => Show (Conj a) where
    showsPrec _ (Cj s) cad = showConj s cad
```

```
showConj []      cad = showString "{}" cad
```

```
showConj (x:xs) cad =
```

```
    showChar '{' (shows x (showl xs cad))
```

```
    where
```

```
        showl []      cad = showChar '}' cad
```

```
        showl (x:xs) cad = showChar ',' (shows x (showl xs cad))
```

Los conjuntos como listas no ordenadas con duplicados

- ▶ Ejemplo de conjunto: `c1` es el conjunto obtenido añadiéndole al conjunto vacío los elementos 2, 5, 1, 3, 7, 5, 3, 2, 1, 9 y 0.

```
| ghci > c1  
| {2,5,1,3,7,5,3,2,1,9,0}
```

```
c1 :: Conj Int
```

```
c1 = foldr inserta vacio [2,5,1,3,7,5,3,2,1,9,0]
```

- ▶ `vacio` es el conjunto vacío. Por ejemplo,

```
| ghci> vacio  
| {}
```

```
vacio :: Conj a
```

```
vacio = Cj []
```

Los conjuntos como listas no ordenadas con duplicados

- ▶ Ejemplo de conjunto: `c1` es el conjunto obtenido añadiéndole al conjunto vacío los elementos 2, 5, 1, 3, 7, 5, 3, 2, 1, 9 y 0.

```
ghci > c1
{2,5,1,3,7,5,3,2,1,9,0}
```

```
c1 :: Conj Int
```

```
c1 = foldr inserta vacio [2,5,1,3,7,5,3,2,1,9,0]
```

- ▶ `vacio` es el conjunto vacío. Por ejemplo,

```
ghci> vacio
{}
```

```
vacio :: Conj a
```

```
vacio = Cj []
```

Los conjuntos como listas no ordenadas con duplicados

- `(inserta x c)` es el conjunto obtenido añadiendo el elemento `x` al conjunto `c`. Por ejemplo,

```
c1 == {2,5,1,3,7,5,3,2,1,9,0}
inserta 5 c1 == {5,2,5,1,3,7,5,3,2,1,9,0}
```

```
inserta :: Eq a => a -> Conj a -> Conj a
inserta x (Cj ys) = Cj (x:ys)
```

- `(elimina x c)` es el conjunto obtenido eliminando el elemento `x` del conjunto `c`. Por ejemplo,

```
c1 == {2,5,1,3,7,5,3,2,1,9,0}
elimina 3 c1 == {2,5,1,7,5,2,1,9,0}
```

```
elimina :: Eq a => a -> Conj a -> Conj a
elimina x (Cj ys) = Cj (filter (/= x) ys)
```

Los conjuntos como listas no ordenadas con duplicados

- `(inserta x c)` es el conjunto obtenido añadiendo el elemento `x` al conjunto `c`. Por ejemplo,

```
c1           == {2,5,1,3,7,5,3,2,1,9,0}
inserta 5 c1 == {5,2,5,1,3,7,5,3,2,1,9,0}
```

```
inserta :: Eq a => a -> Conj a -> Conj a
inserta x (Cj ys) = Cj (x:ys)
```

- `(elimina x c)` es el conjunto obtenido eliminando el elemento `x` del conjunto `c`. Por ejemplo,

```
c1           == {2,5,1,3,7,5,3,2,1,9,0}
elimina 3 c1 == {2,5,1,7,5,2,1,9,0}
```

```
elimina :: Eq a => a -> Conj a -> Conj a
elimina x (Cj ys) = Cj (filter (/= x) ys)
```

Los conjuntos como listas no ordenadas con duplicados

- `(inserta x c)` es el conjunto obtenido añadiendo el elemento `x` al conjunto `c`. Por ejemplo,

```
c1           == {2,5,1,3,7,5,3,2,1,9,0}
inserta 5 c1 == {5,2,5,1,3,7,5,3,2,1,9,0}
```

```
inserta :: Eq a => a -> Conj a -> Conj a
inserta x (Cj ys) = Cj (x:ys)
```

- `(elimina x c)` es el conjunto obtenido eliminando el elemento `x` del conjunto `c`. Por ejemplo,

```
c1           == {2,5,1,3,7,5,3,2,1,9,0}
elimina 3 c1 == {2,5,1,7,5,2,1,9,0}
```

```
elimina :: Eq a => a -> Conj a -> Conj a
elimina x (Cj ys) = Cj (filter (/= x) ys)
```

Los conjuntos como listas no ordenadas con duplicados

- ▶ `(pertenece x c)` se verifica si `x` pertenece al conjunto `c`. Por ejemplo,

```
c1 == {2,5,1,3,7,5,3,2,1,9,0}
pertenece 3 c1 == True
pertenece 4 c1 == False
```

```
pertenece :: Eq a => a -> Conj a -> Bool
pertenece x (Cj xs) = elem x xs
```

- ▶ `(esVacio c)` se verifica si `c` es el conjunto vacío. Por ejemplo,

```
esVacio c1 ~> False
esVacio vacio ~> True
```

```
esVacio :: Conj a -> Bool
esVacio (Cj xs) = null xs
```

Los conjuntos como listas no ordenadas con duplicados

- ▶ `(pertenece x c)` se verifica si `x` pertenece al conjunto `c`. Por ejemplo,

```
c1 == {2,5,1,3,7,5,3,2,1,9,0}
pertenece 3 c1 == True
pertenece 4 c1 == False
```

```
pertenece :: Eq a => a -> Conj a -> Bool
pertenece x (Cj xs) = elem x xs
```

- ▶ `(esVacio c)` se verifica si `c` es el conjunto vacío. Por ejemplo,

