10. Considerando las definiciones de los ejercicios anteriores demostrá por inducción sobre xs las siguientes propiedades:

a) sum.(sumar1.xs) = sum.xs + #xs

Definiciones:

sum [] = 0 (1)

sum (x:xs) = x + (sum xs) (2)

sumar1 [] = [] (3)

sumar1 (x:xs) = (1 + x) : sumar1 xs (4)

#[] = 0 (5)

#(x:xs) = 1 + # xs (6)

Caso base:

Reemplazo a xs por [ ]

```
sum (sumar1 xs) = sum xs + #xs
 sum (sumar1 []) = sum [] + #[]
      ≡ { Por (1), (3) y (5) }
         sum ([]) = 0 + 0
            ≡ { Por (1) }
               0 = 0
      ≡ { Reflexividad del = }
               True
```

Caso Inductivo:

Demostramos la propiedad con una lista no vacia (x:xs).

```
sum (sumar1 xs) = sum xs + #xs
sum (sumar1 (x:xs)) = sum (x:xs) + #(x:xs)
      ≡ { Por (2), (4) y (6) }
sum ((1 + x) : sumar1 xs) = x + (sum xs) + 1 + #xs
            ≡ { Por (2) }
(1 + x) + (sum (sumar1 xs)) = 1 + x + (sum xs) + #xs
            ≡ { Por HI }
1 + x + (sum xs) + #xs = 1 + x + (sum xs) + #xs
      ≡ { Reflexividad del = }
               True
```

b) sum.(duplica.xs) = 2 * sum.xs

Definiciones:

sum [] = 0 (1)

sum (x:xs) = x + (sum xs) (2)

duplica [] = [] (3)

duplica (x:xs) = (2 * x) : duplica xs (4)

Caso base:
Reemplazo xs por [ ]

```
sum (duplica xs) = 2 * sum xs
sum (duplica []) = 2 * sum []
     ≡ { Por (1) y (3) }
    sum ([]) = 2 * 0
       ≡ { Por (1) }
           0 = 0
   ≡ { Reflexividad del = }
           True
```

Caso inductivo:
Demostramos la propiedad con una lista no vacia (x:xs).

```
sum (duplica xs) = 2 * sum xs
sum (duplica (x:xs)) = 2 * sum (x:xs)
     ≡ { Por (2) y (4) }
sum ((2*x) : duplica xs) = 2 * (x + (sum xs))
       ≡ { Por (2) }
(2*x) + sum (duplica xs) = 2*x + 2 * (sum xs)
        ≡ { Por HI }
(2*x) + sum (duplica xs) = (2*x) + sum (duplica xs)
   ≡ { Reflexividad del = }
           True
```

c) #(duplica.xs) = #xs

Definiciones:

#[] = 0	(1)
#(x:xs) = 1 + # xs	(2)
duplica [] = []	(3)
duplica (x:xs) = (2 * x) : duplica xs	(4)

Caso base:

Reemplazamos xs con [ ]

```
        #(duplica xs) = #xs
        #(duplica []) = #[]
        ≡ { Por (1) y (3) }
            #([]) = 0
          ≡ { Por (1) }
              0 = 0
    ≡ { Reflexividad del = }
              True
```

Caso inductivo:

Demostramos la propiedad con una lista no vacia (x:xs).

```
          #(duplica xs) = #xs
     #(duplica (x:xs)) = #(x:xs)
          ≡ { Por (2) y (4) }
    #((2*x): duplica xs) = 1 + #xs
            ≡ { Por (2) }
     1 + #(duplica xs) = 1 + #xs
             ≡ { Por HI }
           1 + #xs = 1 + #xs
      ≡ { Reflexividad del = }
                True
```

11. Demostrá por inducción las siguientes propiedades.
a) xs ++ [ ] = xs

[] ++ ls = ls (1)
(x:xs) ++ ls = x : (xs ++ ls) (2)

Caso base: Reemplazamos a xs con []:

```
xs ++ [] = xs

[] ++ [] = []
={Por (1), xs:= []; ls := []}
[] = []
={Reflexividad del =}
True
```

Caso Inductivo: Demostramos la propiedad con una lista no vacia (x:xs).

```
xs ++ [] = xs

x:xs ++ [] = x:xs
={por (2), x:=x; xs:=xs; ls:=[]}
x: (xs ++ []) = x:xs
={Por (HI)}
x:xs = x:xs
={Reflexividad del =}
True
```

b) #xs ⩾ 0

[] = 0 (1)
(x:xs) = 1 + # xs (2)

Reemplazamos a xs por []:

```
#xs ⩾ 0

#[] ⩾ 0
={Por (1)}
0 ⩾ 0
= {Reflexividad del ⩾}
True
```

Caso inductivo: Reemplazamos a xs por una lista no vacia (x:xs)

```
#xs ⩾ 0

#(x:xs) ⩾ 0
={Por (2), x:=x; xs:=xs}
1 + #xs ⩾ 0
={Por HI, #xs = 0 ∨ #xs > 0}
1 + 0 ⩾ 0
= {Aritmetica}
1 ⩾ 0
= {Se cumple que 1 es mayor a 0}
True
```

