Tema 21: El TAD de los polinomios

Informática (2018–19)

José A. Alonso Jiménez

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

Tema 21: El TAD de los polinomios

Tema 21: El TAD de los polinomios

Signatura del TAD de los polinomios

Signatura:

```
polCero   :: Polinomio a
esPolCero :: Polinomio a -> Bool
consPol   :: (Num a, Eq a) => Int -> a -> Polinomio a -> Poli
grado     :: Polinomio a -> Int
coefLider :: Num a => Polinomio a -> a
restoPol  :: (Num a, Eq a) => Polinomio a -> Polinomio a
```

Descripción de las operaciones:

- `polCero` es el polinomio cero.
- `(esPolCero p)` se verifica si `p` es el polinomio cero.
- `(consPol n b p)` es el polinomio $bx^n + p$.
- `(grado p)` es el grado del polinomio `p`.
- `(coefLider p)` es el coeficiente líder del polinomio `p`.
- `(restoPol p)` es el resto del polinomio `p`.

Ejemplos de polinomios

Ejemplos de polinomios que se usarán en lo sucesivo.

- Definición:

```
ejPol1, ejPol2, ejPol3, ejTerm:: Polinomio Int
ejPol1 = consPol 4 3 (consPol 2 (-5) (consPol 0 3 polCero))
ejPol2 = consPol 5 1 (consPol 2 5 (consPol 1 4 polCero))
ejPol3 = consPol 4 6 (consPol 1 2 polCero)
ejTerm = consPol 1 4 polCero
```

- Evaluación:

```
ejPol1  ⇝  3*x^4 + -5*x^2 + 3
ejPol2  ⇝  x^5 + 5*x^2 + 4*x
ejPol3  ⇝  6*x^4 + 2*x
ejTerm  ⇝  4*x
```

Tema 21: El TAD de los polinomios

Propiedades del TAD de los polinomios

1. `esPolCero polCero`
2. `n > grado p && b /= 0 ==>`
   `not (esPolCero (consPol n b p))`
3. `consPol (grado p) (coefLider p) (restoPol p) == p`
4. `n > grado p && b /= 0 ==>`
   `grado (consPol n b p) == n`
5. `n > grado p && b /= 0 ==>`
   `coefLider (consPol n b p) == b`
6. `n > grado p && b /= 0 ==>`
   `restoPol (consPol n b p) == p`

Tema 21: El TAD de los polinomios

Los polinomios como tipo de dato algebraico

Cabecera del módulo:

```
module PolRepTDA
  ( Polinomio,
    polCero,   -- Polinomio a
    esPolCero, -- Polinomio a -> Bool
    consPol,   -- (Num a, Eq a) => Int -> a -> Polinomio a
               --                  -> Polinomio a
    grado,     -- Polinomio a -> Int
    coefLider, -- Num a => Polinomio a -> a
    restoPol   -- (Num a, Eq a) => Polinomio a -> Polinomio a
  ) where
```

Los polinomios como tipo de dato algebraico

- Representamos un polinomio mediante los constructores `ConsPol` y `PolCero`.
- Por ejemplo, el polinomio

```
6x^4 -5x^2 + 4x -7
```

se representa por

```
ConsPol 4 6
  (ConsPol 2 (-5)
     (ConsPol 1 4
        (ConsPol 0 (-7) PolCero)))
```

- El tipo de los polinomios.

```
data Polinomio a = PolCero
                 | ConsPol Int a (Polinomio a)
                 deriving Eq
```

Los polinomios como tipo de dato algebraico

Procedimiento de escritura de los polinomios.

```
instance (Num a, Show a, Eq a) => Show (Polinomio a) where
    show PolCero                = "0"
    show (ConsPol 0 b PolCero) = show b
    show (ConsPol 0 b p)       = concat [show b," + ",show p]
    show (ConsPol 1 b PolCero) = concat [show b,"*x"]
    show (ConsPol 1 b p)       = concat [show b,"*x + ",show p]
    show (ConsPol n 1 PolCero) = concat ["x^",show n]
    show (ConsPol n b PolCero) = concat [show b,"*x^",show n]
    show (ConsPol n 1 p)       = concat ["x^",show n," + ",show p]
    show (ConsPol n b p)       = concat [show b,"*x^",show n," + ",show p]
```

Los polinomios como tipo de dato algebraico

- `polCero` es el polinomio cero. Por ejemplo,

```
ghci> polCero
0
```

```
polCero :: Polinomio a
polCero = PolCero
```

- `(esPolCero p)` se verifica si `p` es el polinomio cero. Por ejemplo,

```
esPolCero polCero  ⇝  True
esPolCero ejPol1   ⇝  False
```

```
esPolCero :: Polinomio a -> Bool
esPolCero PolCero = True
esPolCero _       = False
```

Los polinomios como tipo de dato algebraico

- `polCero` es el polinomio cero. Por ejemplo,

```
ghci> polCero
0
```

```
polCero :: Polinomio a
polCero = PolCero
```

- `(esPolCero p)` se verifica si `p` es el polinomio cero. Por ejemplo,

```
esPolCero polCero  ⇝  True
esPolCero ejPol1   ⇝  False
```

```
esPolCero :: Polinomio a -> Bool
esPolCero PolCero = True
esPolCero _       = False
```

Los polinomios como tipo de dato algebraico

- `polCero` es el polinomio cero. Por ejemplo,

```
ghci> polCero
0
```

```
polCero :: Polinomio a
polCero = PolCero
```

- `(esPolCero p)` se verifica si `p` es el polinomio cero. Por ejemplo,

```
esPolCero polCero  ⇝  True
esPolCero ejPol1   ⇝  False
```

```
esPolCero :: Polinomio a -> Bool
esPolCero PolCero = True
esPolCero _       = False
```

Los polinomios como tipo de dato algebraico

- `(consPol n b p)` es el polinomio $bx^n + p$. Por ejemplo,

```
ejPol2                  ⇝  x^5 + 5*x^2 + 4*x
consPol 3 0 ejPol2      ⇝  x^5 + 5*x^2 + 4*x
consPol 3 2 polCero     ⇝  2*x^3
consPol 6 7 ejPol2      ⇝  7*x^6 + x^5 + 5*x^2 + 4*x
consPol 4 7 ejPol2      ⇝  x^5 + 7*x^4 + 5*x^2 + 4*x
consPol 5 7 ejPol2      ⇝  8*x^5 + 5*x^2 + 4*x
```

```
consPol :: (Num a, Eq a) => Int -> a -> Polinomio a -> Polinomio a
consPol _ 0 p = p
consPol n b PolCero = ConsPol n b PolCero
consPol n b (ConsPol m c p)
    | n > m      = ConsPol n b (ConsPol m c p)
    | n < m      = ConsPol m c (consPol n b p)
    | b+c == 0   = p
    | otherwise  = ConsPol n (b+c) p
```