```
esVacio c1 ~> False
esVacio vacio ~> True
```

```
esVacio :: Conj a -> Bool
esVacio (Cj xs) = null xs
```

Los conjuntos como listas no ordenadas con duplicados

- ▶ `(pertenece x c)` se verifica si `x` pertenece al conjunto `c`. Por ejemplo,

```
c1 == {2,5,1,3,7,5,3,2,1,9,0}
pertenece 3 c1 == True
pertenece 4 c1 == False
```

```
pertenece :: Eq a => a -> Conj a -> Bool
pertenece x (Cj xs) = elem x xs
```

- ▶ `(esVacio c)` se verifica si `c` es el conjunto vacío. Por ejemplo,

```
esVacio c1 ~> False
esVacio vacio ~> True
```

```
esVacio :: Conj a -> Bool
esVacio (Cj xs) = null xs
```

Los conjuntos como listas no ordenadas con duplicados

- `(subconjunto c1 c2)` se verifica si `c1` es un subconjunto de `c2`. Por ejemplo,

```
subconjunto (Cj [1,3,2,1]) (Cj [3,1,3,2]) ~> True
subconjunto (Cj [1,3,4,1]) (Cj [3,1,3,2]) ~> False
```

```
subconjunto :: Eq a => Conj a -> Conj a -> Bool
subconjunto (Cj xs) (Cj ys) = sublista xs ys
  where sublista [] _      = True
        sublista (x:xs) ys = elem x ys &&
                               sublista xs ys
```

Los conjuntos como listas no ordenadas con duplicados

- `(subconjunto c1 c2)` se verifica si `c1` es un subconjunto de `c2`. Por ejemplo,

```
subconjunto (Cj [1,3,2,1]) (Cj [3,1,3,2]) ~> True
subconjunto (Cj [1,3,4,1]) (Cj [3,1,3,2]) ~> False
```

```
subconjunto :: Eq a => Conj a -> Conj a -> Bool
subconjunto (Cj xs) (Cj ys) = sublista xs ys
  where sublista [] _      = True
        sublista (x:xs) ys = elem x ys &&
                               sublista xs ys
```

Los conjuntos como listas no ordenadas con duplicados

- ▶ `(igualConjunto c1 c2)` se verifica si los conjuntos `c1` y `c2` son iguales. Por ejemplo,

```
igualConjunto (Cj [1,3,2,1]) (Cj [3,1,3,2])  ~>  True
igualConjunto (Cj [1,3,4,1]) (Cj [3,1,3,2])  ~>  False
```

```
igualConjunto :: Eq a => Conj a -> Conj a -> Bool
igualConjunto c1 c2 =
    subconjunto c1 c2 && subconjunto c2 c1
```

- ▶ Los conjuntos son comparables por igualdad.

```
instance Eq a => Eq (Conj a) where
    (==) = igualConjunto
```

Los conjuntos como listas no ordenadas con duplicados

- `(igualConjunto c1 c2)` se verifica si los conjuntos `c1` y `c2` son iguales. Por ejemplo,

```
igualConjunto (Cj [1,3,2,1]) (Cj [3,1,3,2])  ~>  True
igualConjunto (Cj [1,3,4,1]) (Cj [3,1,3,2])  ~>  False
```

```
igualConjunto :: Eq a => Conj a -> Conj a -> Bool
igualConjunto c1 c2 =
    subconjunto c1 c2 && subconjunto c2 c1
```

- Los conjuntos son comparables por igualdad.

```
instance Eq a => Eq (Conj a) where
    (==) = igualConjunto
```

Tema 17: El TAD de los conjuntos

1. Especificación del TAD de los conjuntos

2. Implementaciones del TAD de los conjuntos

Los conjuntos como listas no ordenadas con duplicados

Los conjuntos como listas no ordenadas sin duplicados

Los conjuntos como listas ordenadas sin duplicados

Los conjuntos de números enteros mediante números binarios

3. Comprobación de las implementaciones con QuickCheck

Los conjuntos como listas no ordenadas sin duplicados

- Cabecera del módulo.

```
module ConjuntoConListasNoOrdenadasSinDuplicados
  (Conj,
   vacio,      -- Conj a
   esVacio,   -- Conj a -> Bool
   pertenece, -- Eq a => a -> Conj a -> Bool
   inserta,   -- Eq a => a -> Conj a -> Conj a
   elimina    -- Eq a => a -> Conj a -> Conj a
  ) where
```

Los conjuntos como listas no ordenadas sin duplicados

- ▶ Los conjuntos como listas no ordenadas sin repeticiones.

```
newtype Conj a = Cj [a]
```

- ▶ Procedimiento de escritura de los conjuntos.

```
instance (Show a) => Show (Conj a) where
    showsPrec _ (Cj s) cad = showConj s cad
```

```
showConj []      cad = showString "{}" cad
showConj (x:xs) cad = showChar '{' (shows x (showl xs cad))
    where
        showl []      cad = showChar '}' cad
        showl (x:xs) cad = showChar ',' (shows x (showl xs cad))
```

Los conjuntos como listas no ordenadas sin duplicados

- ▶ Ejemplo de conjunto: `c1` es el conjunto obtenido añadiéndole al conjunto vacío los elementos 2, 5, 1, 3, 7, 5, 3, 2, 1, 9 y 0.

```
|ghci> c1
|{7,5,3,2,1,9,0}
```

```
c1 :: Conj Int
```

```
c1 = foldr inserta vacio [2,5,1,3,7,5,3,2,1,9,0]
```

- ▶ `vacio` es el conjunto vacío. Por ejemplo,

```
|ghci> vacio
|{}
```

```
vacio :: Conj a
```

```
vacio = Cj []
```

Los conjuntos como listas no ordenadas sin duplicados

- ▶ Ejemplo de conjunto: `c1` es el conjunto obtenido añadiéndole al conjunto vacío los elementos 2, 5, 1, 3, 7, 5, 3, 2, 1, 9 y 0.

