Se proponen soluciones de ejemplo para algunos ejercicios del práctico. Las soluciones presentadas no son las únicas posibles. Pueden existir otras soluciones igualmente correctas.

Ejercicio 1

Defina inductivamente (mediante cláusulas base, inductivas y extrema), c/u de los siguientes conjuntos:

a) El conjunto **Z** de los números enteros.

Solución:

i) $0 \in Z$
ii) Si $n \in Z \Rightarrow s(n) \in Z$
iii) Si $n \in Z \Rightarrow (-n) \in Z$
iv) extrema

b) El conjunto **M** de todos los naturales múltiplos de 3.

Solución:

i) $0 \in M$
ii) Si $n \in M \Rightarrow (n + 3) \in M$
iii) extrema

c) El conjunto **O** de todos los naturales múltiplos de 4.

Solución:

i) $0 \in O$
ii) Si $n \in O \Rightarrow (n + 4) \in O$
iii) extrema

d) El conjunto **P** de todas las potencias de 2 (1, 2, 4, 8, 16, 32, 64, etc.).

Solución:

i) $1 \in P$
ii) Si $n \in P \Rightarrow (2n) \in P$
iii) extrema

e) El conjunto **J** de todos los naturales *m* tales que $m = 2n$, siendo *n* un natural impar.

Solución:

i) $2 \in J$
ii) Si $n \in J \Rightarrow (n + 4) \in J$
iii) extrema

f) El conjunto **K** de todos los naturales *p* tales que $p = n + (n/2)$ siendo *n* un natural par.

Solución:

i) $0 \in K$
ii) Si $n \in K \Rightarrow (n + 3) \in K$
iii) extrema

g) El conjunto **L** de todos los naturales x tales que $x = 4n + 1$, siendo n un natural cualquiera.

Solución:

i) $1 \in L$
ii) Si $n \in L \Rightarrow (n + 4) \in L$
iii) extrema

h) El conjunto **Q** de pares de naturales tales que $(x,y) \in Q \Leftrightarrow y = x+1$.

Solución:

iv) $(0,1) \in Q$
v) Si $(n,m) \in Q \Rightarrow (s(n), s(m)) \in Q$
vi) Extrema

i) El conjunto **R** de pares ordenados de naturales tales que $(x,y) \in R \Leftrightarrow y = 2x$.

Solución:

i) $(0,0) \in R$
ii) Si $(n,m) \in R \Rightarrow (n+1, m+2) \in R$
iii) extrema

j) El conjunto **S** de pares ordenados de naturales tales que $(x,y) \in S \Leftrightarrow y = 2^x$.

Solución:

k) $(0,1) \in S$
ii) Si $(n,m) \in S \Rightarrow (n+1, 2m) \in S$
iii) extrema

k) El conjunto **T** de pares ordenados de naturales tales que $(x,y) \in T \Leftrightarrow y = x!$.

Solución:

i) $(0,1) \in T$
ii) Si $(n,m) \in T \Rightarrow (n+1, m.(n+1)) \in T$
iii) extrema

Ejercicio 2

Dadas las siguientes funciones definidas por concordancia de patrones:

Alfa: N x N → N
Alfa (0, 0) = 1
Alfa (0, s(m)) = Alfa (0, m) + 2
Alfa (s(n), m) = Alfa (n,m) + Alfa (m,n)

Beta: Sec (A) → Sec (A)
Beta (nil) = nil
Beta (cons (x, nil)) = nil
Beta (cons (x, cons (y, s))) = cons (y, Beta (s))

a) Indique cuántos pasos base y cuántos pasos recursivos tiene cada una.

Solución:

La función Alfa tiene un paso base (la primera igualdad) y dos pasos recursivos (la segunda y la tercera igualdad). La función beta tiene dos pasos base (las primeras dos igualdades) y un paso recursivo (la tercera igualdad).

b) Calcule Alfa(2,1) aplicando paso a paso la definición de la función Alfa.

Solución:

Alfa (2,1) = Alfa (1,1) + Alfa (1,1)	(por paso recursivo 2)
Alfa (1,1) = Alfa (0,1) + Alfa (1,0)	(por paso recursivo 2)
Alfa (0,1) = Alfa (0, 0) + 2	(por paso recursivo 1)
Alfa (1,0) = Alfa (0,0) + Alfa (0,0)	(por paso recursivo 2)
Alfa (0, 0) = 1	(por paso base)

Como ya conocemos el valor de Alfa (0, 0), completamos el cálculo de Alfa (1,0).
Alfa (1,0) = Alfa (0, 0) + Alfa (0, 0) = 1 + 1 = 2.

Como ya conocemos el valor de Alfa (0, 0), completamos el cálculo de Alfa (0,1).
Alfa (0,1) = Alfa (0, 0) + 2 = 1 + 2 = 3.

Como ya conocemos el valor de Alfa (0,1) y Alfa (1,0), completamos el cálculo de Alfa(1,1).
Alfa (1,1) = Alfa (0,1) + Alfa (1,0) = 3 + 2 = 5.

Como ya conocemos el valor de Alfa (1,1), completamos el cálculo de Alfa (2,1).
Alfa(2,1) = Alfa (1,1) + Alfa (1,1) = 5 + 5 = 10.

c) Calcule Beta ([5, 7, 1, 6, 3]) aplicando paso a paso la definición de la función Beta.

Solución:

Beta ([5, 7, 1, 6, 3]) = Beta (cons (5, cons (7, [1, 6, 3]))) = cons (7, Beta ([1, 6, 3]))	(por paso recursivo)
Beta ([1, 6, 3]) = Beta (cons (1, cons (6, [3]))) = cons (6, Beta ([3]))	(por paso recursivo)
Beta ([3]) = Beta (cons (3, nil)) = nil	(por paso base 2)

Como ya conocemos el valor de Beta ([3]), completamos el cálculo de Beta ([1, 6, 3]).
Beta ([1, 6, 3]) = cons (6, Beta ([3])) = cons (6, nil) = [6].

Como ya conocemos el valor de Beta ([1, 6, 3]), completamos el cálculo de Beta ([5, 7, 1, 6, 3]).
Beta ([5, 7, 1, 6, 3]) = cons (7, Beta ([1, 6, 3])) = cons (7, Beta ([6])) = [7, 6].

d) ¿Son funciones totales o parciales? Justifique su respuesta.