Los polinomios como tipo de dato algebraico

- `(consPol n b p)` es el polinomio $bx^n + p$. Por ejemplo,

```
ejPol2                 ⇝  x^5 + 5*x^2 + 4*x
consPol 3 0 ejPol2     ⇝  x^5 + 5*x^2 + 4*x
consPol 3 2 polCero    ⇝  2*x^3
consPol 6 7 ejPol2     ⇝  7*x^6 + x^5 + 5*x^2 + 4*x
consPol 4 7 ejPol2     ⇝  x^5 + 7*x^4 + 5*x^2 + 4*x
consPol 5 7 ejPol2     ⇝  8*x^5 + 5*x^2 + 4*x
```

```
consPol :: (Num a, Eq a) => Int -> a -> Polinomio a -> Polinomio a
consPol _ 0 p = p
consPol n b PolCero = ConsPol n b PolCero
consPol n b (ConsPol m c p)
    | n > m      = ConsPol n b (ConsPol m c p)
    | n < m      = ConsPol m c (consPol n b p)
    | b+c == 0   = p
    | otherwise  = ConsPol n (b+c) p
```

Los polinomios como tipo de dato algebraico

- `(grado p)` es el grado del polinomio p. Por ejemplo,

```
ejPol3          ⇝  6*x^4 + 2*x
grado ejPol3    ⇝  4
```

```
grado:: Polinomio a -> Int
grado PolCero          = 0
grado (ConsPol n _ _) = n
```

- `(coefLider p)` es el coeficiente líder del polinomio p. Por ejemplo,

```
coefLider ejPol3  ⇝  6
```

```
coefLider:: Num t => Polinomio t -> t
coefLider PolCero          = 0
coefLider (ConsPol _ b _) = b
```

Los polinomios como tipo de dato algebraico

- `(grado p)` es el grado del polinomio p. Por ejemplo,

```
ejPol3          ⇝  6*x^4 + 2*x
grado ejPol3    ⇝  4
```

```
grado:: Polinomio a -> Int
grado PolCero          = 0
grado (ConsPol n _ _) = n
```

- `(coefLider p)` es el coeficiente líder del polinomio p. Por ejemplo,

```
coefLider ejPol3  ⇝  6
```

```
coefLider:: Num t => Polinomio t -> t
coefLider PolCero          = 0
coefLider (ConsPol _ b _) = b
```

Los polinomios como tipo de dato algebraico

- `(grado p)` es el grado del polinomio p. Por ejemplo,

```
ejPol3          ⇝  6*x^4 + 2*x
grado ejPol3    ⇝  4
```

```
grado:: Polinomio a -> Int
grado PolCero          = 0
grado (ConsPol n _ _) = n
```

- `(coefLider p)` es el coeficiente líder del polinomio p. Por ejemplo,

```
coefLider ejPol3  ⇝  6
```

```
coefLider:: Num t => Polinomio t -> t
coefLider PolCero          = 0
coefLider (ConsPol _ b _) = b
```

Los polinomios como tipo de dato algebraico

- `(restoPol p)` es el resto del polinomio `p`. Por ejemplo,

```
ejPol3           ⇝  6*x^4 + 2*x
restoPol ejPol3  ⇝  2*x
ejPol2           ⇝  x^5 + 5*x^2 + 4*x
restoPol ejPol2  ⇝  5*x^2 + 4*x
```

```
restoPol :: (Num a, Eq a) => Polinomio t -> Polinomio t
restoPol PolCero          = PolCero
restoPol (ConsPol _ _ p) = p
```

Los polinomios como tipo de dato algebraico

- `(restoPol p)` es el resto del polinomio `p`. Por ejemplo,

```
ejPol3           ⇝  6*x^4 + 2*x
restoPol ejPol3  ⇝  2*x
ejPol2           ⇝  x^5 + 5*x^2 + 4*x
restoPol ejPol2  ⇝  5*x^2 + 4*x
```

```
restoPol :: (Num a, Eq a) => Polinomio t -> Polinomio t
restoPol PolCero          = PolCero
restoPol (ConsPol _ _ p) = p
```

Tema 21: El TAD de los polinomios

Los polinomios como listas dispersas

Cabecera del módulo

```
module PolRepDispersa
  ( Polinomio,
    polCero,   -- Polinomio a
    esPolCero, -- Polinomio a -> Bool
    consPol,   -- (Num a, Eq a) => Int -> a -> Polinomio a
               --                  -> Polinomio a
    grado,     -- Polinomio a -> Int
    coefLider, -- Num a => Polinomio a -> a
    restoPol   -- (Num a, Eq a) => Polinomio a -> Polinomio a
  ) where
```

Los polinomios como tipo listas dispersas

- Representaremos un polinomio por la lista de sus coeficientes ordenados en orden decreciente según el grado.
- Por ejemplo, el polinomio

  ```
  6x^4 -5x^2 + 4x -7
  ```

  se representa por la lista

  ```
  [6,0,-2,4,-7]
  ```

- Los polinomios como listas dispersas.

```
data Polinomio a = Pol [a]
                 deriving Eq
```

Los polinomios como listas dispersas

Procedimiento de escritura de los polinomios.

```
instance (Num a, Show a, Eq a) => Show (Polinomio a) where
  show pol
      | esPolCero pol          = "0"
      | n == 0 && esPolCero p = show a
      | n == 0                 = concat [show a," + ",show p]
      | n == 1 && esPolCero p = concat [show a,"*x"]
      | n == 1                 = concat [show a,"*x + ",show p]
      | a == 1 && esPolCero p = concat ["x^",show n]
      | esPolCero p            = concat [show a,"*x^",show n]
      | a == 1                 = concat ["x^",show n," + ",show p]
      | otherwise              = concat [show a,"*x^",show n," + ",show p]
    where n = grado pol
          a = coefLider pol
          p = restoPol pol
```