```
|ghci> c1  
{7,5,3,2,1,9,0}
```

```
c1 :: Conj Int
```

```
c1 = foldr inserta vacio [2,5,1,3,7,5,3,2,1,9,0]
```

- ▶ `vacio` es el conjunto vacío. Por ejemplo,

```
|ghci> vacio  
{}
```

```
vacio :: Conj a
```

```
vacio = Cj []
```

Los conjuntos como listas no ordenadas sin duplicados

- `(esVacio c)` se verifica si `c` es el conjunto vacío. Por ejemplo,

```
esVacio c1      ~> False
esVacio vacio  ~> True
```

```
esVacio :: Conj a -> Bool
esVacio (Cj xs) = null xs
```

- `(pertenece x c)` se verifica si `x` pertenece al conjunto `c`. Por ejemplo,

```
c1          == {2,5,1,3,7,5,3,2,1,9,0}
pertenece 3 c1 == True
pertenece 4 c1 == False
```

```
pertenece :: Eq a => a -> Conj a -> Bool
pertenece x (Cj xs) = elem x xs
```

Los conjuntos como listas no ordenadas sin duplicados

- `(esVacio c)` se verifica si `c` es el conjunto vacío. Por ejemplo,

```
esVacio c1      ~> False
esVacio vacio  ~> True
```

```
esVacio :: Conj a -> Bool
esVacio (Cj xs) = null xs
```

- `(pertenece x c)` se verifica si `x` pertenece al conjunto `c`. Por ejemplo,

```
c1          == {2,5,1,3,7,5,3,2,1,9,0}
pertenece 3 c1 == True
pertenece 4 c1 == False
```

```
pertenece :: Eq a => a -> Conj a -> Bool
pertenece x (Cj xs) = elem x xs
```

Los conjuntos como listas no ordenadas sin duplicados

- `(esVacio c)` se verifica si `c` es el conjunto vacío. Por ejemplo,

```
esVacio c1      ~> False
esVacio vacio  ~> True
```

```
esVacio :: Conj a -> Bool
esVacio (Cj xs) = null xs
```

- `(pertenece x c)` se verifica si `x` pertenece al conjunto `c`. Por ejemplo,

```
c1          == {2,5,1,3,7,5,3,2,1,9,0}
pertenece 3 c1 == True
pertenece 4 c1 == False
```

```
pertenece :: Eq a => a -> Conj a -> Bool
pertenece x (Cj xs) = elem x xs
```

Los conjuntos como listas no ordenadas sin duplicados

- ▶ `(inserta x c)` es el conjunto obtenido añadiendo el elemento `x` al conjunto `c`. Por ejemplo,

```
inserta 4 c1 == {4,7,5,3,2,1,9,0}
inserta 5 c1 == {7,5,3,2,1,9,0}
```

```
inserta :: Eq a => a -> Conj a -> Conj a
inserta x s@(Cj xs) | pertenece x s = s
                   | otherwise   = Cj (x:xs)
```

- ▶ `(elimina x c)` es el conjunto obtenido eliminando el elemento `x` del conjunto `c`. Por ejemplo,

```
elimina 3 c1 == {7,5,2,1,9,0}
```

```
elimina :: Eq a => a -> Conj a -> Conj a
elimina x (Cj ys) = Cj [y | y <- ys, y /= x]
```

Los conjuntos como listas no ordenadas sin duplicados

- ▶ `(inserta x c)` es el conjunto obtenido añadiendo el elemento `x` al conjunto `c`. Por ejemplo,

```
inserta 4 c1 == {4,7,5,3,2,1,9,0}
inserta 5 c1 == {7,5,3,2,1,9,0}
```

```
inserta :: Eq a => a -> Conj a -> Conj a
inserta x s@(Cj xs) | pertenece x s = s
                   | otherwise   = Cj (x:xs)
```

- ▶ `(elimina x c)` es el conjunto obtenido eliminando el elemento `x` del conjunto `c`. Por ejemplo,

```
elimina 3 c1 == {7,5,2,1,9,0}
```

```
elimina :: Eq a => a -> Conj a -> Conj a
elimina x (Cj ys) = Cj [y | y <- ys, y /= x]
```

Los conjuntos como listas no ordenadas sin duplicados

- `(inserta x c)` es el conjunto obtenido añadiendo el elemento `x` al conjunto `c`. Por ejemplo,

```
inserta 4 c1 == {4,7,5,3,2,1,9,0}
inserta 5 c1 == {7,5,3,2,1,9,0}
```

```
inserta :: Eq a => a -> Conj a -> Conj a
inserta x s@(Cj xs) | pertenece x s = s
                   | otherwise   = Cj (x:xs)
```

- `(elimina x c)` es el conjunto obtenido eliminando el elemento `x` del conjunto `c`. Por ejemplo,

```
elimina 3 c1 == {7,5,2,1,9,0}
```

```
elimina :: Eq a => a -> Conj a -> Conj a
elimina x (Cj ys) = Cj [y | y <- ys, y /= x]
```

Los conjuntos como listas no ordenadas sin duplicados

- `(subconjunto c1 c2)` se verifica si `c1` es un subconjunto de `c2`. Por ejemplo,

```
subconjunto (Cj [1,3,2]) (Cj [3,1,2])    ~> True
subconjunto (Cj [1,3,4,1]) (Cj [1,3,2]) ~> False
```

```
subconjunto :: Eq a => Conj a -> Conj a -> Bool
subconjunto (Cj xs) (Cj ys) = sublista xs ys
  where sublista [] _      = True
        sublista (x:xs) ys = elem x ys &&
                               sublista xs ys
```

Los conjuntos como listas no ordenadas sin duplicados

- ▶ `(subconjunto c1 c2)` se verifica si `c1` es un subconjunto de `c2`. Por ejemplo,