Solución:

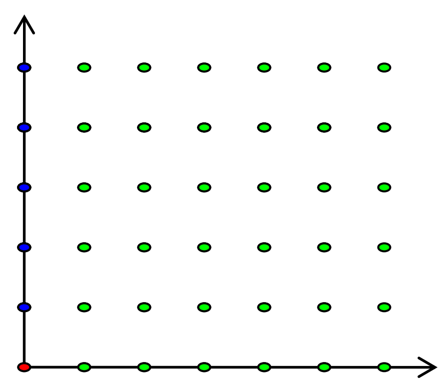
La función Alfa es total.

Para el valor **(0,0)**, el paso base produce un resultado.

Para los valores de la forma **(0, s(m))**, el primer paso recursivo produce un resultado.

Para los valores de la forma **(s(n), m)**, el segundo paso recursivo produce un resultado.

Por lo tanto, para cualquier elemento de NxN, la función Alfa produce un resultado, de donde concluimos que es total.



La función Beta también es total.

Para la secuencia vacía, el primer paso base produce un resultado.

Para cualquier secuencia con un solo elemento, el segundo paso base produce un resultado.

Para cualquier otra secuencia (con dos o más elementos), el paso recursivo produce un resultado.

Por lo tanto, para cualquier secuencia, la función Beta produce un resultado, concluimos que es total.

Ejercicio 3

Defina por concordancia de patrones las siguientes funciones sobre números naturales. Indique además si son totales, inyectivas, sobreyectivas, justificando sus respuestas.

a) ***EsPar: N → Boolean*** devuelve true si el número es par, false en caso contrario. Se pide definir esta función sin utilizar el operador módulo (resto de la división).

Solución:

EsPar (0) = true
EsPar (1) = false
EsPar (s(s(n))) = EsPar (n)

La función **es** total, de acuerdo con el siguiente análisis de casos:

- Para el natural 0, el primer paso base produce el resultado.
- Para el natural 1, el segundo paso base produce el resultado.
- Para cualquier natural del 2 en adelante, el paso recursivo produce el resultado.

En conclusión, dado cualquier natural, la función produce un resultado, lo que la hace total.

La función **no** es inyectiva, dado que existen naturales diferentes que tienen la misma imagen. Por ejemplo, EsPar (1) = false y EsPar (3) = false.

La función **es** sobreyectiva, dado que los únicos dos valores del codominio (false y true) tienen ambos al menos una preimagen. Por ejemplo, EsPar (0) = true y EsPar (1) = false.

b) ***SumaHastaN: N → N*** dado n∈N, devuelve la suma de todos los naturales entre 0 y n.

Solución:

SumaHastaN (0) = 0
SumaHastaN (s(n)) = s(n) + SumaHastaN (n)

La función **es** total, de acuerdo con el siguiente análisis de casos:

- Para el natural 0, el paso base produce el resultado.
- Para cualquier natural del 1 en adelante, el paso recursivo produce el resultado.

En conclusión, dado cualquier natural, la función produce un resultado, lo que la hace total.

La función **es** inyectiva. La función devuelve la suma de todos los naturales menores o iguales a un natural dado. Dados dos naturales *n*, *m*, si *n* < *m*, entonces la suma de todos los naturales hasta *n* será menor que la suma de todos los naturales hasta *m*.

La función **no** es sobreyectiva. Existe al menos un valor del codominio que no tiene preimagen. Por ejemplo, 2 no tiene preimagen, ya que SumaHastaN (0) = 0, SumaHastaN (1) = 1, SumaHastaN (2) = 3 y SumaHastaN (*m*), siendo *m* > 2, devolverá un resultado aún mayor.

c) ***SumaPares: N → N*** dado n∈N, devuelve la suma de todos los naturales pares entre 0 y n.

Solución:

SumaPares (0) = 0
SumaPares (s(n)) = Si EsPar (s(n))
        s(n) + SumaPares (n)
    Sino
        SumaPares (n)

La función **<u>es</u>** total, de acuerdo con el siguiente análisis de casos:

- Para el natural 0, el paso base produce el resultado.
- Para cualquier natural del 1 en adelante, el paso recursivo produce el resultado.

En conclusión, dado cualquier natural, la función produce un resultado, lo que la hace total.

La función **<u>no</u>** es inyectiva, dado que existen naturales diferentes que tienen la misma imagen. Por ejemplo, SumaPares (0) = 0 y SumaPares (1) = 0.

La función **<u>no</u>** es sobreyectiva, dado que ningún natural impar tiene preimagen en la función, ya que la misma devuelve como resultado una suma de valores pares, que nunca puede dar impar. Por ejemplo, 7 no tiene preimagen, ya que ninguna suma de naturales pares puede dar como resultado 7.

d) ***Potencia: N×N → N*** dados dos naturales n, m devuelve el resultado de elevar n^m. Se pide definir esta función sin utilizar el operador de potencia.

<u>Solución</u>:

Potencia (0, s(m)) = 0
Potencia (s(n), 0) = 1
Potencia (s(n), s(m)) = s(n) * Potencia (s(n), m)

La función **<u>no</u>** es total, dado que para el par (0,0) no hay ningún patrón que produzca un resultado. Esto es porque no se puede calcular el resultado de 0^0.

La función **<u>no</u>** es inyectiva, dado que existen parejas de naturales diferentes que tienen la misma imagen. Por ejemplo, Potencia (2,4) = 16 y Potencia (4,2) = 16.

La función **<u>es</u>** sobreyectiva, dado que todo natural del codominio tiene al menos una preimagen ya que puede calcularse como resultado de elevarlo a 1. Dado m∈N cualquiera, se cumple Potencia (m,1) = m.

e) ***Max: N×N → N*** dados dos naturales n, m devuelve el valor más grande de entre ellos dos. Se pide definir esta función sin utilizar resta ni operadores de comparación.

<u>Solución</u>:

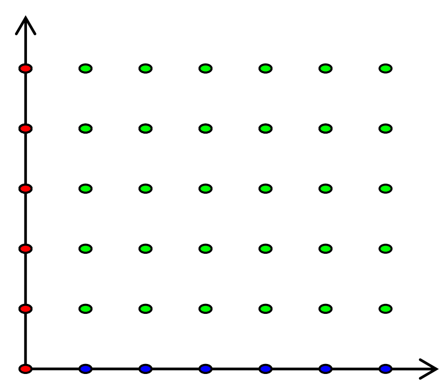
Max (0, m) = m
Max (s(n), 0) = s(n)
Max (s(n), s(m)) = 1 + max (n, m)

La función **<u>es</u>** total, dado que para cualquier pareja de naturales siempre es posible determinar el mayor.

Para los valores de la forma **(0, m)**, el 1° paso base produce un resultado.

Para los valores de la forma **(s(n), 0)**, el 2º paso base produce un resultado.