Los polinomios como tipo listas dispersas

- `polCero` es el polinomio cero. Por ejemplo,

```
ghci> polCero
0
```

```
polCero :: Polinomio a
polCero = Pol []
```

- `(esPolCero p)` se verifica si `p` es el polinomio cero. Por ejemplo,

```
esPolCero polCero  ⇝  True
esPolCero ejPol1   ⇝  False
```

```
esPolCero :: Polinomio a -> Bool
esPolCero (Pol []) = True
esPolCero _        = False
```

Los polinomios como tipo listas dispersas

- `polCero` es el polinomio cero. Por ejemplo,

```
ghci> polCero
0
```

```
polCero :: Polinomio a
polCero = Pol []
```

- `(esPolCero p)` se verifica si `p` es el polinomio cero. Por ejemplo,

```
esPolCero polCero  ⇝  True
esPolCero ejPol1   ⇝  False
```

```
esPolCero :: Polinomio a -> Bool
esPolCero (Pol []) = True
esPolCero _        = False
```

Los polinomios como tipo listas dispersas

- `polCero` es el polinomio cero. Por ejemplo,

```
ghci> polCero
0
```

```
polCero :: Polinomio a
polCero = Pol []
```

- `(esPolCero p)` se verifica si `p` es el polinomio cero. Por ejemplo,

```
esPolCero polCero  ⇝  True
esPolCero ejPol1   ⇝  False
```

```
esPolCero :: Polinomio a -> Bool
esPolCero (Pol []) = True
esPolCero _        = False
```

Los polinomios como tipo listas dispersas

- `(consPol n b p)` es el polinomio $bx^n + p$. Por ejemplo,

```
ejPol2                 ⇝  x^5 + 5*x^2 + 4*x
consPol 3 0 ejPol2     ⇝  x^5 + 5*x^2 + 4*x
consPol 3 2 polCero    ⇝  2*x^3
consPol 6 7 ejPol2     ⇝  7*x^6 + x^5 + 5*x^2 + 4*x
consPol 4 7 ejPol2     ⇝  x^5 + 7*x^4 + 5*x^2 + 4*x
consPol 5 7 ejPol2     ⇝  8*x^5 + 5*x^2 + 4*x
```

```haskell
consPol :: (Num a, Eq a) => Int -> a -> Polinomio a -> Polinomio a
consPol _ 0 p = p
consPol n b p@(Pol xs)
    | esPolCero p = Pol (b:replicate n 0)
    | n > m       = Pol (b:(replicate (n-m-1) 0)++xs)
    | n < m       = consPol m c (consPol n b (restoPol p))
    | b+c == 0    = Pol (dropWhile (==0) (tail xs))
    | otherwise   = Pol ((b+c):tail xs)
    where
      c = coefLider p
      m = grado p
```

Los polinomios como tipo listas dispersas

- `(consPol n b p)` es el polinomio $bx^n + p$. Por ejemplo,

```
ejPol2                 ⇝  x^5 + 5*x^2 + 4*x
consPol 3 0 ejPol2     ⇝  x^5 + 5*x^2 + 4*x
consPol 3 2 polCero    ⇝  2*x^3
consPol 6 7 ejPol2     ⇝  7*x^6 + x^5 + 5*x^2 + 4*x
consPol 4 7 ejPol2     ⇝  x^5 + 7*x^4 + 5*x^2 + 4*x
consPol 5 7 ejPol2     ⇝  8*x^5 + 5*x^2 + 4*x
```

```
consPol :: (Num a, Eq a) => Int -> a -> Polinomio a -> Polinomio a
consPol _ 0 p = p
consPol n b p@(Pol xs)
    | esPolCero p = Pol (b:replicate n 0)
    | n > m       = Pol (b:(replicate (n-m-1) 0)++xs)
    | n < m       = consPol m c (consPol n b (restoPol p))
    | b+c == 0    = Pol (dropWhile (==0) (tail xs))
    | otherwise   = Pol ((b+c):tail xs)
    where
      c = coefLider p
      m = grado p
```

Los polinomios como tipo listas dispersas

- `(grado p)` es el grado del polinomio p. Por ejemplo,

```
ejPol3          ⇝  6*x^4 + 2*x
grado ejPol3    ⇝  4
```

```
grado:: Polinomio a -> Int
grado (Pol []) = 0
grado (Pol xs) = length xs - 1
```

- `(coefLider p)` es el coeficiente líder del polinomio p. Por ejemplo,

```
coefLider ejPol3  ⇝  6
```

```
coefLider:: Num t => Polinomio t -> t
coefLider (Pol [])    = 0
coefLider (Pol (a:_)) = a
```

Los polinomios como tipo listas dispersas

- `(grado p)` es el grado del polinomio p. Por ejemplo,

```
ejPol3          ⇝  6*x^4 + 2*x
grado ejPol3    ⇝  4
```

```
grado:: Polinomio a -> Int
grado (Pol []) = 0
grado (Pol xs) = length xs - 1
```

- `(coefLider p)` es el coeficiente líder del polinomio p. Por ejemplo,

```
coefLider ejPol3  ⇝  6
```

```
coefLider:: Num t => Polinomio t -> t
coefLider (Pol [])    = 0
coefLider (Pol (a:_)) = a
```

Los polinomios como tipo listas dispersas

- `(grado p)` es el grado del polinomio p. Por ejemplo,

```
ejPol3          ⇝  6*x^4 + 2*x
grado ejPol3    ⇝  4
```

```
grado:: Polinomio a -> Int
grado (Pol []) = 0
grado (Pol xs) = length xs - 1
```

- `(coefLider p)` es el coeficiente líder del polinomio p. Por ejemplo,

```
coefLider ejPol3  ⇝  6
```

```
coefLider:: Num t => Polinomio t -> t
coefLider (Pol [])    = 0
coefLider (Pol (a:_)) = a
```

Los polinomios como tipo listas dispersas

- `(restoPol p)` es el resto del polinomio `p`. Por ejemplo,

```
ejPol3            ⇝  6*x^4 + 2*x
restoPol ejPol3   ⇝  2*x
ejPol2            ⇝  x^5 + 5*x^2 + 4*x
restoPol ejPol2   ⇝  5*x^2 + 4*x
```

```
restoPol :: (Num t, Eq t) => Polinomio t -> Polinomio t
restoPol (Pol [])     = polCero
restoPol (Pol [_])    = polCero
restoPol (Pol (_:b:as))
    | b == 0    = Pol (dropWhile (==0) as)
    | otherwise = Pol (b:as)
```