```
subconjunto (Cj [1,3,2]) (Cj [3,1,2])    ~> True
subconjunto (Cj [1,3,4,1]) (Cj [1,3,2]) ~> False
```

```
subconjunto :: Eq a => Conj a -> Conj a -> Bool
subconjunto (Cj xs) (Cj ys) = sublista xs ys
  where sublista [] _      = True
        sublista (x:xs) ys = elem x ys &&
                               sublista xs ys
```

Los conjuntos como listas no ordenadas sin duplicados

- ▶ `(igualConjunto c1 c2)` se verifica si los conjuntos `c1` y `c2` son iguales. Por ejemplo,

```
igualConjunto (Cj [3,2,1]) (Cj [1,3,2])  ~>  True
igualConjunto (Cj [1,3,4]) (Cj [1,3,2])  ~>  False
```

```
igualConjunto :: Eq a => Conj a -> Conj a -> Bool
igualConjunto c1 c2 =
    subconjunto c1 c2 && subconjunto c2 c1
```

- ▶ Los conjuntos son comparables por igualdad.

```
instance Eq a => Eq (Conj a) where
    (==) = igualConjunto
```

Los conjuntos como listas no ordenadas sin duplicados

- `(igualConjunto c1 c2)` se verifica si los conjuntos `c1` y `c2` son iguales. Por ejemplo,

```
igualConjunto (Cj [3,2,1]) (Cj [1,3,2])  ~> True
igualConjunto (Cj [1,3,4]) (Cj [1,3,2])  ~> False
```

```
igualConjunto :: Eq a => Conj a -> Conj a -> Bool
igualConjunto c1 c2 =
    subconjunto c1 c2 && subconjunto c2 c1
```

- Los conjuntos son comparables por igualdad.

```
instance Eq a => Eq (Conj a) where
    (==) = igualConjunto
```

Tema 17: El TAD de los conjuntos

1. Especificación del TAD de los conjuntos

2. Implementaciones del TAD de los conjuntos

Los conjuntos como listas no ordenadas con duplicados

Los conjuntos como listas no ordenadas sin duplicados

Los conjuntos como listas ordenadas sin duplicados

Los conjuntos de números enteros mediante números binarios

3. Comprobación de las implementaciones con QuickCheck

Los conjuntos como listas ordenadas sin duplicados

► Cabecera del módulo

```
module ConjuntoConListasOrdenadasSinDuplicados
  (Conj,
   vacio,      -- Conj a
   esVacio,   -- Conj a -> Bool
   pertenece, -- Ord a => a -> Conj a -> Bool
   inserta,   -- Ord a => a -> Conj a -> Conj a
   elimina    -- Ord a => a -> Conj a -> Conj a
  ) where
```

► Los conjuntos como listas ordenadas sin repeticiones.

```
newtype Conj a = Cj [a]
  deriving Eq
```

Los conjuntos como listas ordenadas sin duplicados

- Procedimiento de escritura de los conjuntos.

```
instance (Show a) => Show (Conj a) where
    showsPrec _ (Cj s) cad = showConj s cad

showConj []      cad = showString "{}" cad
showConj (x:xs) cad = showChar '{' (shows x (showl xs cad))
                    where showl []      cad = showChar '}' cad
                          showl (x:xs) cad = showChar ',' (shows x (showl xs cad))
```

Los conjuntos como listas ordenadas sin duplicados

- ▶ Ejemplo de conjunto: `c1` es el conjunto obtenido añadiéndole al conjunto vacío los elementos 2, 5, 1, 3, 7, 5, 3, 2, 1, 9 y 0.

```
|ghci> c1  
{0,1,2,3,5,7,9}
```

```
c1 :: Conj Int
```

```
c1 = foldr inserta vacio [2,5,1,3,7,5,3,2,1,9,0]
```

- ▶ `vacio` es el conjunto vacío. Por ejemplo,

```
|ghci> vacio  
{}
```

```
vacio :: Conj a
```

```
vacio = Cj []
```

Los conjuntos como listas ordenadas sin duplicados

- ▶ Ejemplo de conjunto: `c1` es el conjunto obtenido añadiéndole al conjunto vacío los elementos 2, 5, 1, 3, 7, 5, 3, 2, 1, 9 y 0.

```
| ghci> c1
| {0,1,2,3,5,7,9}
```

```
c1 :: Conj Int
```

```
c1 = foldr inserta vacio [2,5,1,3,7,5,3,2,1,9,0]
```

- ▶ `vacio` es el conjunto vacío. Por ejemplo,

```
| ghci> vacio
| {}
```

```
vacio :: Conj a
```

```
vacio = Cj []
```

Los conjuntos como listas ordenadas sin duplicados

- `(esVacio c)` se verifica si `c` es el conjunto vacío. Por ejemplo,

```
esVacio c1      ~> False
esVacio vacio  ~> True
```

```
esVacio :: Conj a -> Bool
esVacio (Cj xs) = null xs
```

- `(pertenece x c)` se verifica si `x` pertenece al conjunto `c`. Por ejemplo,

```
c1          == {0,1,2,3,5,7,9}
pertenece 3 c1 == True
pertenece 4 c1 == False
```

```
pertenece :: Ord a => a -> Conj a -> Bool
pertenece x (Cj ys) = elem x (takeWhile (<= x) ys)
```

Los conjuntos como listas ordenadas sin duplicados

- `(esVacio c)` se verifica si `c` es el conjunto vacío. Por ejemplo,

