10 (Diez)

COM-4
TOMÁS ACHAUAR
45083116

Examen Parcial de Introducción a los Algoritmos - 18 de Abril de 2022
Comisiones Turno Tarde

(a) Definir la función recursiva `sina :: [Char] -> [Char]`, que dada una lista de caracteres, devuelve la lista que elimina todas las letras `'a'` dejando el resto en el mismo orden. Ejemplo:

- `sina  "casa" = "cs"`

(b) Evaluar manualmente la función para el ejemplo dado. Justificar cada paso, siguiendo la notación usada en clase.

(a) Definir la función recursiva `checkSum :: [(Int, Int, Int)] -> [Bool]` dada una lista de triplas de números, verifica si es verdad o no que el tercero es la suma de los dos primeros. Ejemplo:

- `checkSum [(1,3,4),(4,8,10)] =[True,False]`

(b) Evaluar manualmente la función para el ejemplo dado. Justificar cada paso, siguiendo la notación usada en clase.

Dadas las siguientes funciones

```
prod1  :: [Int] -> Int
prod1 [ ]  = 1                    --- (1)
prod1 (x:xs) =  x*(prod1 xs)      --- (2)

prod2 :: [Int] -> Int
prod2 [ ] = 1                     --- (3)
prod2 (x:xs)
```

18/04/2022 PARCIAL 1 - TURNO TARDE - INTRO A LOS ALGORITMOS

TOMÁS ACHÁVAL - COM 4
45085746

1. a) sina :: [Char] -> [Char]

① sina [] = []

sina (x:xs) | x == a = sina xs ②a

| otherwise = x:(sina xs) ②b

b) sina "casa"

= {②b x:='c' xs:="asa"}

'c':(sina "asa")

= {②a x:='a' xs:="sa"}

'c':(sina "sa")

= {②b x:='s' xs:="a"}

'c':'s':(sina "a")

= {②a x:='a' xs:=[]}

'c':'s':(sina [])

= {①}

'c':'s':[]

= {POR DEF. DE :}

['c','s'] LO CUAL TAMBIÉN PUEDE SER ESCRITO COMO "cs"

2. a) checkSum :: [(Int, Int, Int)] -> [Bool]

① checkSum [] = []

② checkSum ((a,b,c):xs) = ((a+b)==c):(checkSum xs)

b) EVALUAR checkSum [(1,3,4),(4,8,10)]

= {② (a,b,c):=(1,3,4) xs:=[(4,8,10)]}

((1+3)==4):(checkSum [(4,8,10)])

= {② (a,b,c):=(4,8,10) xs:=[]}

((1+3)==4):((4+8)==10):[checkSum []]

= {①} (Y ARITMÉTICA)

(4==4):(12==10):[]

= {POR EVALUACIÓN DE IGUALDADES}

TRUE:FALSE:[]

= {POR DEF. DE :}

[TRUE, FALSE]

```
Prod1 :: [Int] -> Int
① Prod1 [] = 1
② Prod1 (x:xs) = x * (Prod1 xs)

Prod2 :: [Int] -> Int
③ Prod2 [] = 1
(4a) Prod2 (x:xs) | x == 0 = 0
(4b)              | otherwise = x * (Prod2 xs)
```

a) Prod1 xs = Prod2 xs → HIPÓTESIS INDUCTIVA.

VEAMOS EL CASO BASE:

Prod1 [] = Prod2 []

≡{①}

1 = Prod2 []

≡{③}

1 = 1

TRUE

b) AHORA DEMOSTREMOS EL CASO INDUCTIVO:

Prod1 (x:xs) = Prod2 (x:xs)

≡{② x:=x xs:=xs}

x * (Prod1 xs) = Prod2 (x:xs)

CASO 1 (x==0):

≡{(4a) x:=x xs:=xs}

x * (Prod1 xs) = 0

≡{como x==0 y Prod1 xs es un Int, sabemos que cualquier Int * 0 = 0}

0 * (Prod1 xs) = 0

≡{Int * 0 = 0}

0 = 0

≡{REFLEX de =}

TRUE

CASO 2 (otherwise): (x/=0)

≡{(4b) x:=x xs:=xs}

x * (Prod1 xs) = x * (Prod2 xs)

≡{HI}

x * (Prod2 xs) = x * (Prod2 xs)

≡{REFLEXIVIDAD DE =}

TRUE

OBS: CABE ACLARAR QUE EN EL CASO 1 NO FUE NECESARIO UTILIZAR LA HIPÓTESIS INDUCTIVA YA QUE LA FÓRMULA DEL CASO 4a NO ESTÁ DEFINIDA DE MANERA RECURSIVA, POR LO QUE NOS QUEDA UNA MULTIPLICACIÓN POR CERO IGUALADA A CERO, LO CUAL ES TRUE TANTO PARA 0*(Prod1 xs) COMO PARA 0*(Prod2 xs), IMPLICANDO QUE NO ES NECESARIO EL REEMPLAZO DE (Prod1 xs) POR (Prod2 xs).