Los polinomios como tipo listas dispersas

- `(restoPol p)` es el resto del polinomio `p`. Por ejemplo,

```
ejPol3           ⇝  6*x^4 + 2*x
restoPol ejPol3  ⇝  2*x
ejPol2           ⇝  x^5 + 5*x^2 + 4*x
restoPol ejPol2  ⇝  5*x^2 + 4*x
```

```
restoPol :: (Num t, Eq t) => Polinomio t -> Polinomio t
restoPol (Pol [])      = polCero
restoPol (Pol [_])     = polCero
restoPol (Pol (_:b:as))
    | b == 0    = Pol (dropWhile (==0) as)
    | otherwise = Pol (b:as)
```

Tema 21: El TAD de los polinomios

Los polinomios como tipo listas densas

Cabecera del módulo.

```
module PolRepDensa
  ( Polinomio,
    polCero,   -- Polinomio a
    esPolCero, -- Polinomio a -> Bool
    consPol,   -- (Num a, Eq a) => Int -> a -> Polinomio a
               --                  -> Polinomio a
    grado,     -- Polinomio a -> Int
    coefLider, -- Num a => Polinomio a -> a
    restoPol   -- (Num a, Eq a) => Polinomio a -> Polinomio a
  ) where
```

Los polinomios como tipo listas densas

- Representaremos un polinomio mediante una lista de pares (grado,coef), ordenados en orden decreciente según el grado. Por ejemplo, el polinomio

  ```
  6x^4 -5x^2 + 4x -7
  ```

  se representa por la lista de pares

  ```
  [(4,6),(2,-5),(1,4),(0,-7)].
  ```

- Los polinomios como listas densas.

```
data Polinomio a = Pol [(Int,a)]
                 deriving Eq
```

Los polinomios como tipo listas densas

Procedimiento de escritura de polinomios

```
instance (Num a, Show a, Eq a) => Show (Polinomio a) where
  show pol
      | esPolCero pol          = "0"
      | n == 0 && esPolCero p = show a
      | n == 0                 = concat [show a," + ",show p]
      | n == 1 && esPolCero p = concat [show a,"*x"]
      | n == 1                 = concat [show a,"*x + ",show p]
      | a == 1 && esPolCero p = concat ["x^",show n]
      | esPolCero p            = concat [show a,"*x^",show n]
      | a == 1                 = concat ["x^",show n," + ",show p]
      | otherwise              = concat [show a,"*x^",show n," + ",show p]
    where n = grado pol
          a = coefLider pol
          p = restoPol pol
```

Los polinomios como tipo listas densas

- `polCero` es el polinomio cero. Por ejemplo,

```
ghci> polCero
0
```

```
polCero :: Polinomio a
polCero = Pol []
```

- `(esPolCero p)` se verifica si `p` es el polinomio cero. Por ejemplo,

```
esPolCero polCero  ⇝  True
esPolCero ejPol1   ⇝  False
```

```
esPolCero :: Polinomio a -> Bool
esPolCero (Pol []) = True
esPolCero _        = False
```

Los polinomios como tipo listas densas

- `polCero` es el polinomio cero. Por ejemplo,

```
ghci> polCero
0
```

```
polCero :: Polinomio a
polCero = Pol []
```

- `(esPolCero p)` se verifica si `p` es el polinomio cero. Por ejemplo,

```
esPolCero polCero  ⇝  True
esPolCero ejPol1   ⇝  False
```

```
esPolCero :: Polinomio a -> Bool
esPolCero (Pol []) = True
esPolCero _        = False
```

Los polinomios como tipo listas densas

- `polCero` es el polinomio cero. Por ejemplo,

```
ghci> polCero
0
```

```
polCero :: Polinomio a
polCero = Pol []
```

- `(esPolCero p)` se verifica si `p` es el polinomio cero. Por ejemplo,

```
esPolCero polCero  ⇝  True
esPolCero ejPol1   ⇝  False
```

```
esPolCero :: Polinomio a -> Bool
esPolCero (Pol []) = True
esPolCero _        = False
```

Los polinomios como tipo listas densas

- `(consPol n b p)` es el polinomio $bx^n + p$. Por ejemplo,

```
ejPol2                 ⇝  x^5 + 5*x^2 + 4*x
consPol 3 0 ejPol2     ⇝  x^5 + 5*x^2 + 4*x
consPol 3 2 polCero    ⇝  2*x^3
consPol 6 7 ejPol2     ⇝  7*x^6 + x^5 + 5*x^2 + 4*x
consPol 4 7 ejPol2     ⇝  x^5 + 7*x^4 + 5*x^2 + 4*x
consPol 5 7 ejPol2     ⇝  8*x^5 + 5*x^2 + 4*x
```

```haskell
consPol :: (Num a, Eq a) => Int -> a -> Polinomio a -> Polinomio a
consPol _ 0 p = p
consPol n b p@(Pol xs)
    | esPolCero p = Pol [(n,b)]
    | n > m       = Pol ((n,b):xs)
    | n < m       = consPol m c (consPol n b (Pol (tail xs)))
    | b+c == 0    = Pol (tail xs)
    | otherwise   = Pol ((n,b+c):(tail xs))
    where
      c = coefLider p
      m = grado p
```

Los polinomios como tipo listas densas

- `(consPol n b p)` es el polinomio $bx^n + p$. Por ejemplo,

```
ejPol2                 ⇝  x^5 + 5*x^2 + 4*x
consPol 3 0 ejPol2     ⇝  x^5 + 5*x^2 + 4*x
consPol 3 2 polCero    ⇝  2*x^3
consPol 6 7 ejPol2     ⇝  7*x^6 + x^5 + 5*x^2 + 4*x
consPol 4 7 ejPol2     ⇝  x^5 + 7*x^4 + 5*x^2 + 4*x
consPol 5 7 ejPol2     ⇝  8*x^5 + 5*x^2 + 4*x
```

```
consPol :: (Num a, Eq a) => Int -> a -> Polinomio a -> Polinomio a
consPol _ 0 p = p
consPol n b p@(Pol xs)
    | esPolCero p = Pol [(n,b)]
    | n > m       = Pol ((n,b):xs)
    | n < m       = consPol m c (consPol n b (Pol (tail xs)))
    | b+c == 0    = Pol (tail xs)
    | otherwise   = Pol ((n,b+c):(tail xs))
    where
      c = coefLider p
      m = grado p
```