```
esVacio c1      ~> False
esVacio vacio  ~> True
```

```
esVacio :: Conj a -> Bool
esVacio (Cj xs) = null xs
```

- `(pertenece x c)` se verifica si `x` pertenece al conjunto `c`. Por ejemplo,

```
c1          == {0,1,2,3,5,7,9}
pertenece 3 c1 == True
pertenece 4 c1 == False
```

```
pertenece :: Ord a => a -> Conj a -> Bool
pertenece x (Cj ys) = elem x (takeWhile (<= x) ys)
```

Los conjuntos como listas ordenadas sin duplicados

- `(esVacio c)` se verifica si `c` es el conjunto vacío. Por ejemplo,

```
esVacio c1      ~> False
esVacio vacio  ~> True
```

```
esVacio :: Conj a -> Bool
esVacio (Cj xs) = null xs
```

- `(pertenece x c)` se verifica si `x` pertenece al conjunto `c`. Por ejemplo,

```
c1          == {0,1,2,3,5,7,9}
pertenece 3 c1 == True
pertenece 4 c1 == False
```

```
pertenece :: Ord a => a -> Conj a -> Bool
pertenece x (Cj ys) = elem x (takeWhile (<= x) ys)
```

Los conjuntos como listas ordenadas sin duplicados

- `(inserta x c)` es el conjunto obtenido añadiendo el elemento `x` al conjunto `c`. Por ejemplo,

```

c1           == {0,1,2,3,5,7,9}
inserta 5 c1 == {0,1,2,3,5,7,9}
inserta 4 c1 == {0,1,2,3,4,5,7,9}

```

```

inserta :: Ord a => a -> Conj a -> Conj a
inserta x (Cj s) = Cj (agrega x s) where
    agrega x []           = [x]
    agrega x s@(y:ys) | x > y     = y : (agrega x ys)
                      | x < y     = x : s
                      | otherwise = s

```

Los conjuntos como listas ordenadas sin duplicados

- `(inserta x c)` es el conjunto obtenido añadiendo el elemento `x` al conjunto `c`. Por ejemplo,

```

c1           == {0,1,2,3,5,7,9}
inserta 5 c1 == {0,1,2,3,5,7,9}
inserta 4 c1 == {0,1,2,3,4,5,7,9}

```

```

inserta :: Ord a => a -> Conj a -> Conj a
inserta x (Cj s) = Cj (agrega x s) where
    agrega x []           = [x]
    agrega x s@(y:ys) | x > y     = y : (agrega x ys)
                      | x < y     = x : s
                      | otherwise = s

```

Los conjuntos como listas ordenadas sin duplicados

- `(elimina x c)` es el conjunto obtenido eliminando el elemento `x` del conjunto `c`. Por ejemplo,

```
c1 == {0,1,2,3,5,7,9}
elimina 3 c1 == {0,1,2,5,7,9}
```

```
elimina :: Ord a => a -> Conj a -> Conj a
elimina x (Cj s) = Cj (elimina x s) where
  elimina x [] = []
  elimina x s@(y:ys) | x > y = y : elimina x ys
                    | x < y = s
                    | otherwise = ys
```

Los conjuntos como listas ordenadas sin duplicados

- `(elimina x c)` es el conjunto obtenido eliminando el elemento `x` del conjunto `c`. Por ejemplo,

```
c1 == {0,1,2,3,5,7,9}
elimina 3 c1 == {0,1,2,5,7,9}
```

```
elimina :: Ord a => a -> Conj a -> Conj a
elimina x (Cj s) = Cj (elimina x s) where
    elimina x [] = []
    elimina x s@(y:ys) | x > y = y : elimina x ys
                       | x < y = s
                       | otherwise = ys
```

Tema 17: El TAD de los conjuntos

1. Especificación del TAD de los conjuntos

2. Implementaciones del TAD de los conjuntos

Los conjuntos como listas no ordenadas con duplicados

Los conjuntos como listas no ordenadas sin duplicados

Los conjuntos como listas ordenadas sin duplicados

Los conjuntos de números enteros mediante números binarios

3. Comprobación de las implementaciones con QuickCheck

Los conjuntos de enteros mediante números binarios

- ▶ Los conjuntos que sólo contienen números (de tipo Int) entre 0 y $n - 1$, se pueden representar como números binarios con n bits donde el bit i ($0 \leq i < n$) es 1 si el número i pertenece al conjunto. Por ejemplo,

		43210	
$\{3,4\}$	en binario es	11000	en decimal es 24
$\{1,2,3,4\}$	en binario es	11110	en decimal es 30
$\{1,2,4\}$	en binario es	10110	en decimal es 22

Los conjuntos de enteros mediante números binarios

► Cabecera del módulo

```
module ConjuntoConNumerosBinarios
  (Conj,
   vacio,      -- Conj
   esVacio,   -- Conj -> Bool
   pertenece, -- Int -> Conj -> Bool
   inserta,   -- Int -> Conj -> Conj
   elimina    -- Int -> Conj -> Conj
  ) where
```

► Los conjuntos de números enteros como números binarios.

```
newtype Conj = Cj Int deriving Eq
```

Los conjuntos de enteros mediante números binarios

- `(conj2Lista c)` es la lista de los elementos del conjunto `c`. Por ejemplo,

```

conj2Lista (Cj 24)  ~>  [3,4]
conj2Lista (Cj 30)  ~>  [1,2,3,4]
conj2Lista (Cj 22)  ~>  [1,2,4]

```

```

conj2Lista (Cj s) = c2l s 0
  where
    c2l 0 _      = []
    c2l n i | odd n      = i : c2l (n `div` 2) (i+1)
            | otherwise = c2l (n `div` 2) (i+1)

```

Los conjuntos de enteros mediante números binarios

- `(conj2Lista c)` es la lista de los elementos del conjunto `c`. Por ejemplo,

```

conj2Lista (Cj 24)  ~>  [3,4]
conj2Lista (Cj 30)  ~>  [1,2,3,4]
conj2Lista (Cj 22)  ~>  [1,2,4]

```

```

conj2Lista (Cj s) = c2l s 0
  where
    c2l 0 _           = []
    c2l n i | odd n   = i : c2l (n `div` 2) (i+1)
            | otherwise = c2l (n `div` 2) (i+1)

```

Los conjuntos de enteros mediante números binarios

- Procedimiento de escritura de conjuntos.