12. Considerando la función $quitarCeros : [Num] \to [Num]$ definida de la siguiente manera

$$
\begin{array}{lcll}
quitarCeros.[] & \doteq & [] & (3) \\
quitarCeros.(x \triangleright xs) & \doteq & (\quad x \neq 0 \to x \triangleright quitarCeros.xs & (4) \\
 & & \Box \quad x = 0 \to quitarCeros.xs & (5) \\
 & &) &
\end{array}
$$

demostrá que

$$sum.(quitarCeros.xs) = sum.xs$$

sum [] = 0 (1)

sum (x:xs) = x + (sum xs) (2)

Caso base:

```
sum (quitarCeros xs) = sum xs

sum (quitarCeros []) = sum []
={Por (3) y (1)}
sum [] = 0
={Por (1)}
0 = 0
={Reflexividad del = }
True
```

Caso inductivo:

Caso 1: x /= 0

```
sum (quitarCeros xs) = sum xs

sum (quitarCeros x:xs) = sum x:xs
={Por (4) y (2)}
sum (x: quitarCeros xs) = x + sum xs
={Por (2), x:=x; xs:= quitarCeros xs}
x + sum (quitarCeros xs) = x + sum xs
={Por HI}
x + sum xs = x + sum xs
={Reflexividad del =}
True
```

Caso 2: x == 0

```
sum (quitarCeros xs) = sum xs

sum (quitarCeros x:xs) = sum x:xs
={Por (5) y (2)}
sum (quitarCeros xs) = x + sum xs
={Por HI y x == 0}
sum xs = 0 + sum xs
={Aritmética}
sum xs = sum xs
={Reflexividad del =}
True
```

13. Considerando la función $soloPares : [Num] \to [Num]$ definida de la siguiente manera

$$\begin{array}{lcll} soloPares.[] & \doteq & [] & (1) \\ soloPares.(x \triangleright xs) & \doteq & (\quad x \ mod\ 2 = 0 \to x \triangleright soloPares.xs & (2) \\ & & \Box \quad x \ mod\ 2 \neq 0 \to soloPares.xs & (3) \\ & &) & \end{array}$$

demostrá que

$$soloPares.(xs +\!\!+ ys) = soloPares.xs +\!\!+ soloPares.ys$$

[] ++ ys = ys (4)
(x:xs) ++ ys = x : (xs ++ ys) (5)

Caso base:

```
soloPares (xs ++ ys) = soloPares xs ++ soloPares ys

soloPares ([] ++ []) = soloPares [] ++ soloPares []
                ={Por (4), ys:=[]}
    soloPares [] = soloPares [] ++ soloPares []
                    ={Por (1)}
                   [] = [] ++ []
                ={Por (4), ys:=[]}
                      [] = []
              ={Reflexividad del = }
                       True
```

Caso inductivo:

Caso 1: x `mod` 2 == 0

```
soloPares (xs ++ ys) = soloPares xs ++ soloPares ys

soloPares (x:xs ++ ys) = soloPares x:xs ++ soloPares ys
                    ={Por (5)}
soloPares (x: (xs ++ ys)) = soloPares x:xs ++ soloPares ys
                    ={Por (2)}
x: (soloPares (xs ++ ys)) = x: (soloPares xs) ++ soloPares ys
                    ={Por (5)}
x: (soloPares (xs ++ ys)) = x: ((soloPares xs) ++ soloPares ys)
                    ={Por HI}
x:(soloPares xs ++ soloPares ys) = x:((soloPares xs) ++ soloPares ys)
               ={Reflexividad del =}
                       True
```

Caso 2: x `mod` 2 /= 0

```
soloPares (xs ++ ys) = soloPares xs ++ soloPares ys

soloPares (x:(xs ++ ys)) = soloPares x:xs ++ soloPares ys
={Por (5)}
soloPares (x: (xs ++ ys)) = soloPares x:xs ++ soloPares ys
={Por (3), x:=x, xs:= (xs ++ ys)}
soloPares (xs ++ ys) = soloPares x:xs ++ soloPares ys
={Por (3)}
soloPares (xs ++ ys) = soloPares xs ++ soloPares ys
={Por HI}
soloPares xs ++ soloPares ys = soloPares xs ++ soloPares ys
={Reflexividad del =}
True
```

14. Considerando la función sum : [Num] → Num que toma una lista de números y devuelve la suma de ellos, definí sum y demostrá que:

```
sum(xs ++ ys) = sum xs + sum ys
```

sum [] = 0	(1)	[] ++ ys = ys	(3)
sum (x:xs) = x + (sum xs)	(2)	(x:xs) ++ ys = x : (xs ++ ys)	(4)