Los polinomios como tipo listas densas

- `(grado p)` es el grado del polinomio p. Por ejemplo,

```
ejPol3          ⇝  6*x^4 + 2*x
grado ejPol3    ⇝  4
```

```
grado:: Polinomio a -> Int
grado (Pol [])        = 0
grado (Pol ((n,_):_)) = n
```

- `(coefLider p)` es el coeficiente líder del polinomio p. Por ejemplo,

```
coefLider ejPol3  ⇝  6
```

```
coefLider:: Num t => Polinomio t -> t
coefLider (Pol [])        = 0
coefLider (Pol ((_,b):_)) = b
```

Los polinomios como tipo listas densas

- `(grado p)` es el grado del polinomio p. Por ejemplo,

```
ejPol3          ⇝  6*x^4 + 2*x
grado ejPol3    ⇝  4
```

```
grado:: Polinomio a -> Int
grado (Pol [])        = 0
grado (Pol ((n,_):_)) = n
```

- `(coefLider p)` es el coeficiente líder del polinomio p. Por ejemplo,

```
coefLider ejPol3  ⇝  6
```

```
coefLider:: Num t => Polinomio t -> t
coefLider (Pol [])        = 0
coefLider (Pol ((_,b):_)) = b
```

Los polinomios como tipo listas densas

- `(grado p)` es el grado del polinomio p. Por ejemplo,

```
ejPol3          ⇝  6*x^4 + 2*x
grado ejPol3    ⇝  4
```

```
grado:: Polinomio a -> Int
grado (Pol [])         = 0
grado (Pol ((n,_):_)) = n
```

- `(coefLider p)` es el coeficiente líder del polinomio p. Por ejemplo,

```
coefLider ejPol3  ⇝  6
```

```
coefLider:: Num t => Polinomio t -> t
coefLider (Pol [])         = 0
coefLider (Pol ((_,b):_)) = b
```

Los polinomios como tipo listas densas

- `(restoPol p)` es el resto del polinomio `p`. Por ejemplo,

```
ejPol3           ⇝  6*x^4 + 2*x
restoPol ejPol3  ⇝  2*x
ejPol2           ⇝  x^5 + 5*x^2 + 4*x
restoPol ejPol2  ⇝  5*x^2 + 4*x
```

```
restoPol :: (Num t, Eq t) => Polinomio t -> Polinomio t
restoPol (Pol [])     = polCero
restoPol (Pol [_])    = polCero
restoPol (Pol (_:xs)) = Pol xs
```

Los polinomios como tipo listas densas

- `(restoPol p)` es el resto del polinomio `p`. Por ejemplo,

```
ejPol3           ⇝  6*x^4 + 2*x
restoPol ejPol3  ⇝  2*x
ejPol2           ⇝  x^5 + 5*x^2 + 4*x
restoPol ejPol2  ⇝  5*x^2 + 4*x
```

```
restoPol :: (Num t, Eq t) => Polinomio t -> Polinomio t
restoPol (Pol [])     = polCero
restoPol (Pol [_])    = polCero
restoPol (Pol (_:xs)) = Pol xs
```

Tema 21: El TAD de los polinomios

Comprobación de las propiedades del TAD de los polinomios

- Importación de la implementación a verificar.

```
import PolRepTDA
-- import PolRepDispersa
-- import PolRepDensa
```

- Librerías auxiliares.

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

Tema 21: El TAD de los polinomios

Generador de polinomios

- (genPol n) es un generador de polinomios. Por ejemplo,

```
ghci> sample (genPol 1)
7*x^9 + 9*x^8 + 10*x^7 + -14*x^5 + -15*x^2 + -10
-4*x^8 + 2*x
```

```
genPol :: Int -> Gen (Polinomio Int)
genPol 0 = return polCero
genPol n = do n <- choose (0,10)
              b <- choose (-10,10)
              p <- genPol (div n 2)
              return (consPol n b p)

instance Arbitrary (Polinomio Int) where
    arbitrary = sized genPol
```

Tema 21: El TAD de los polinomios

Especificación de las propiedades de los polinomios

- `polCero` es el polinomio cero.

```
prop_polCero_es_cero :: Bool
prop_polCero_es_cero =
    esPolCero polCero
```

- Si `n` es mayor que el grado de `p` y `b` no es cero, entonces `(consPol n b p)` es un polinomio distinto del cero.

```
prop_consPol_no_cero :: Int -> Int -> Polinomio Int
                        -> Property
prop_consPol_no_cero n b p =
    n > grado p && b /= 0  ==>
      not (esPolCero (consPol n b p))
```

Especificación de las propiedades de los polinomios

- `polCero` es el polinomio cero.

```
prop_polCero_es_cero :: Bool
prop_polCero_es_cero =
    esPolCero polCero
```

- Si `n` es mayor que el grado de `p` y `b` no es cero, entonces `(consPol n b p)` es un polinomio distinto del cero.

```
prop_consPol_no_cero :: Int -> Int -> Polinomio Int
                     -> Property
prop_consPol_no_cero n b p =
    n > grado p && b /= 0  ==>
      not (esPolCero (consPol n b p))
```

Especificación de las propiedades de los polinomios

- `polCero` es el polinomio cero.

```
prop_polCero_es_cero :: Bool
prop_polCero_es_cero =
    esPolCero polCero
```

- Si `n` es mayor que el grado de `p` y `b` no es cero, entonces `(consPol n b p)` es un polinomio distinto del cero.

```
prop_consPol_no_cero :: Int -> Int -> Polinomio Int
                        -> Property
prop_consPol_no_cero n b p =
    n > grado p && b /= 0  ==>
      not (esPolCero (consPol n b p))
```

Especificación de las propiedades de los polinomios

- `(consPol (grado p) (coefLider p) (restoPol p))` es igual a `p`.

```
prop_consPol :: Polinomio Int -> Bool
prop_consPol p =
    consPol (grado p) (coefLider p) (restoPol p) == p
```

- Si `n` es mayor que el grado de `p` y `b` no es cero, entonces el grado de `(consPol n b p)` es `n`.

```
prop_grado :: Int -> Int -> Polinomio Int -> Property
prop_grado n b p =
    n > grado p && b /= 0 ==>
      grado (consPol n b p) == n
```