Para los valores de la forma **(s(n), s(m))**, el paso recursivo produce un resultado.



La función **<u>no</u>** es inyectiva, dado que existen parejas de naturales diferentes que tienen la misma imagen. Por ejemplo, Max (2,4) = 4 y Max (4,2) = 4.

La función **<u>es</u>** sobreyectiva, dado que todo natural del codominio tiene al menos una preimagen ya que es el máximo entre sí mismo y cero. Dado m∈N cualquiera, se cumple que Max (m,0) = m.

f) ***SumaCuad: N → N*** dado un natural n, devuelve la suma de los cuadrados de todos los naturales que hay entre n y 0. Por ejemplo: SumCuad (3) = $3^2 + 2^2 + 1^2 + 0^2$.

<u>Solución</u>:

SumaCuad (0) = 0
SumaCuad (s(n)) = $s(n)^2$ + SumaCuad (n)

La función **<u>es</u>** total, de acuerdo con el siguiente análisis de casos:

- Para el natural 0, el paso base produce el resultado.
- Para cualquier natural del 1 en adelante, el paso recursivo produce el resultado.

En conclusión, dado cualquier natural, la función produce un resultado, lo que la hace total.

La función **<u>es</u>** inyectiva. La función devuelve la suma de todos los cuadrados de naturales menores o iguales a un natural dado. Dados dos naturales *n*, *m*, si *n* < *m*, entonces la suma de todos los cuadrados de naturales hasta *n* será menor que la suma de todos los cuadrados de naturales hasta *m*.

La función **<u>no</u>** es sobreyectiva. Existe al menos un valor del codominio que no tiene preimagen. Por ejemplo, 2 no tiene preimagen, ya que SumaCuad (0) = 0, SumaCuad (1) = 1, SumaCuad (2) = 5 y SumaCuad (*m*), siendo *m* > 2, devolverá un resultado aún mayor.

g) ***SumaPots2: N → N*** dado un natural n, devuelve la suma de todas las potencias de 2 que hay entre 0 y n. Por ejemplo: SumPot (3) = $2^3 + 2^2 + 2^1 + 2^0$. Para resolverlo, hacer uso de la función *Potencia* definida en la parte (d).

<u>Solución</u>:

SumaPots2 (0) = 1
SumaPots2 (s(n)) = Potencia (2, s(n)) + SumaPots2 (n)

La función **<u>es</u>** total, de acuerdo con el siguiente análisis de casos:

- Para el natural 0, el paso base produce el resultado.
- Para cualquier natural del 1 en adelante, el paso recursivo produce el resultado.

En conclusión, dado cualquier natural, la función produce un resultado, lo que la hace total.

La función **<u>es</u>** inyectiva. La función devuelve la suma de todas las potencias de 2 menores o iguales a un natural dado. Dados dos naturales *n*, *m*, si *n* < *m*, entonces la suma de todas las potencias de 2 de naturales hasta *n* será menor que la suma de todas las potencias de 2 de naturales hasta *m*.

La función **<u>no</u>** es sobreyectiva. Existe al menos un valor del codominio que no tiene preimagen. Por ejemplo, 0 no tiene preimagen, ya que SumaPots2 (0) = 1 y SumaPots2 (*m*), siendo *m* > 0, devolverá un resultado aún mayor.

Ejercicio 4

Defina por concordancia de patrones las siguientes funciones sobre secuencias. Indique además si son totales, inyectivas, sobreyectivas, justificando sus respuestas.

a) ***Esvacia: Sec(A) → Boolean*** devuelve true si y sólo si la secuencia es vacía.

<u>Solución</u>:

EsVacía: Sec(A) → Bolean
EsVacía (nil) = true
EsVacía (cons (x,s)) = false

La función **<u>es</u>** total, dado que para cualquier secuencia es posible calcular un resultado. Para la secuencia vacía, el primer patrón calcula el resultado, mientras que para cualquier secuencia no vacía, es el segundo patrón quien lo hace.

La función **no** es inyectiva, dado que existen secuencias diferentes que tienen la misma imagen. Por ejemplo, EsVacía ([2, 4]) = false y EsVacía ([3, 7]) = false.

La función **es** sobreyectiva, dado que los únicos dos valores del codominio (false y true) tienen ambos al menos una preimagen. Por ejemplo, EsVacía (nil) = true y EsVacía ([2, 4]) = false.

b) ***Primero: Sec(A) → A*** dada una secuencia no vacía, devuelve su primer elemento.

Solución:

Primero: Sec(A) → A
Primero (cons (x,s)) = x

La función **no** es total, dado que para la secuencia vacía, no hay ningún patrón que permita calcular el resultado. Esto es porque es imposible devolver el primer elemento de una secuencia que no tiene ningún elemento.

La función **no** es inyectiva, dado que existen secuencias diferentes que tienen la misma imagen. Por ejemplo, Primero ([2, 4]) = 2 y Primero ([2, 3, 7]) = 2.

La función **es** sobreyectiva, dado que para cualquier elemento $a \in A$, siempre existe al menos una secuencia que lo tiene como primer elemento. Por ejemplo, Primero (cons (a, nil)) = a.

c) ***Resto: Sec(A) → Sec(A)*** dada una secuencia no vacía, devuelve la secuencia que resulta de quitarle su primer elemento.

Solución:

Resto: Sec(A) → A
Resto (cons (x,s)) = s

La función **no** es total, dado que para la secuencia vacía, no hay ningún patrón que permita calcular el resultado. Esto es porque es imposible quitar el primer elemento de una secuencia que no tiene ningún elemento.

La función **no** es inyectiva, dado que existen secuencias diferentes que tienen la misma imagen. Por ejemplo, Resto ([2, 4, 5]) = [4, 5] y Resto ([3, 4, 5]) = [4, 5].

La función **es** sobreyectiva, dado que para cualquier secuencia $s \in Sec(A)$, siempre existe otra secuencia que la tiene como resto. Por ejemplo, dado $a \in A$, Resto (cons (a, s)) = s.

d) ***ContarDistintos: Sec(A) x A → N*** dada una secuencia y un elemento, cuenta la cantidad de ocurrencias de elementos en la secuencia que son diferentes al elemento dado.

Solución:

ContarDistintos (nil, y) = 0
ContarDistintos (cons (x,s), y) = Si ($x \neq y$)
    1 + ContarDistintos (s, y)
  Sino
    ContarDistintos (s, y)

La función **es** total, de acuerdo con el siguiente análisis de casos:

- Cuando la secuencia es vacía, el primer patrón produce el resultado.
- Cuando la secuencia es no vacía, el segundo patrón produce el resultado.