```
instance Show Conj where
    showsPrec _ s cad = showConj (conj2Lista s) cad

showConj []      cad = showString "{}" cad
showConj (x:xs) cad =
    showChar '{' (shows x (showl xs cad))
  where
    showl []      cad = showChar '}' cad
    showl (x:xs) cad = showChar ',' (shows x (showl xs cad))
```

Los conjuntos de enteros mediante números binarios

- `maxConj` es el máximo número que puede pertenecer al conjunto. Depende de la implementación de Haskell. Por ejemplo,

```
| maxConj  ~> 29
```

```
maxConj :: Int
maxConj =
    truncate (logBase 2 (fromIntegral maxInt)) - 1
    where maxInt = maxBound :: Int
```

- Ejemplo de conjunto: `c1` es el conjunto obtenido añadiéndole al conjunto vacío los elementos 2, 5, 1, 3, 7, 5, 3, 2, 1, 9 y 0.

```
| ghci> c1
| {0,1,2,3,5,7,9}
```

```
c1 :: Conj
c1 = foldr inserta vacio [2,5,1,3,7,5,3,2,1,9,0]
```

Los conjuntos de enteros mediante números binarios

- ▶ `vacío` es el conjunto vacío. Por ejemplo,

```
| ghci> vacío  
| {}
```

```
vacío :: Conj  
vacío = Cj 0
```

- ▶ `(esVacio c)` se verifica si `c` es el conjunto vacío. Por ejemplo,

```
| esVacio c1      ~> False  
| esVacio vacío   ~> True
```

```
esVacio :: Conj -> Bool  
esVacio (Cj n) = n == 0
```

Los conjuntos de enteros mediante números binarios

- ▶ `vacio` es el conjunto vacío. Por ejemplo,

```
| ghci> vacio  
| {}
```

```
vacio :: Conj
```

```
vacio = Cj 0
```

- ▶ `(esVacio c)` se verifica si `c` es el conjunto vacío. Por ejemplo,

```
| esVacio c1      ~> False  
| esVacio vacio  ~> True
```

```
esVacio :: Conj -> Bool
```

```
esVacio (Cj n) = n == 0
```

Los conjuntos de enteros mediante números binarios

- ▶ `vacío` es el conjunto vacío. Por ejemplo,

```
| ghci> vacío  
| {}
```

```
vacío :: Conj
```

```
vacío = Cj 0
```

- ▶ `(esVacio c)` se verifica si `c` es el conjunto vacío. Por ejemplo,

```
| esVacio c1      ~> False  
| esVacio vacío   ~> True
```

```
esVacio :: Conj -> Bool
```

```
esVacio (Cj n) = n == 0
```

Los conjuntos de enteros mediante números binarios

- `(pertenece x c)` se verifica si `x` pertenece al conjunto `c`. Por ejemplo,

```
c1 == {0,1,2,3,5,7,9}
pertenece 3 c1 == True
pertenece 4 c1 == False
```

```
pertenece :: Int -> Conj -> Bool
pertenece i (Cj s)
  | (i>=0) && (i<=maxConj) = odd (s `div` (2i))
  | otherwise
    = error ("pertenece: elemento ilegal =" ++ show i)
```

Los conjuntos de enteros mediante números binarios

- `(pertenece x c)` se verifica si `x` pertenece al conjunto `c`. Por ejemplo,

```

c1 == {0,1,2,3,5,7,9}
pertenece 3 c1 == True
pertenece 4 c1 == False

```

```

pertenece :: Int -> Conj -> Bool
pertenece i (Cj s)
  | (i>=0) && (i<=maxConj) = odd (s `div` (2^i))
  | otherwise
    = error ("pertenece: elemento ilegal =" ++ show i)

```

Los conjuntos de enteros mediante números binarios

- `(inserta x c)` es el conjunto obtenido añadiendo el elemento `x` al conjunto `c`. Por ejemplo,

```

c1           == {0,1,2,3,5,7,9}
inserta 5 c1 == {0,1,2,3,5,7,9}
inserta 4 c1 == {0,1,2,3,4,5,7,9}

```

```

inserta i (Cj s)
  | (i>=0) && (i<=maxConj) = Cj (d'*e+m)
  | otherwise
    = error ("inserta: elemento ilegal =" ++ show i)
where (d,m) = divMod s e
      e      = 2^i
      d'     = if odd d then d else d+1

```

Los conjuntos de enteros mediante números binarios

- `(inserta x c)` es el conjunto obtenido añadiendo el elemento `x` al conjunto `c`. Por ejemplo,

```

c1                == {0,1,2,3,5,7,9}
inserta 5 c1      == {0,1,2,3,5,7,9}
inserta 4 c1      == {0,1,2,3,4,5,7,9}

```

```

inserta i (Cj s)
  | (i>=0) && (i<=maxConj) = Cj (d'*e+m)
  | otherwise
    = error ("inserta: elemento ilegal =" ++ show i)
where (d,m) = divMod s e
      e      = 2^i
      d'     = if odd d then d else d+1

```

Los conjuntos de enteros mediante números binarios

- `(elimina x c)` es el conjunto obtenido eliminando el elemento `x` del conjunto `c`. Por ejemplo,