Especificación de las propiedades de los polinomios

- `(consPol (grado p) (coefLider p) (restoPol p))` es igual a `p`.

```
prop_consPol :: Polinomio Int -> Bool
prop_consPol p =
    consPol (grado p) (coefLider p) (restoPol p) == p
```

- Si `n` es mayor que el grado de `p` y `b` no es cero, entonces el grado de `(consPol n b p)` es `n`.

```
prop_grado :: Int -> Int -> Polinomio Int -> Property
prop_grado n b p =
    n > grado p && b /= 0 ==>
      grado (consPol n b p) == n
```

Especificación de las propiedades de los polinomios

- `(consPol (grado p) (coefLider p) (restoPol p))` es igual a `p`.

```
prop_consPol :: Polinomio Int -> Bool
prop_consPol p =
    consPol (grado p) (coefLider p) (restoPol p) == p
```

- Si `n` es mayor que el grado de `p` y `b` no es cero, entonces el grado de `(consPol n b p)` es `n`.

```
prop_grado :: Int -> Int -> Polinomio Int -> Property
prop_grado n b p =
    n > grado p && b /= 0 ==>
      grado (consPol n b p) == n
```

Especificación de las propiedades de los polinomios

- Si `n` es mayor que el grado de `p` y `b` no es cero, entonces el coeficiente líder de `(consPol n b p)` es `b`.

```
prop_coefLider :: Int -> Int -> Polinomio Int -> Property
prop_coefLider n b p =
    n > grado p && b /= 0 ==>
      coefLider (consPol n b p) == b
```

- Si `n` es mayor que el grado de `p` y `b` no es cero, entonces el resto de `(consPol n b p)` es `p`.

```
prop_restoPol :: Int -> Int -> Polinomio Int -> Property
prop_restoPol n b p =
    n > grado p && b /= 0 ==>
      restoPol (consPol n b p) == p
```

Especificación de las propiedades de los polinomios

- Si `n` es mayor que el grado de `p` y `b` no es cero, entonces el coeficiente líder de `(consPol n b p)` es `b`.

```
prop_coefLider :: Int -> Int -> Polinomio Int -> Property
prop_coefLider n b p =
    n > grado p && b /= 0 ==>
      coefLider (consPol n b p) == b
```

- Si `n` es mayor que el grado de `p` y `b` no es cero, entonces el resto de `(consPol n b p)` es `p`.

```
prop_restoPol :: Int -> Int -> Polinomio Int -> Property
prop_restoPol n b p =
    n > grado p && b /= 0 ==>
      restoPol (consPol n b p) == p
```

Especificación de las propiedades de los polinomios

- Si `n` es mayor que el grado de `p` y `b` no es cero, entonces el coeficiente líder de `(consPol n b p)` es `b`.

```
prop_coefLider :: Int -> Int -> Polinomio Int -> Property
prop_coefLider n b p =
    n > grado p && b /= 0 ==>
      coefLider (consPol n b p) == b
```

- Si `n` es mayor que el grado de `p` y `b` no es cero, entonces el resto de `(consPol n b p)` es `p`.

```
prop_restoPol :: Int -> Int -> Polinomio Int -> Property
prop_restoPol n b p =
    n > grado p && b /= 0 ==>
      restoPol (consPol n b p) == p
```

Tema 21: El TAD de los polinomios

Procedimiento de comprobación

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

```
compruebaPropiedades =
    defaultMain
        [testGroup "Propiedades del TAD polinomio:"
          [testProperty "P1" prop_polCero_es_cero,
           testProperty "P2" prop_consPol_no_cero,
           testProperty "P3" prop_consPol,
           testProperty "P4" prop_grado,
           testProperty "P5" prop_coefLider,
           testProperty "P6" prop_restoPol]]
```

Comprobación de las propiedades de los polinomios

```
ghci> compruebaPropiedades
Propiedades del TAD polinomio::
  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]

         Properties  Total
 Passed  6           6
 Failed  0           0
 Total   6           6
```

Tema 21: El TAD de los polinomios

1. Especificación del TAD de los polinomios
2. Implementación del TAD de los polinomios
3. Comprobación de las implementaciones con QuickCheck
4. Operaciones con polinomios
   - Operaciones con polinomios

Operaciones con polinomios

- Importación de la implementación a utilizar.

```
import PolRepTDA
-- import PolRepDispersa
-- import PolRepDensa
```

- Importación de librerías auxiliares.

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

Funciones sobre términos

- `(creaTermino n a)` es el término ax^n. Por ejemplo,

```
creaTermino 2 5  ⇝  5*x^2
```

```
creaTermino:: (Num t, Eq t) => Int -> t -> Polinomio t
creaTermino n a = consPol n a polCero
```

- `(termLider p)` es el término líder del polinomio `p`. Por ejemplo,

```
ejPol2              ⇝  x^5 + 5*x^2 + 4*x
termLider ejPol2    ⇝  x^5
```

```
termLider:: (Num t, Eq t) => Polinomio t -> Polinomio t
termLider p = creaTermino (grado p) (coefLider p)
```

Funciones sobre términos

- `(creaTermino n a)` es el término ax^n. Por ejemplo,

```
creaTermino 2 5   ⇝   5*x^2
```

```
creaTermino:: (Num t, Eq t) => Int -> t -> Polinomio t
creaTermino n a = consPol n a polCero
```

- `(termLider p)` es el término líder del polinomio `p`. Por ejemplo,

```
ejPol2              ⇝   x^5 + 5*x^2 + 4*x
termLider ejPol2    ⇝   x^5
```

```
termLider:: (Num t, Eq t) => Polinomio t -> Polinomio t
termLider p = creaTermino (grado p) (coefLider p)
```

Funciones sobre términos

- `(creaTermino n a)` es el término ax^n. Por ejemplo,

```
creaTermino 2 5   ⇝   5*x^2
```

```
creaTermino:: (Num t, Eq t) => Int -> t -> Polinomio t
creaTermino n a = consPol n a polCero
```

- `(termLider p)` es el término líder del polinomio `p`. Por ejemplo,

```
ejPol2              ⇝   x^5 + 5*x^2 + 4*x
termLider ejPol2    ⇝   x^5
```

```
termLider:: (Num t, Eq t) => Polinomio t -> Polinomio t
termLider p = creaTermino (grado p) (coefLider p)
```