En conclusión, dados cualquier secuencia y cualquier elemento, la función produce un resultado, lo que la hace total.

La función **no** es inyectiva, dado que existen secuencias diferentes que tienen la misma imagen. Por ejemplo, ContarDistintos ([3, 4, 5], 3) = 2 y ContarDistintos ([7, 3, 8], 3) = 2.

La función **es** sobreyectiva, dado que para cualquier natural n, siempre existe alguna secuencia que tiene n elementos distintos de algún elemento dado.

e) ***ContarPosImpares: Sec(A) → N*** dada una secuencia, cuenta la cantidad de elementos que ocupan posiciones impares en ella (suponga que las posiciones se numeran a partir de 1).

Solución:

ContarPosImpares (nil) = 0
ContarPosImpares (cons (x, nil)) = 1
ContarPosImpares (cons (x, cons (y, s))) = 1 + ContarPosImpares (s)

La función **es** total, de acuerdo con el siguiente análisis de casos:

- Cuando la secuencia es vacía, el primer patrón produce el resultado.
- Cuando la secuencia tiene un solo elemento, el segundo patrón produce el resultado.
- Cuando la secuencia tiene dos o más elementos, el tercer patrón produce el resultado.

En conclusión, dada cualquier secuencia, la función produce un resultado, lo que la hace total.

La función **no** es inyectiva, dado que existen secuencias diferentes que tienen la misma imagen. Por ejemplo, ContarPosImpares ([3, 4, 5]) = 2 y ContarPosImpares ([3, 7, 2]) = 2.

La función **es** sobreyectiva, dado que para cualquier natural n, siempre existe alguna secuencia que tiene n elementos que ocupan posiciones impares en ella.

f) ***Pertenece: Sec(A) x A → Boolean*** dada una secuencia y un elemento, devuelve true si y sólo si el elemento recibido pertenece a la secuencia, false en caso contrario.

Solución:

```
Pertenece (nil, e) = false
Pertenece (cons (x,s), y) =  Si (x = y)
                                 true
                             Sino
                                 Pertenece (s, y)
```

La función **es** total, de acuerdo con el siguiente análisis de casos:

- Cuando la secuencia es vacía, el primer patrón produce el resultado.
- Cuando la secuencia es no vacía, el segundo patrón produce el resultado.

En conclusión, dados cualquier secuencia y cualquier elemento, la función produce un resultado, lo que la hace total.

La función **no** es inyectiva, dado que existen secuencias diferentes que tienen la misma imagen. Por ejemplo, Pertenece ([3, 4, 5], 4) = true y Pertenece ([2, 1, 5], 5) = true.

La función **es** sobreyectiva, dado que los únicos dos valores del codominio (false y true) tienen ambos al menos una preimagen. Por ejemplo, Pertenece (nil, 5) = false y Pertenece ([3, 4, 5], 4) = true.

g) ***TodosIguales: Sec(A) → Boolean*** dada una secuencia y un elemento, devuelve true si y sólo si todos sus elementos son iguales entre sí, false en caso contrario.

Solución:

```
TodosIguales (nil) = true
TodosIguales (cons (x, nil)) =  true
TodosIguales (cons (x, cons (y, s))) =  Si (x ≠ y)
                                            false
                                        Sino
                                            TodosIguales (cons (y, s))
```

La función **es** total, de acuerdo con el siguiente análisis de casos:

- Cuando la secuencia es vacía, el primer patrón produce el resultado.
- Cuando la secuencia tiene un solo elemento, el segundo patrón produce el resultado.
- Cuando la secuencia tiene dos o más elementos, el tercer patrón produce el resultado.

En conclusión, dada cualquier secuencia, la función produce un resultado, lo que la hace total.

La función **no** es inyectiva, dado que existen secuencias diferentes que tienen la misma imagen. Por ejemplo, TodosIguales ([3, 3, 3]) = true y TodosIguales ([1, 1]) = true.

La función **es** sobreyectiva, dado que los únicos dos valores del codominio (false y true) tienen ambos al menos una preimagen. Por ejemplo, TodosIguales (nil) = true y TodosIguales ([3, 4, 4]) = false.

h) ***SecIguales: Sec(A) x Sec(A) → Boolean*** dadas dos secuencias, determina si son iguales.

Solución:

SecIguales (nil, nil) = true
SecIguales (nil, cons (y, t)) = false
SecIguales (cons (x, s), nil) = false
SecIguales (cons (x, s), cons (y, t)) = Si (x = y)
    SecIguales (s, t)
  Sino
    false

La función **es** total, de acuerdo con el siguiente análisis de casos:

- Cuando las dos secuencias son vacías, el primer patrón produce el resultado.
- Cuando la 1º secuencia es vacía y la 2º no, el segundo patrón produce el resultado.
- Cuando la 1º secuencia es no vacía y la 2º sí, el tercer patrón produce el resultado.
- Cuando ambas secuencias son no vacías, el cuarto patrón produce el resultado.

En conclusión, dada cualquier pareja de secuencias, la función produce un resultado, lo que la hace total.

La función **no** es inyectiva, dado que existen parejas de secuencias diferentes que tienen la misma imagen. Por ejemplo, SecIguales ([3, 4], [3, 4]) = true y SecIguales ([2, 1], [2, 1]) = true.

La función **es** sobreyectiva, dado que los únicos dos valores del codominio (false y true) tienen ambos al menos una preimagen. Por ejemplo, SecIguales (nil, nil) = true y SecIguales (nil, [3, 4]) = false.

i) ***Elemsec: Sec(A) x N → A*** dada una secuencia y un número natural *n* > 0, devuelve el elemento que ocupa la posición *n* de la secuencia (suponga que las posiciones se numeran a partir de 1).

Solución:

ElemSec (cons (x, s), 1) = x
ElemSec (cons (x, s), s(s(n))) = ElemSec (s, s(n))

La función **no** es total, dado que para la secuencia vacía, no hay ningún patrón que permita calcular el resultado. Esto es porque es imposible devolver el primer elemento de una secuencia que no tiene ningún elemento.

La función **no** es inyectiva, dado que existen secuencias diferentes que tienen la misma imagen. Por ejemplo, ElemSec ([3, 4, 5], 2) = 4 y ElemSec ([7, 2, 4], 3) = 4.

La función **es** sobreyectiva, dado que para cualquier elemento a∈A, siempre existe al menos una secuencia que lo tiene como primer elemento. Por ejemplo, ElemSec (cons (a, nil), 1) = a.

j) ***Ultimo: Sec(A) → A*** dada una secuencia no vacía, devuelve su último elemento.

