Introducción a los Algoritmos 2C 2025

Práctico 5: Inducción Matemática

Una técnica poderosa para demostrar propiedades sobre un dominio construido inductivamente, como son los naturales o las listas, es usar el principio de inducción. La idea que rige este principio consiste en demostrar dos cosas. Por un lado, verificar que la propiedad se satisface para el caso base, es decir para los elementos “más chicos” del dominio, por ejemplo la lista vacía ([ ]) para listas, o el cero para naturales. Por otro lado, demostrar que si la propiedad es cierta para un elemento cualquiera del dominio, por ejemplo xs, entonces la propiedad también es satisfecha para un elemento “más grande” construido a partir de ese elemento, por ejemplo x ▹ xs, o el sucesor en el caso de naturales.

Este procedimiento demuestra que la propiedad es satisfecha por todos los elementos del dominio, y por lo tanto es válida.

En símbolos, podríamos describir los casos que hay que demostrar de la siguiente manera:

- Si queremos demostrar la propiedad “P.n es válida”, es decir, es verdadera para todos los n naturales, tenemos que demostrar:

  1. (CASO BASE) P. 0

  2. (CASO INDUCTIVO) P.k ⇒ P.(k + 1)

- Si queremos demostrar la propiedad “P.xs es válida”, es decir, es verdadera para todos las listas xs cualquiera sean los elementos que ésta tenga, tenemos que demostrar:

  1. (CASO BASE) P.[ ]

  2. (CASO INDUCTIVO) P.ks ⇒ P.(k ▹ ks)

La implicación se puede probar, tomando como hipótesis el antecedente (llamada normalmente *Hipótesis Inductiva*), y probando el consecuente a partir de esta hipótesis y el resto de las definiciones, axiomas y teoremas conocidos.

Muchas veces a la hora de realizar una demostración es importante tener presente las definiciones y propiedades (axiomas y teoremas) de los operadores que están involucrados en el teorema que queremos demostrar.

En ese sentido, es recomendable considerar los siguientes pasos:

1. Identificar la fórmula a demostrar.

2. Recordar la definición de todos los operadores y funciones que aparecen en la fórmula.

3. Decidir sobre qué variable se hará inducción.

4. Demostrar el CASO BASE, que se obtiene al reemplazar la variable sobre la que hacemos inducción con el primer elemento del conjunto. Por ejemplo, “0” para los naturales o “[ ]” para las listas.

5. Identificar la HIPÓTESIS INDUCTIVA, que se obtiene al reemplazar la variable sobre la que hacemos inducción por otra variable que represente un valor arbitrario del conjunto. Por ejemplo “k” para los naturales o “ks” para las listas.

6. Demostrar el CASO INDUCTIVO, que se obtiene al reemplazar la variable por una expresión que represente el sucesor de la variable que elegimos para la hipótesis inductiva. Por ejemplo “k + 1” para los naturales o “k ▷ ks” para las listas.

Ejercicio 1: Considerando las definiciones de los prácticos anteriores, demostrá por inducción sobre xs las siguientes propiedades:

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

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

c) #(duplica.xs) = #xs

Ejercicio 2: Considerando las definiciones dadas en cada caso, demostrá por inducción sobre n las siguientes propiedades:

a) f.n = 2 * n

donde
f.0 ≐ 0
f.(n + 1) ≐ 2 + f.n

b) g.n = n

donde
g.0 ≐ 0
g.(n + 1) = 1 + g.n

c) sumatoria.n = n * (n + 1)/2

donde
sumatoria.0 ≐ 0
sumatoria.(n + 1) ≐ (n + 1) +
sumatoria.n

Ejercicio 3: Considerando la función quitarCeros : [Num] → [Num] definida de la siguiente manera

quitarCeros.[ ] ≐ [ ]

quitarCeros.(x ▹ xs) ≐ (x = 0 → quitarCeros.xs
□ ¬(x = 0) → x ▹ quitarCeros.xs
)

demostrá

sum.(quitarCeros.xs) = sum.xs

Ayuda: Tené en cuenta que como la función quitarCeros se define por casos, por lo tanto el caso inductivo también deberá dividirse en dos casos.