Suma de polinomios

- `(sumaPol p q)` es la suma de los polinomios `p` y `q`. Por ejemplo,

```
ejPol1                  ⇝  3*x^4 + -5*x^2 + 3
ejPol2                  ⇝  x^5 + 5*x^2 + 4*x
sumaPol ejPol1 ejPol2   ⇝  x^5 + 3*x^4 + 4*x + 3
```

```
sumaPol:: (Num a, Eq a) => Polinomio a -> Polinomio a -> Polinomio a
sumaPol p q
    | esPolCero p = q
    | esPolCero q = p
    | n1 > n2     = consPol n1 a1 (sumaPol r1 q)
    | n1 < n2     = consPol n2 a2 (sumaPol p r2)
    | otherwise   = consPol n1 (a1+a2) (sumaPol r1 r2)
    where n1 = grado p
          a1 = coefLider p
          r1 = restoPol p
          n2 = grado q
          a2 = coefLider q
          r2 = restoPol q
```

Suma de polinomios

- `(sumaPol p q)` es la suma de los polinomios `p` y `q`. Por ejemplo,

```
ejPol1                  ⇝  3*x^4 + -5*x^2 + 3
ejPol2                  ⇝  x^5 + 5*x^2 + 4*x
sumaPol ejPol1 ejPol2   ⇝  x^5 + 3*x^4 + 4*x + 3
```

```
sumaPol:: (Num a, Eq a) => Polinomio a -> Polinomio a -> Polinomio a
sumaPol p q
    | esPolCero p = q
    | esPolCero q = p
    | n1 > n2     = consPol n1 a1 (sumaPol r1 q)
    | n1 < n2     = consPol n2 a2 (sumaPol p r2)
    | otherwise   = consPol n1 (a1+a2) (sumaPol r1 r2)
    where n1 = grado p
          a1 = coefLider p
          r1 = restoPol p
          n2 = grado q
          a2 = coefLider q
          r2 = restoPol q
```

Propiedades de la suma de polinomios

- El polinomio cero es el elemento neutro de la suma.

```
prop_neutroSumaPol :: Polinomio Int -> Bool
prop_neutroSumaPol p =
    sumaPol polCero p == p
```

- La suma es conmutativa.

```
prop_conmutativaSuma :: Polinomio Int -> Polinomio Int
                     -> Bool
prop_conmutativaSuma p q =
    sumaPol p q == sumaPol q p
```

Propiedades de la suma de polinomios

- El polinomio cero es el elemento neutro de la suma.

```
prop_neutroSumaPol :: Polinomio Int -> Bool
prop_neutroSumaPol p =
    sumaPol polCero p == p
```

- La suma es conmutativa.

```
prop_conmutativaSuma :: Polinomio Int -> Polinomio Int
                     -> Bool
prop_conmutativaSuma p q =
    sumaPol p q == sumaPol q p
```

Propiedades de la suma de polinomios

- El polinomio cero es el elemento neutro de la suma.

```
prop_neutroSumaPol :: Polinomio Int -> Bool
prop_neutroSumaPol p =
    sumaPol polCero p == p
```

- La suma es conmutativa.

```
prop_conmutativaSuma :: Polinomio Int -> Polinomio Int
                        -> Bool
prop_conmutativaSuma p q =
    sumaPol p q == sumaPol q p
```

Producto de polinomios

- `(multPorTerm t p)` es el producto del término `t` por el polinomio `p`. Por ejemplo,

```
ejTerm                          ⇝  4*x
ejPol2                          ⇝  x^5 + 5*x^2 + 4*x
multPorTerm ejTerm ejPol2       ⇝  4*x^6 + 20*x^3 + 16*x^2
```

```
multPorTerm :: (Num t, Eq t) => Polinomio t -> Polinomio t -> Poli
multPorTerm term pol
    | esPolCero pol = polCero
    | otherwise     = consPol (n+m) (a*b) (multPorTerm term r)
    where n = grado term
          a = coefLider term
          m = grado pol
          b = coefLider pol
          r = restoPol pol
```

Producto de polinomios

- `(multPorTerm t p)` es el producto del término `t` por el polinomio `p`. Por ejemplo,

```
ejTerm                          ⇝  4*x
ejPol2                          ⇝  x^5 + 5*x^2 + 4*x
multPorTerm ejTerm ejPol2       ⇝  4*x^6 + 20*x^3 + 16*x^2
```

```
multPorTerm :: (Num t, Eq t) => Polinomio t -> Polinomio t -> Poli
multPorTerm term pol
    | esPolCero pol = polCero
    | otherwise     = consPol (n+m) (a*b) (multPorTerm term r)
    where n = grado term
          a = coefLider term
          m = grado pol
          b = coefLider pol
          r = restoPol pol
```

Producto de polinomios

- `(multPol p q)` es el producto de los polinomios `p` y `q`. Por ejemplo,

```
ghci> ejPol1
3*x^4 + -5*x^2 + 3
ghci> ejPol2
x^5 + 5*x^2 + 4*x
ghci> multPol ejPol1 ejPol2
3*x^9 + -5*x^7 + 15*x^6 + 15*x^5 + -25*x^4 + -20*x^3
      + 15*x^2 + 12*x
```

```
multPol :: (Num a, Eq a) => Polinomio a -> Polinomio a -> Polinom
multPol p q
    | esPolCero p = polCero
    | otherwise   = sumaPol (multPorTerm (termLider p) q)
                            (multPol (restoPol p) q)
```

Producto de polinomios

- `(multPol p q)` es el producto de los polinomios `p` y `q`. Por ejemplo,

```
ghci> ejPol1
3*x^4 + -5*x^2 + 3
ghci> ejPol2
x^5 + 5*x^2 + 4*x
ghci> multPol ejPol1 ejPol2
3*x^9 + -5*x^7 + 15*x^6 + 15*x^5 + -25*x^4 + -20*x^3
      + 15*x^2 + 12*x
```

```
multPol :: (Num a, Eq a) => Polinomio a -> Polinomio a -> Polinom
multPol p q
    | esPolCero p = polCero
    | otherwise   = sumaPol (multPorTerm (termLider p) q)
                            (multPol (restoPol p) q)
```

Propiedades del producto polinomios

- El producto de polinomios es conmutativo.

```
prop_conmutativaProducto :: Polinomio Int
                         -> Polinomio Int -> Bool
prop_conmutativaProducto p q =
    multPol p q == multPol q p
```