<u>Solución</u>:

Ultimo (cons (x, nil)) = x

Ultimo (cons (x, cons (y, s))) = Ultimo (cons (y, s))

La función **<u>no</u>** es total, dado que para la secuencia vacía, no hay ningún patrón que permita calcular el resultado. Esto es porque es imposible devolver el último elemento de una secuencia que no tiene ningún elemento.

La función **<u>no</u>** es inyectiva, dado que existen secuencias diferentes que tienen la misma imagen. Por ejemplo, Ultimo ([2, 5]) = 5 y Ultimo ([4, 9, 5]) = 5.

La función **<u>es</u>** sobreyectiva, dado que para cualquier elemento a∈A, siempre existe al menos una secuencia que lo tiene como último elemento. Por ejemplo, Ultimo (cons (b, cons (a, nil))) = a.

k) ***BorrarUltimo: Sec(A) → Sec(A)*** dada una secuencia no vacía, devuelve la secuencia resultante de borrarle su último elemento.

<u>Solución</u>:

BorrarUltimo (cons (x, nil)) = nil

BorrarUltimo (cons (x, cons (y, s))) = cons (x, BorrarUltimo (cons (y, s)))

La función **<u>no</u>** es total, dado que para la secuencia vacía, no hay ningún patrón que permita calcular el resultado. Esto es porque es imposible borrar el último elemento de una secuencia que no tiene ningún elemento.

La función **<u>no</u>** es inyectiva, dado que existen secuencias diferentes que tienen la misma imagen. Por ejemplo, BorrarUltimo ([3, 2, 5]) = [3, 2] y BorrarUltimo ([3, 2, 7]) = [3, 2].

La función **<u>es</u>** sobreyectiva, dado que para cualquier secuencia s∈Sec(A), siempre existe otra secuencia que, tras borrarle su último elemento, dé como resultado s. Por ejemplo, cualquier secuencia que tenga los mismos elementos que s, más un elemento cualquiera adicional luego de todos ellos.

l) ***Tomar: Sec(A) x N → Sec(A)*** dada una secuencia y un número natural *n*, devuelve otra secuencia que resulta de tomar los primeros *n* elementos de la secuencia.

<u>Solución</u>:

Tomar (s, 0) = nil

Tomar (nil, s(n)) = nil

Tomar (cons (x,s), s(n)) = cons (x, Tomar (s, n))

La función **<u>es</u>** total, de acuerdo con el siguiente análisis de casos:

- Para cualquier secuencia y el natural 0, el primer patrón produce el resultado.
- Para la secuencia vacía y cualquier natural del 1 en adelante, el segundo patrón produce el resultado.
- Para una secuencia no vacía y cualquier natural del 1 en adelante, el tercer patrón produce el resultado.

En conclusión, dados cualquier natural y cualquier secuencia, la función produce un resultado, lo que la hace total.

La función **<u>no</u>** es inyectiva, dado que existen secuencias diferentes que tienen la misma imagen. Por ejemplo, Tomar ([4, 3, 5], 2) = [4, 3] y Tomar ([4, 3, 7, 9], 2) = [4, 3].

La función **<u>es</u>** sobreyectiva, dado que para cualquier secuencia s∈Sec(A), siempre existe otra secuencia que, tras tomar sus primeros n elementos, dé como resultado s. Por ejemplo, cualquier secuencia que tenga los mismos elementos que s, más uno o más elementos cualesquiera luego de todos ellos.

m) ***Borrar: Sec(A) x N → Sec(A)*** Dada una secuencia y un número natural *n*, devuelve otra secuencia que resulta de borrar los primeros *n* elementos de la secuencia.

Solución:

Borrar (s, 0) = s
Borrar (nil, s(n)) = nil
Borrar (cons (x,s), s(n)) = Borrar (s, n)

La función **es** total, de acuerdo con el siguiente análisis de casos:

- Para cualquier secuencia y el natural 0, el primer patrón produce el resultado.
- Para la secuencia vacía y cualquier natural del 1 en adelante, el segundo patrón produce el resultado.
- Para una secuencia no vacía y cualquier natural del 1 en adelante, el tercer patrón produce el resultado.

En conclusión, dados cualquier natural y cualquier secuencia, la función produce un resultado, lo que la hace total.

La función **no** es inyectiva, dado que existen secuencias diferentes que tienen la misma imagen. Por ejemplo, Borrar ([4, 3, 5, 7], 2) = [5, 7] y Borrar ([9, 5, 7], 1) = [5, 7].

La función **es** sobreyectiva, dado que para cualquier secuencia $s \in Sec(A)$, siempre existe otra secuencia que, tras borrar sus primeros n elementos, dé como resultado s. Por ejemplo, cualquier secuencia que tenga los mismos elementos que s, más n elementos cualesquiera antes de todos ellos.

n) ***FiltrarDistintos: Sec(A) x A → Sec(A)*** dada una secuencia y un elemento, devuelve otra secuencia formada por aquellos elementos en la secuencia original que son diferentes al elemento dado.

Solución:

FiltrarDistintos (nil, y) = nil
FiltrarDistintos (cons (x,s), y) = Si ($x \neq y$)
    cons (x, FiltrarDistintos (s, y))
  Sino
    FiltrarDistintos (s, y)

La función **es** total, de acuerdo con el siguiente análisis de casos:

- Cuando la secuencia es vacía, el primer patrón produce el resultado.
- Cuando la secuencia es no vacía, el segundo patrón produce el resultado.

En conclusión, dados cualquier secuencia y cualquier elemento, la función produce un resultado, lo que la hace total.

La función **no** es inyectiva, dado que existen secuencias diferentes que tienen la misma imagen. Por ejemplo, FiltrarDistintos ([4, 3, 5, 7], 3) = [4, 5, 7] y FiltrarDistintos ([9, 4, 5, 7], 9) = [4, 5, 7].

La función **es** sobreyectiva, dado que para cualquier secuencia $s \in Sec(A)$, siempre existe otra secuencia que tenga los mismos elementos que s más un elemento adicional distinto a todos ellos al comienzo. Tras filtrar dicho elemento distinto, el resultado será s.

o) ***InsBack: Sec(A) x A → Sec(A)*** dadas una secuencia y un valor, inserta el nuevo valor al final de la secuencia.

Solución:

InsBack (nil, e) = cons (e, nil)
InsBack (cons (x,s), e) = cons (x, InsBack (s, e))

La función **es** total, de acuerdo con el siguiente análisis de casos:

- Para la secuencia vacía y sin importar el valor de *e*, el primer patrón produce el resultado.
- Para cualquier secuencia no vacía y sin importar el valor de *e*, el segundo patrón produce el resultado.