```
c1 == {0,1,2,3,5,7,9}
elimina 3 c1 == {0,1,2,5,7,9}
```

```
elimina i (Cj s)
  | (i>=0) && (i<=maxConj) = Cj (d'*e+m)
  | otherwise
    = error ("elimina: elemento ilegal =" ++ show i)
where (d,m) = divMod s e
      e      = 2^i
      d'     = if odd d then d-1 else d
```

Los conjuntos de enteros mediante números binarios

- `(elimina x c)` es el conjunto obtenido eliminando el elemento `x` del conjunto `c`. Por ejemplo,

```
c1 == {0,1,2,3,5,7,9}
elimina 3 c1 == {0,1,2,5,7,9}
```

```
elimina i (Cj s)
  | (i>=0) && (i<=maxConj) = Cj (d'*e+m)
  | otherwise
    = error ("elimina: elemento ilegal =" ++ show i)
where (d,m) = divMod s e
      e      = 2^i
      d'     = if odd d then d-1 else d
```

Tema 17: El TAD de los conjuntos

1. Especificación del TAD de los conjuntos
2. Implementaciones del TAD de los conjuntos
3. Comprobación de las implementaciones con QuickCheck
 - Librerías auxiliares
 - Generador de conjuntos
 - Especificación de las propiedades de los conjuntos
 - Comprobación de las propiedades

Comprobación de las propiedades de los conjuntos

- ▶ Importación de la implementación de los conjuntos que se desea verificar.

```
import ConjuntoConListasNoOrdenadasConDuplicados
-- import ConjuntoConListasNoOrdenadasSinDuplicados
-- import ConjuntoConListasOrdenadasSinDuplicados
```

- ▶ Importación de las librerías de comprobación

```
import Test.QuickCheck
import Test.Framework
import Test.Framework.Providers.QuickCheck2
```

Tema 17: El TAD de los conjuntos

1. Especificación del TAD de los conjuntos
2. Implementaciones del TAD de los conjuntos
3. Comprobación de las implementaciones con QuickCheck
 - Librerías auxiliares
 - Generador de conjuntos**
 - Especificación de las propiedades de los conjuntos
 - Comprobación de las propiedades

Generador de conjuntos

- ▶ `genConjunto` es un generador de conjuntos. Por ejemplo,

```
ghci> sample genConjunto  
{3,-2,-2,-3,-2,4}  
{-8,0,4,6,-5,-2}  
{}
```

```
genConjunto :: Gen (Conj Int)  
genConjunto = do xs <- listOf arbitrary  
                return (foldr inserta vacio xs)
```

```
instance Arbitrary (Conj Int) where  
    arbitrary = genConjunto
```

Tema 17: El TAD de los conjuntos

1. Especificación del TAD de los conjuntos
2. Implementaciones del TAD de los conjuntos
3. Comprobación de las implementaciones con QuickCheck
 - Librerías auxiliares
 - Generador de conjuntos
 - Especificación de las propiedades de los conjuntos
 - Comprobación de las propiedades

Especificación de las propiedades de los conjuntos

- ▶ El número de veces que se añade un elemento a un conjunto no importa.

```
prop_independencia_repeticiones :: Int -> Conj Int  
                                -> Bool
```

```
prop_independencia_repeticiones x c =  
    inserta x (inserta x c) == inserta x c
```

- ▶ El orden en que se añadan los elementos a un conjunto no importa.

```
prop_independencia_del_orden :: Int -> Int -> Conj Int  
                              -> Bool
```

```
prop_independencia_del_orden x y c =  
    inserta x (inserta y c) == inserta y (inserta x c)
```

Especificación de las propiedades de los conjuntos

- ▶ El número de veces que se añade un elemento a un conjunto no importa.

```
prop_independencia_repeticiones :: Int -> Conj Int
                                   -> Bool
```

```
prop_independencia_repeticiones x c =
    inserta x (inserta x c) == inserta x c
```

- ▶ El orden en que se añadan los elementos a un conjunto no importa.

```
prop_independencia_del_orden :: Int -> Int -> Conj Int
                               -> Bool
```

```
prop_independencia_del_orden x y c =
    inserta x (inserta y c) == inserta y (inserta x c)
```

Especificación de las propiedades de los conjuntos

- ▶ El número de veces que se añade un elemento a un conjunto no importa.

```
prop_independencia_repeticiones :: Int -> Conj Int
                                   -> Bool
```

```
prop_independencia_repeticiones x c =
    inserta x (inserta x c) == inserta x c
```

- ▶ El orden en que se añadan los elementos a un conjunto no importa.

```
prop_independencia_del_orden :: Int -> Int -> Conj Int
                                   -> Bool
```

```
prop_independencia_del_orden x y c =
    inserta x (inserta y c) == inserta y (inserta x c)
```

Especificación de las propiedades de los conjuntos

- ▶ El conjunto vacío no tiene elementos.

```
prop_vacio_no_elementos :: Int -> Bool
prop_vacio_no_elementos x =
    not (pertenece x vacio)
```

- ▶ Un elemento pertenece al conjunto obtenido añadiendo x al conjunto c `sys` es igual a x o pertenece a c .

```
prop_pertenece_inserta :: Int -> Int -> Conj Int -> Bool
prop_pertenece_inserta x y c =
    pertenece y (inserta x c) == (x==y) || pertenece y c
```

Especificación de las propiedades de los conjuntos

- ▶ El conjunto vacío no tiene elementos.

```
prop_vacio_no_elementos :: Int -> Bool
prop_vacio_no_elementos x =
    not (pertenece x vacio)
```

- ▶ Un elemento pertenece al conjunto obtenido añadiendo x al conjunto c syss es igual a x o pertenece a c.