- El producto es distributivo respecto de la suma.

```
prop_distributiva :: Polinomio Int -> Polinomio Int
                  -> Polinomio Int -> Bool
prop_distributiva p q r =
    multPol p (sumaPol q r) ==
    sumaPol (multPol p q) (multPol p r)
```

Propiedades del producto polinomios

- El producto de polinomios es conmutativo.

```
prop_conmutativaProducto :: Polinomio Int
                         -> Polinomio Int -> Bool
prop_conmutativaProducto p q =
    multPol p q == multPol q p
```

- El producto es distributivo respecto de la suma.

```
prop_distributiva :: Polinomio Int -> Polinomio Int
                  -> Polinomio Int -> Bool
prop_distributiva p q r =
    multPol p (sumaPol q r) ==
    sumaPol (multPol p q) (multPol p r)
```

Propiedades del producto polinomios

- El producto de polinomios es conmutativo.

```
prop_conmutativaProducto :: Polinomio Int
                         -> Polinomio Int -> Bool
prop_conmutativaProducto p q =
    multPol p q == multPol q p
```

- El producto es distributivo respecto de la suma.

```
prop_distributiva :: Polinomio Int -> Polinomio Int
                  -> Polinomio Int -> Bool
prop_distributiva p q r =
    multPol p (sumaPol q r) ==
    sumaPol (multPol p q) (multPol p r)
```

Polinomio unidad

- `polUnidad` es el polinomio unidad. Por ejemplo,

```
ghci> polUnidad
1
```

```
polUnidad:: (Num t, Eq t) => Polinomio t
polUnidad = consPol 0 1 polCero
```

- El polinomio unidad es el elemento neutro del producto.

```
prop_polUnidad :: Polinomio Int -> Bool
prop_polUnidad p =
    multPol p polUnidad == p
```

Polinomio unidad

- `polUnidad` es el polinomio unidad. Por ejemplo,

```
ghci> polUnidad
1
```

```
polUnidad:: (Num t, Eq t) => Polinomio t
polUnidad = consPol 0 1 polCero
```

- El polinomio unidad es el elemento neutro del producto.

```
prop_polUnidad :: Polinomio Int -> Bool
prop_polUnidad p =
    multPol p polUnidad == p
```

Polinomio unidad

- `polUnidad` es el polinomio unidad. Por ejemplo,

```
ghci> polUnidad
1
```

```
polUnidad:: (Num t, Eq t) => Polinomio t
polUnidad = consPol 0 1 polCero
```

- El polinomio unidad es el elemento neutro del producto.

```
prop_polUnidad :: Polinomio Int -> Bool
prop_polUnidad p =
    multPol p polUnidad == p
```

Valor de un polinomio en un punto

- `(valor p c)` es el valor del polinomio `p` al sustituir su variable por `c`. Por ejemplo,

```
ejPol1              ⇝  3*x^4 + -5*x^2 + 3
valor ejPol1 0      ⇝  3
valor ejPol1 1      ⇝  1
valor ejPol1 (-2)   ⇝  31
```

```
valor:: (Num a, Eq a) => Polinomio a -> a -> a
valor p c
    | esPolCero p = 0
    | otherwise   =  b*c^n + valor r c
    where n = grado p
          b = coefLider p
          r = restoPol p
```

Valor de un polinomio en un punto

- `(valor p c)` es el valor del polinomio `p` al sustituir su variable por `c`. Por ejemplo,

```
ejPol1              ⇝  3*x^4 + -5*x^2 + 3
valor ejPol1 0      ⇝  3
valor ejPol1 1      ⇝  1
valor ejPol1 (-2)   ⇝  31
```

```
valor:: (Num a, Eq a) => Polinomio a -> a -> a
valor p c
    | esPolCero p = 0
    | otherwise   =  b*c^n + valor r c
    where n = grado p
          b = coefLider p
          r = restoPol p
```

Verificación de raices de polinomios

- `(esRaiz c p)` se verifica si c es una raiz del polinomio p. por ejemplo,

```
ejPol3              ⇝  6*x^4 + 2*x
esRaiz 1 ejPol3     ⇝  False
esRaiz 0 ejPol3     ⇝  True
```

```
esRaiz:: (Num a, Eq a) => a -> Polinomio a -> Bool
esRaiz c p = valor p c == 0
```

Verificación de raices de polinomios

- `(esRaiz c p)` se verifica si c es una raiz del polinomio p. por ejemplo,

```
ejPol3            ⇝  6*x^4 + 2*x
esRaiz 1 ejPol3   ⇝  False
esRaiz 0 ejPol3   ⇝  True
```

```
esRaiz:: (Num a, Eq a) => a -> Polinomio a -> Bool
esRaiz c p = valor p c == 0
```

Derivación de polinomios

- `(derivada p)` es la derivada del polinomio `p`. Por ejemplo,

```
ejPol2            ⇝  x^5 + 5*x^2 + 4*x
derivada ejPol2   ⇝  5*x^4 + 10*x + 4
```

```
derivada :: Polinomio Int -> Polinomio Int
derivada p
    | n == 0     = polCero
    | otherwise  = consPol (n-1) (n*b) (derivada r)
    where n = grado p
          b = coefLider p
          r = restoPol p
```

Derivación de polinomios

- `(derivada p)` es la derivada del polinomio `p`. Por ejemplo,

```
ejPol2            ⇝  x^5 + 5*x^2 + 4*x
derivada ejPol2   ⇝  5*x^4 + 10*x + 4
```

```
derivada :: Polinomio Int -> Polinomio Int
derivada p
    | n == 0     = polCero
    | otherwise  = consPol (n-1) (n*b) (derivada r)
    where n = grado p
          b = coefLider p
          r = restoPol p
```

Propiedades de las derivadas de polinomios

- La derivada de la suma es la suma de las derivadas.

```
prop_derivada :: Polinomio Int -> Polinomio Int -> Bool
prop_derivada p q =
    derivada (sumaPol p q) ==
    sumaPol (derivada p) (derivada q)
```

Propiedades de las derivadas de polinomios

- La derivada de la suma es la suma de las derivadas.

```
prop_derivada :: Polinomio Int -> Polinomio Int -> Bool
prop_derivada p q =
    derivada (sumaPol p q) ==
    sumaPol (derivada p) (derivada q)
```