En conclusión, dada cualquier secuencia, la función produce un resultado, lo que la hace total.

La función **es** inyectiva, dado que si a dos secuencias diferentes cualesquiera les insertamos el mismo valor al final, las secuencias resultantes seguirán siendo diferentes.

La función **no** es sobreyectiva, dado que existe una secuencia en el codominio que no tiene preimagen. Concretamente, la secuencia vacía (nil) nunca puede ser el resultado de insertarle un elemento al final a alguna otra secuencia.

p) ***Repetidos: Sec(A) → Boolean*** dada una secuencia, determina si en ella hay o no algún valor repetido.

Solución:

Repetidos (nil) = false
Repetidos (cons (x,s)) = Si Pertenece (s, x)
true
Sino
Repetidos (s)

La función es total, de acuerdo con el siguiente análisis de casos:

- Para la secuencia vacía, el primer patrón produce el resultado.
- Para cualquier secuencia no vacía, el segundo patrón produce el resultado.

En conclusión, dada cualquier secuencia, la función produce un resultado, lo que la hace total.

La función **no** es inyectiva, dado que existen secuencias diferentes que tienen la misma imagen. Por ejemplo, Repetidos ([4, 3, 5]) = false y Repetidos ([9, 4, 5, 7]) = false.

La función **es** sobreyectiva, dado que los únicos dos valores del codominio (false y true) tienen ambos al menos una preimagen. Por ejemplo, Repetidos (nil) = false y Repetidos ([3, 4, 4]) = true.

Ejercicio 5

Demuestre las siguientes propiedades por inducción estructural en los naturales:

a) $\forall n \in N$ se cumple que (2n + 1) es impar.

Solución:

Paso base: Debemos probar que (2.0 + 1) es impar.

Demostración:

2.0 + 1 = 1 que es impar.

Paso inductivo: HI) (2n + 1) es impar.
TI) (2.s(n) + 1) es impar.

Demostración:

(2.s(n) + 1) = 2(n+1) + 1 (por definición de s(n))

2(n+1) + 1 = 2n + 2 + 1 (por propiedad distributiva)

2n + 2 + 1 = (2n + 1) + 2 (por propiedad conmutativa)

Por hipótesis inductiva, sabemos que (2n + 1) es impar. Luego (2n + 1) + 2 también lo es, porque si a un número impar le sumamos 2, el resultado sigue siendo impar.

b) $\forall n \in N$ se cumple que $2^n \leq 3^n$.

Solución:

Paso base: Debemos probar que $2^0 \leq 3^0$.

Demostración:

Tanto 2^0 como 3^0 dan como resultado 1, y sabemos que $1 \leq 1$.

Paso inductivo: HI) $2^n \leq 3^n$.
TI) $2^{s(n)} \leq 3^{s(n)}$.

Demostración:

$$2^{s(n)} \underset{(1)}{=} 2^{n+1} \underset{(2)}{=} 2^n.2 \underset{(3)}{\leq} 3^n.2 \underset{(4)}{\leq} 3^n.3 \underset{(2)}{=} 3^{n+1} \underset{(1)}{=} 3^{s(n)}$$

(1) por definición de s(n).

(2) por definición de potencia.

(3) porque $2^n \leq 3^n$ por hipótesis inductiva.

(4) porque $2 \leq 3$.

c) $\forall n \in N$ se cumple que $(5^n - 1)$ es múltiplo de 4.

Solución:

Paso base: Debemos probar que $(5^0 - 1)$ es múltiplo de 4.

Demostración:

$5^0 - 1 = 1 - 1 = 0$, que es múltiplo de 4.

Paso inductivo: HI) $(5^n - 1)$ es múltiplo de 4.
TI) $(5^{s(n)} - 1)$ es múltiplo de 4.

Demostración:

$5^{s(n)} - 1 = 5^{n+1} - 1$ (por definición de s(n))

$5^{n+1} - 1 = 5^n.5 - 1$ (por definición de potencia)

$5^n.5 - 1 = 5^n.(4 + 1) - 1$ (pues $5 = 4 + 1$)

$5^n.(4 + 1) – 1 = 5^n.4 + 5^n – 1$ (por propiedad distributiva)

Por hipótesis inductiva, sabemos que $(5^n - 1)$ es múltiplo de 4. También sabemos que $5^n.4$ es múltiplo de 4 (porque está multiplicado por 4). Luego, la suma de dos múltiplos de 4 es también múltiplo de 4.

d) $\forall n \in N$ se cumple que $(n^3 - n)$ es múltiplo de 3.

Solución:

Paso base: Debemos probar que $(0^3 - 0)$ es múltiplo de 3.

Demostración:

$0^3 - 0 = 0 – 0 = 0$, que es múltiplo de 3.

Paso inductivo: HI) $(n^3 - n)$ es múltiplo de 3.
TI) $(s(n)^3 - s(n))$ es múltiplo de 3.

Demostración:

$s(n)^3 - s(n) = (n + 1)^3 - (n + 1)$ (por definición de s(n))

$(n + 1)^3 - (n + 1) = (n^3 + 3n^2 + 3n + 1) - (n + 1)$ (elevamos n+1 al cubo)

$(n^3 + 3n^2 + 3n + 1) - (n + 1) = (n^3 - n) + 3(n^2 + n)$ (tras hacer operaciones)

Por hipótesis inductiva, sabemos que (n^3 - n) es múltiplo de 3. También sabemos que $3(n^2 + n)$ es múltiplo de 3 (porque está multiplicado por 3). Luego, la suma de dos múltiplos de 3 es también múltiplo de 3.

e) $\forall n \in N$, se cumple que (4^n - 1) es múltiplo de 3.

Solución:

Paso base: Debemos probar que (4^0 - 1) es múltiplo de 3.

Demostración:

$4^0 - 1 = 1 - 1 = 0$, que es múltiplo de 3.

Paso inductivo: HI) (4^n - 1) es múltiplo de 3.

TI) ($4^{s(n)}$ - 1) es múltiplo de 3.

Demostración:

$4^{s(n)} - 1 = 4^{n+1} - 1$ (por definición de s(n))

$4^{n+1} - 1 = 4^n.4 - 1$ (por definición de potencia)

$4^n.4 - 1 = 4^n.(3 + 1) - 1$ (pues 4 = 3 + 1)

$4^n.(3 + 1) - 1 = 4^n.3 + 4^n - 1$ (por propiedad distributiva)

Por hipótesis inductiva, sabemos que (4^n - 1) es múltiplo de 3. También sabemos que $4^n.3$ es múltiplo de 3 (porque está multiplicado por 3). Luego, la suma de dos múltiplos de 3 es también múltiplo de 3.

f) $\forall n \in N$, $\forall m \in N$ se cumple que $a^n \text{ x } a^m = a^{(n+m)}$ (siendo $a \neq 0$).