```
prop_pertenece_inserta :: Int -> Int -> Conj Int -> Bool
prop_pertenece_inserta x y c =
    pertenece y (inserta x c) == (x==y) || pertenece y c
```

Especificación de las propiedades de los conjuntos

- ▶ El conjunto vacío no tiene elementos.

```
prop_vacio_no_elementos :: Int -> Bool
prop_vacio_no_elementos x =
    not (pertenece x vacio)
```

- ▶ Un elemento pertenece al conjunto obtenido añadiendo x al conjunto c syss es igual a x o pertenece a c.

```
prop_pertenece_inserta :: Int -> Int -> Conj Int -> Bool
prop_pertenece_inserta x y c =
    pertenece y (inserta x c) == (x==y) || pertenece y c
```

Especificación de las propiedades de los conjuntos

- ▶ Al eliminar cualquier elemento del conjunto vacío se obtiene el conjunto vacío.

```
prop_elimina_vacio :: Int -> Bool
prop_elimina_vacio x =
    elimina x vacio == vacio
```

- ▶ El resultado de eliminar x en el conjunto obtenido añadiéndole x al conjunto c es c menos x , si x e y son iguales y es el conjunto obtenido añadiéndole y a c menos x , en caso contrario.

```
prop_elimina_inserta :: Int -> Int -> Conj Int -> Bool
prop_elimina_inserta x y c =
    elimina x (inserta y c)
    == if x == y then elimina x c
       else inserta y (elimina x c)
```

Especificación de las propiedades de los conjuntos

- ▶ Al eliminar cualquier elemento del conjunto vacío se obtiene el conjunto vacío.

```
prop_elimina_vacio :: Int -> Bool
prop_elimina_vacio x =
    elimina x vacio == vacio
```

- ▶ El resultado de eliminar x en el conjunto obtenido añadiéndole x al conjunto c es c menos x , si x e y son iguales y es el conjunto obtenido añadiéndole y a c menos x , en caso contrario.

```
prop_elimina_inserta :: Int -> Int -> Conj Int -> Bool
prop_elimina_inserta x y c =
    elimina x (inserta y c)
    == if x == y then elimina x c
       else inserta y (elimina x c)
```

Especificación de las propiedades de los conjuntos

- ▶ Al eliminar cualquier elemento del conjunto vacío se obtiene el conjunto vacío.

```
prop_elimina_vacio :: Int -> Bool
prop_elimina_vacio x =
    elimina x vacio == vacio
```

- ▶ El resultado de eliminar x en el conjunto obtenido añadiéndole x al conjunto c es c menos x , si x e y son iguales y es el conjunto obtenido añadiéndole y a c menos x , en caso contrario.

```
prop_elimina_inserta :: Int -> Int -> Conj Int -> Bool
prop_elimina_inserta x y c =
    elimina x (inserta y c)
    == if x == y then elimina x c
       else inserta y (elimina x c)
```

Especificación de las propiedades de los conjuntos

- ▶ vacío es vacío.

```
prop_vacio_es_vacio :: Bool
prop_vacio_es_vacio =
    esVacio (vacio :: Conj Int)
```

- ▶ Los conjuntos construidos con inserta no son vacío.

```
prop_inserta_es_no_vacio :: Int -> Conj Int -> Bool
prop_inserta_es_no_vacio x c =
    not (esVacio (inserta x c))
```

Especificación de las propiedades de los conjuntos

- ▶ vacío es vacío.

```
prop_vacio_es_vacio :: Bool
prop_vacio_es_vacio =
    esVacio (vacio :: Conj Int)
```

- ▶ Los conjuntos construidos con `inserta` no son vacío.

```
prop_inserta_es_no_vacio :: Int -> Conj Int -> Bool
prop_inserta_es_no_vacio x c =
    not (esVacio (inserta x c))
```

Especificación de las propiedades de los conjuntos

- ▶ vacío es vacío.

```
prop_vacio_es_vacio :: Bool
prop_vacio_es_vacio =
    esVacio (vacio :: Conj Int)
```

- ▶ Los conjuntos construidos con `inserta` no son vacío.

```
prop_inserta_es_no_vacio :: Int -> Conj Int -> Bool
prop_inserta_es_no_vacio x c =
    not (esVacio (inserta x c))
```

Tema 17: El TAD de los conjuntos

1. Especificación del TAD de los conjuntos
2. Implementaciones del TAD de los conjuntos
3. Comprobación de las implementaciones con QuickCheck
 - Librerías auxiliares
 - Generador de conjuntos
 - Especificación de las propiedades de los conjuntos
 - Comprobación de las propiedades

Definición del procedimiento de comprobación

- `compruebaPropiedades` comprueba todas las propiedades con la plataforma de verificación.

```
compruebaPropiedades =  
  defaultMain  
    [testGroup "Propiedades del TAD conjunto:"  
      [testProperty "P1" prop_vacio_es_vacio,  
        testProperty "P2" prop_inserta_es_no_vacio,  
        testProperty "P3" prop_independencia_repeticiones,  
        testProperty "P4" prop_independencia_del_orden,  
        testProperty "P5" prop_vacio_no_elementos,  
        testProperty "P6" prop_pertenece_inserta,  
        testProperty "P7" prop_elimina_vacio,  
        testProperty "P8" prop_elimina_inserta]]
```

Comprobación de las propiedades de los conjuntos

```
ghci> compruebaPropiedades
Propiedades del TAD conjunto:
P1: [OK, passed 100 tests]
P2: [OK, passed 100 tests]
P3: [OK, passed 100 tests]
P4: [OK, passed 100 tests]
P5: [OK, passed 100 tests]
P6: [OK, passed 100 tests]
P7: [OK, passed 100 tests]
P8: [OK, passed 100 tests]
```

	Properties	Total
Passed	8	8
Failed	0	0
Total	8	8