Ejercicio 4: Considerando el operador ⧺: [A] → [A] → [A], que concatena dos listas y definida recursivamente como:

[] ⧺ ys ≐ ys
(x ▹ xs) ⧺ ys ≐ x ▹ (xs ⧺ ys)

demostrá las siguientes propiedades

a) sum.(xs ⧺ ys) = sum.xs + sum.ys

b) duplica.(xs ⧺ ys) = (duplica.xs) ⧺ (duplica.ys)

c) solopares.(xs ⧺ ys) = (solopares.xs) ⧺ (solopares.ys)

Recordá que la función soloPares : [Num] → [Num] se define de la siguiente manera

soloPares.[ ] ≐ [ ]
soloPares.(x ▹ xs) ≐ (x mod 2 = 0 → x ▹ soloPares.xs
□ ¬(x mod 2 = 0) → soloPares.xs
)

Ejercicio 5: Demostrá por inducción las siguientes propiedades. Ayuda: Recordá la definición de cada uno de los operadores implicados en cada expresión.

a) xs ⧺ [ ] = xs (la lista vacía es el elemento neutro de la concatenación)

b) #xs >= 0

Ejercicio 6: 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 ≐ [ ]
repetir.(n + 1).x ≐ x ▹ repetir.n.x

demostrá

#repetir.n.x = n

Ejercicio 7: Considerando la función concat : [[A]] → [A] que toma una lista de listas y devuelve la concatenación de todas ellas, definida recursivamente como:

concat.[ ] ≐ [ ]
concat.(xs ▹ xss) ≐ xs ⧺ concat.xss

demostrá

concat.(xss ⧺ yss) = concat.xss ⧺ concat.yss

Ejercicio 8: Demostrá por inducción las siguientes propiedades. Ayuda: Recordá la definición de cada uno de los operadores implicados en cada expresión.

a) xs ⧺ (ys ⧺ zs) = (xs ⧺ ys) ⧺ zs (la concatenación es asociativa)

b) (xs ⧺ ys) ↑ #xs = xs

c) (xs ⧺ ys) ↓ #xs = ys

d) xs ⧺ (y ▹ ys) = (xs ◃ y) ⧺ ys

e) xs ⧺ (ys ◃ y) = (xs ⧺ ys) ◃ y

Ejercicio 9: Dadas las siguientes funciones

soloPares.[ ] ≐ [ ]
soloPares.(x ▹ xs) ≐ (x mod 2 = 0 → x ▹ soloPares.xs
□ ¬(x mod 2 = 0) → soloPares.xs
)
duplica.[ ] ≐ [ ]
duplica.(x ▹ xs) ≐ (2*x) ▹ duplica.xs

#[ ] ≐ 0
#(x ▹ xs) ≐ 1 + #xs

demostrá que

soloPares.(duplica.xs) = #xs

Ejercicio 10: Dada las funciones (realizar este ejercicio luego del práctico 6)

hayTr.[ ] ≐ [Figura] -> Bool
hayTr.[ ] ≐ False
hayTr.(x ▹ xs) ≐ (triangulo.x ∧ rojo.x) ∨ hayTr.xs

$\in_l$: A -> [A] -> Bool
a $\in_l$ [ ] ≐ False
a $\in_l$ (x ▹ xs) ≐ a = x ∨ a $\in_l$ xs

demostrá por inducción que

$$hayTr.ys \equiv \langle \exists\ z : z \in_{l} ys : triangulo.z \wedge rojo.z \rangle$$

Ejercicio 11: Dada las funciones

$$saca0 : [\ Int\ ] \to [\ Int\ ]$$
$$saca0.[\ ] \doteq [\ ]$$
$$saca0.(x \triangleright xs) \doteq (\ x = 0 \to saca0.xs \ \Box\ x \neq 0 \to x \triangleright saca0.xs\ )$$

$$\in_{l} : A \to [\ A\ ] \to Bool$$
$$z \in_{l} [\ ] \doteq False$$
$$z \in_{l} (x \triangleright xs\ ) \doteq z = x \vee z \in_{l} xs$$

demostrá por inducción que

$$0 \in_{l} (saca0.xs) \equiv False$$

Ejercicio 12: Dada las funciones

$$dup : [\ Int\ ] \to [\ Int\ ]$$
$$dup.[\ ] \doteq [\ ]$$
$$dup.(x \triangleright xs) \doteq (2*x) \triangleright dup.xs$$

$$mas1 : [\ Int\ ] \to [\ Int\ ]$$
$$mas1.[\ ] \doteq [\ ]$$
$$mas1.(x \triangleright xs) \doteq (\ 1 + x) \triangleright mas1.xs$$

$$sum : [\ Int\ ] \to Int$$
$$sum.[\ ] \doteq 0$$
$$sum.(x \triangleright xs) \doteq x + sum.xs$$

$$\#: [Int] \to Int$$
$$\#[\ ] = 0$$
$$\#(x \triangleright xs) = 1 + \#xs$$

demostrá por inducción que

$$sum.(mas1.\ (dup.xs)) = 2*(sum.xs) + \#xs$$

Ejercicio 13: Dada la funciones

ultimo : [A] -> A
ultimo.[a] ≐ a
ultimo.(x ▹ y ▹ xs) ≐ ultimo.(y ▹ xs)

⧺: [A] → [A] → [A]
[ ] ⧺ ys ≐ ys
(x ▹ xs) ⧺ ys ≐ x ▹ (xs ⧺ ys)

demostrá por inducción que

ultimo.(ys ⧺ (x ▹ xs)) = ultimo.(x ▹ xs)