Solución:

En este caso es indistinto realizar inducción en n o en m. Optamos por hacerlo en n.

Paso base: Debemos probar que $a^0 \text{ x } a^m = a^{(0+m)}$.

Demostración:

$a^0 \text{ x } a^m = 1 \text{ x } a^m$ (porque $a^0 = 1$).

$1 \text{ x } a^m = a^m$ (porque multiplicamos por 1).

$a^m = a^{(0+m)}$ (porque sumamos 0 en el exponente).

Paso inductivo: HI) $a^n \text{ x } a^m = a^{(n+m)}$.

TI) $a^{s(n)} \text{ x } a^m = a^{(s(n) + m)}$.

Demostración:

$a^{s(n)} \text{ x } a^m = a^{n+1} \text{ x } a^m$ (por definición de s(n))

$a^{n+1} \text{ x } a^m = a^n.a \text{ x } a^m$ (por definición de potencia)

$a^n.a \text{ x } a^m = a^n.\ a^m \text{ x } a$ (por conmutativa de la multiplicación)

$a^n.\ a^m \text{ x } a = a^{n+m} \text{ x } a$ (por hipótesis inductiva)

$a^{n+m} \text{ x } a = a^{n+m+1}$ (por definición de potencia)

$a^{n+m+1} = a^{(s(n)+m)}$. (por definición de s(n))

Si seguimos toda la cadena de igualdades, concluimos que $a^{s(n)} \text{ x } a^m = a^{(s(n) + m)}$.

g) $\sum_{i=0}^{n} i = n(n+1)/2$

<u>Solución</u>:

Paso base: Debemos probar que $\sum_{i=0}^{0} i = 0(0+1)/2$

<u>Demostración</u>:

$\sum_{i=0}^{0} i = 0 = 0(0+1)/2$, por lo que se cumple el paso base

Paso inductivo: HI) $\sum_{i=0}^{n} i = n(n+1)/2$

TI) $\sum_{i=0}^{s(n)} i = s(n)(s(n)+1)/2$

<u>Demostración</u>:

$\sum_{i=0}^{s(n)} i = (1 + 2 + 3 + \ldots + n) + s(n) =$	(por definición de sumatoria hasta n)
$\sum_{i=0}^{n} i \quad + \quad s(n) \; =$	(por hipótesis inductiva)
$n(n+1)/2 + s(n) =$	(por definición de s(n))
$n(n+1)/2 + (n + 1) =$	(sacamos n+1 de factor común)
$(n+1)(n / 2 + 1) =$	(usamos 2 como denominador común)
$(n+1)(n + 2) / 2 =$	(pues n+2 = n+1+1)
$(n+1)(n + 1 + 1) / 2 =$	(por definición de s(n))
$s(n)(s(n)+1)/2$	

Si seguimos toda la cadena de igualdades, concluimos que $\sum_{i=0}^{s(n)} i = s(n)(s(n)+1)/2$.

Ejercicio 6

Dadas las siguientes funciones definidas sobre secuencias de elementos:

Largo : Sec(A) $\to$ N
Largo (nil) = 0
Largo (cons (x,s)) = 1 + Largo(s)

Clonar : Sec(A) $\to$ Sec(A)
Clonar (nil) = nil
Clonar (cons (x,s)) = cons (x, Clonar(s))

Duplicar : Sec(A) $\to$ Sec(A)
Duplicar (nil) = nil
Duplicar (cons (x,s)) = cons (x, cons (x, Duplicar(s)))

a) Aplique paso a paso la función Largo para calcular Largo ([a, b, c]).

Solución:

Largo ([a,b,c]) = Largo (cons (a,[b,c])) = 1 + Largo ([b,c]) (por paso recursivo Largo)
Largo ([b,c]) = Largo (cons (b,[c])) = 1 + Largo ([c]) (por paso recursivo Largo)
Largo ([c]) = Largo (cons (c,nil)) = 1 + Largo (nil) por ii)= 1 + 0 = 1 (por paso base Largo)
Largo ([b,c]) = 1 + Largo ([c]) = 1 + 1 = 2
Largo ([a,b,c]) = 1 + Largo ([b,c]) = 1 + 2 = 3

b) Aplique paso a paso la función Clonar para calcular Clonar ([a, b, c]).

Solución:

Clonar ([a,b,c]) = cons (a, Clonar([b,c])) (por paso recursivo Clonar)
Clonar ([b,c]) = cons (b, Clonar([c])) (por paso recursivo Clonar)
Clonar ([c]) = cons (c, Clonar(nil)) (por paso recursivo Clonar)
Clonar(nil) = nil (por paso base Clonar)
Uniendo todos los resultados anteriores tenemos que:
Clonar ([a,b,c]) = cons (a, cons (b, cons (c, nil))) = [a, b, c]

c) Aplique paso a paso la función Duplicar para calcular Duplicar ([a, b, c]).

Solución:

Duplicar ([a,b,c]) = cons (a, cons (a, Duplicar ([b,c]))) (por paso recursivo Duplicar)
Duplicar ([b,c]) = cons (b, cons (b, Duplicar ([c]))) (por paso recursivo Duplicar)
Duplicar ([c]) = cons (c, cons (c, Duplicar (nil))) (por paso recursivo Duplicar)
Duplicar (nil) = nil (por paso base Duplicar)
Uniendo todos los resultados anteriores tenemos que:
Duplicar ([a,b,c]) = cons (a, cons (a, cons (b, cons (b, cons (c, cons (c, nil)))))) =
= [a, a, b, b, c, c]

d) Demuestre por inducción estructural que para toda s $\in$ Sec(A) se cumple que:
Largo (Clonar (s)) = Largo (s).

Solución:

Paso base: Debemos probar que Largo (Clonar (nil)) = Largo (nil).

<u>Demostración</u>:

Dado que Clonar (nil) = nil, se cumple que Largo (Clonar (nil)) = Largo (nil) = 0.