Caso base:

```
sum(xs ++ ys) = sum xs + sum ys

sum([] ++ ys) = sum [] + sum ys
={Por (1) y (3)}
sum ys = 0 + sum ys
={Aritmética}
sum ys = sum ys
={Reflexividad del =}
True
```

Caso inductivo:

```
sum(xs ++ ys) = sum xs + sum ys

sum(x:xs ++ ys) = sum x:xs + sum ys
={Por (4) y (2)}
sum(x: (xs ++ ys)) = x + sum xs + sum ys
={Por (2), x:=x, xs:=(xs ++ ys)}
x + sum (xs ++ ys) = x + sum xs + sum ys
={Por HI}
x + sum xs + sum ys = x + sum xs + sum ys
={Reflexividad del =}
True
```

15. Considerando la función repetir : Nat → Num → [Num], que construye una lista de un mismo número repetido cierta cantidad de veces, definida recursivamente como:

`repetir 0 x = []`	(1)
`repetir n x = x : repetir (n-1) x`	(2)

Demostrá que

```
#(repetir n x) = n
```

#[] = 0	(3)
#(x:xs) = 1 + # xs	(4)

Caso base:

```
#(repetir n x) = n

#(repetir 0 x) = 0
={Por (1)}
#[] = 0
={Por (3)}
0 = 0
```

Caso inductivo:

```
#(repetir n x) = n

#(repetir (n+1) x) = n+1
={Por (2)}
#(x:(repetir (n+1)-1 x)) = n+1
={Aritmética}
#(x:(repetir n x)) = n+1
={Por (4), x:=x; xs:=(repetir n x)}
1 + (repetir n x) = n+1
={Por HI}
1 + n = n+1
={Aritmética}
n+1 = n+1
={Reflexividad del =}
True
```

16. Considerando la función $concat : [[A]] \to [A]$ que toma una lista de listas y devuelve la concatenación de todas ellas, definida recursivamente como:

$$concat.[\,] \doteq [\,] \quad (1)$$
$$concat.(xs \triangleright xss) \doteq xs +\!\!+ concat.xss \quad (2)$$

demostrá que $concat.(xss +\!\!+ yss) = concat.xss +\!\!+ concat.yss$

```
concat (xss ++ yss) = concat xss ++ concat yss
```

[] ++ ys = ys (3)

(x:xs) ++ ys = x : (xs ++ ys) (4)

Caso base:

```
concat (xss ++ yss) = concat xss ++ concat yss

concat ([] ++ yss) = concat [] ++ concat yss
={Por (3)}
concat yss = concat [] ++ concat yss
={Por (1)}
concat yss = [] ++ concat yss
={Por (3), ys:= (concat yss)}
concat yss = concat yss
={Reflexividad del =}
True
```

Caso inductivo:

```
concat (xss ++ yss) = concat xss ++ concat yss

concat (xs:xss ++ yss) = concat xs:xss ++ concat yss
={Por (3)}
concat (xs: (xss ++ yss)) = concat xs:xss ++ concat yss
={Por (2), xs:=xs; xss:=(xss ++ yss)}
xs ++ concat (xss ++ yss) = concat xs:xss ++ concat yss
={Por (2)}
xs ++ concat (xss ++ yss) = xs ++ concat xss ++ concat yss
={Por HI}
xs ++ concat xss ++ concat yss = xs ++ concat xss ++ concat yss
={Por reflexividad del =}
True
```

17. Considerando la función $rev : [A] \to [A]$ que toma una lista y devuelve una lista con los mismos elementos pero en orden inverso, definida recursivamente como:

$$\begin{array}{rcl} rev.[\,] & \doteq & [\,] \qquad (1) \\ rev.(x \triangleright xs) & \doteq & rev.xs \triangleleft x \qquad (2) \end{array}$$

demostrá que $rev.(xs +\!\!+ ys) = rev.ys +\!\!+ rev.xs$

```
rev (xs ++ ys) = rev ys ++ rev xs
```

[] ++ ys = ys (3)

(x:xs) ++ ys = x : (xs ++ ys) (4)

Caso base:

```
rev (xs ++ ys) = rev ys ++ rev xs

rev ([] ++ ys) = rev ys ++ rev []
={Por (3) y (1)}
rev ys = rev ys ++ []
={Por 11a, xs:=(rev ys)}
rev ys = rev ys
={Reflexivilidad del =}
True
```

Caso inductivo:

```
rev (xs ++ ys) = rev ys ++ rev xs

rev (x:xs ++ ys) = rev ys ++ rev x:xs
={Por (4)}
rev (x: (xs ++ ys)) = rev ys ++ rev x:xs
={Por (2), x:=x; xs:=(xs ++ ys)}
rev (xs ++ ys) ++ [x] = rev ys ++ rev x:xs
={Por (2)}
rev (xs ++ ys) ++ [x] = rev ys ++ rev xs ++ [x]
={Por HI}
rev ys ++ rev xs ++ [x] = rev ys ++ rev xs ++ [x]
={Reflexividad del =}
True
```