Paso inductivo: HI) Largo (Clonar (s)) = Largo (s)
TI) Largo (Clonar (cons (x,s))) = Largo (cons (x,s))

<u>Demostración</u>:

Largo (Clonar (cons (x,s))) = (por paso recursivo de Clonar)
Largo (cons (x, Clonar (s))) = (por paso recursivo de Largo)
1 + Largo (Clonar (s)) = (por hip. inductiva)
1 + Largo (s) = (por paso recursivo de Largo)
Largo (cons (x,s))

Si seguimos toda la cadena de igualdades, concluimos que:
Largo (Clonar (cons (x,s))) = Largo (cons (x,s))

e) Demuestre por inducción estructural que para toda s $\in$ Sec(A) se cumple que:

Largo (Duplicar(s)) = 2 * Largo (s).

<u>Solución</u>:

Paso base: Debemos probar que Largo (Duplicar (nil)) = 2 x Largo (nil).

<u>Demostración</u>:

Dado que Duplicar (nil) = nil, se cumple que Largo (Duplicar (nil)) = Largo (nil) = 0.

También se cumple que 2 x Largo (nil) = 0, lo que prueba la igualdad.

Paso inductivo: HI) Largo (Duplicar(s)) = 2 * Largo (s)

TI) Largo (Duplicar (cons (x,s))) = 2 * Largo (cons (x,s))

<u>Demostración</u>:

Largo (Duplicar (cons (x,s))) =	(por paso recursivo Duplicar)
Largo (cons (x, cons (x, Duplicar(s)))) =	(por paso recursivo Largo, aplicado dos veces)
1 + 1 + Largo (Duplicar (s)) =	(por hip. inductiva)
1 + 1 + 2 * Largo (s) =	(pues 1+1 = 2)
2 + 2 * Largo (s) =	(sacamos 2 de factor común)
2 * (1 + Largo (s)) =	(por paso recursivo Largo)
2 * Largo (cons (x,s))	

Si seguimos toda la cadena de igualdades, concluimos que:

Largo (Duplicar (cons (x,s))) = 2 * Largo (cons (x,s)).

Ejercicio 7

Dadas las siguientes funciones definidas sobre secuencias de elementos:

Largo : Sec(N) $\rightarrow$ N
Largo (nil) = 0
Largo (cons (x,s)) = 1 + Largo (s)

Sumar : Sec(N) $\rightarrow$ N
Sumar (nil) = 0
Sumar (cons (x,s)) = x + Sumar (s)

Sustituir : Sec (N) x N $\rightarrow$ Sec (N)
Sustituir (nil, n) = nil
Sustituir (cons (x,s), n) = cons (n, Sustituir (s, n))

a) Calcule el resultado de Largo (Sustituir ([1, 2], 5)), aplicando paso a paso las definiciones de las funciones Largo y Sustituir.

<u>Solución</u>:

Largo (Sustituir ([1, 2], 5))	=	(por paso recursivo de Sustituir)
Largo (cons (5, Sustituir ([2],5)))	=	(por paso recursivo de Sustituir)
Largo (cons (5, cons (5, Sustituir ([ ],5))))	=	(por paso base de Sustituir)
Largo (cons (5, cons (5, [ ])))	=	(por paso recursivo de Largo)
1 + Largo (cons (5, [ ]))	=	(por paso recursivo de Largo)
1 + 1 + Largo ([ ])	=	(por paso base de Largo)
1 + 1 + 0	=	2

b) Calcule el resultado de Sumar (Sustituir ([3, 2, 7], 1)), aplicando paso a paso las definiciones de las funciones Sumar y Sustituir.

<u>Solución</u>: (próxima página)

Sumar (Sustituir ([3, 2, 7], 1))	=	(por paso recursivo de Sustituir)
Sumar (cons (1, Sustituir ([2, 7], 1)))	=	(por paso recursivo de Sustituir)
Sumar (cons (1, cons (1, Sustituir ([7], 1))))	=	(por paso recursivo de Sustituir)
Sumar (cons (1, cons (1, cons (1, Sustituir ([ ], 1)))))	=	(por paso base de Sustituir)
Sumar (cons (1, cons (1, cons (1, [ ]))))	=	(por paso recursivo de Sumar)
1 + Sumar (cons (1, cons (1, [ ])))	=	(por paso recursivo de Sumar)
1 + 1 + Sumar (1, cons (1, [ ]))	=	(por paso recursivo de Sumar)
1 + 1 + 1 + Sumar ([ ])	=	(por paso base de Sumar)
1 + 1 + 1 + 0	=	3

c) Demuestre por inducción estructural que para toda s $\in$ Sec(A) y para todo n $\in$ A se cumple que:

Largo (s) = Largo (Sustituir (s, n)).

Solución:

Paso base: Debemos probar que Largo (nil) = Largo (Sustituir (nil, n)).

Demostración: Por paso base de Sustituir, tenemos que Sustituir (nil, n) = nil, por lo tanto Largo (Sustituir (nil, n)) = Largo (nil) = 0 (por paso base de Largo)

Paso inductivo:

HI) Largo (s) = Largo (Sustituir (s, n))

TI) Largo (cons (x,s)) = Largo (Sustituir (cons (x,s), n))

Demostración:

Largo (cons (x,s))	=	(por paso recursivo de Largo)
1 + Largo (s)	=	(por hipótesis inductiva)
1 + Largo (Sustituir (s, n))	=	(por paso recursivo de Largo)
Largo (cons (n, Sustituir (s, n)))	=	(por paso recursivo de Sustituir)
Largo (Sustituir (cons (x,s), n)).		

Si seguimos la cadena de igualdades, concluimos que:
Largo (cons (x,s)) = Largo (Sustituir (cons (x,s), n)).

d) Demuestre por inducción estructural que para toda s $\in$ Sec(A) se cumple que:

Largo (s) = Sumar (Sustituir (s, 1)).

Solución:

Paso base: Debemos probar que Largo (nil) = Sumar (Sustituir (nil, 1)).

Demostración: Por paso base de Sustituir, tenemos que Sustituir (nil, 1) = nil, por lo tanto Sumar (Sustituir (nil, 1)) = Sumar (nil) = 0 (por paso base de Sumar)

Paso inductivo:

HI) Largo (s) = Sumar (Sustituir (s, 1))

TI) Largo (cons (x,s)) = Sumar (Sustituir (cons (x,s), 1))

Demostración:

Largo (cons (x,s))	=	(por paso recursivo de Largo)
1 + Largo (s)	=	(por hipótesis inductiva)
1 + Sumar (Sustituir (s, 1))	=	(por paso recursivo de Sumar)
Sumar (cons (1, Sustituir (s, 1)))	=	(por paso recursivo de Sustituir)
Sumar (Sustituir (cons (x,s), 1))		

Si seguimos la cadena de igualdades, concluimos que:
Largo (cons (x,s)) = Sumar (Sustituir (cons (x,s), 1))