FUNDAMENTOS DE COMPUTACIÓN
TRABAJO ENTREGABLE 2
ABRIL 2022

Este trabajo tiene un puntaje de 5 puntos, y debe ser realizado en forma **INDIVIDUAL**. Se debe subir a Aulas antes del 10/4/22 a las 21hs.

El conectivo implica tiene la siguiente tabla de verdad:

b1	b2	b1 >> b2
False	False	True
False	True	True
True	False	False
True	True	True

y se define en Haskell como la siguiente función:

```
(>>) :: Bool -> Bool -> Bool
(>>) = \b1 b2 -> case b1 of {False -> True ; True -> b2}
```

SE PIDE:

1) Demuestre que `(∀x::Bool)(∀y::Bool)(∀z::Bool) x >> (y >> z) = (x && y) >> z`, donde `(&&) :: Bool -> Bool -> Bool` se define como:
`(&&) = \b1 -> \b2 -> case b1 of {False -> False ; True -> b2}`.

   Solución:

   `(∀x::Bool)(∀y::Bool)(∀z::Bool) x >> (y >> z) = (x && y) >> z`.

   Dem. Por casos en `x::Bool`

   Caso `x = False`: `(∀y::Bool)(∀z::Bool) False >> (y >> z) = (False && y) >> z`
   Sean `y,z::Bool cualesquiera`, reducimos ambos lados de la igualdad:

`False >> (y >> z)`	`(False && y) >> z`
= (def. `(>>)`, $\beta\times 2$, `case`)	= (def. `(&&)`, $\beta\times 2$, `case`)
`True`	`False >> z`
	= (def. `(>>)`, $\beta\times 2$, `case`)
	`True`

   Ambas expresiones son iguales por reducir a la misma expresión.

Caso `x = True:` $(\forall$`y::Bool`$)(\forall$`z::Bool`$)$ `True >> (y >> z) = (True && y) >> z`

Sean `y,z::Bool cualesquiera`, reducimos ambos lados de la igualdad:

`True >> (y >> z)`	`(True && y) >> z`
$=$ (def. `(>>)`, $\beta\times 2$, `case`)	$=$ (def. `(&&)`, $\beta\times 2$, `case`)
`y >> z`	`y >> z`

Ambas expresiones son iguales por reducir a la misma expresión.

2) Defina en Haskell, sin usar funciones auxiliares, la negación del implica como la función `(*) :: Bool -> Bool -> Bool`, que tiene la siguiente tabla de verdad:

b1	b2	b1 * b2
True	True	False
True	False	True
False	True	False
False	False	False

Solución:

```
(*) :: Bool -> Bool -> Bool
(*) =\b1 b2 -> case b1 of {False -> False ;
                           True -> case b2 of {False -> True ;
                                               True -> False }}
```

3) Demuestre que `(`$\forall$`x::Bool)(`$\forall$`y::Bool) x * y = not (x >> y)`, donde:
`not :: Bool -> Bool` se define como:
`not = \b -> case b of {False -> True ; True -> False}`.

Solución:

`(`$\forall$`x::Bool)(`$\forall$`y::Bool) x * y = not (x >> y)`.

Dem. Por casos en `x::Bool`

Caso `x = False:` $(\forall$`y::Bool`$)$ `False * y = not (False >> y)`

Sea `y::Bool` cualquiera, reducimos ambos lados de la igualdad:

`False * y`	`not(False >> y)`
$=$ (def. `(*)`, $\beta\times 2$, `case`)	$=$ (def. `(>>)`, $\beta\times 2$, `case`)
`False`	`not True`
	$=$ (def. `not`, β, `case`)
	`False`

Ambas expresiones son iguales por reducir a la misma expresión.

Caso `x = True:` $(\forall$`y::Bool) True * y = not (True >> y)`

Por casos en `y::Bool`

Caso `y = False:` `True * False = not (True >> False)`

Reducimos ambos lados de la igualdad:

`True * False`	`not(True >> False)`
= (def. `(*)`, $\beta\times 2$, `case`$\times 2$)	= (def. `(>>)`, $\beta\times 2$, `case`)
`True`	`not False`
	= (def. `not`, β, `case`)
	`True`

Ambas expresiones son iguales por reducir a la misma expresión.

Caso `y = True:` `True * True = not (True >> True)`

Reducimos ambos lados de la igualdad:

`True * True`	`not(True >> True)`
= (def. `(*)`, $\beta\times 2$, `case`$\times 2$)	= (def. `(>>)`, $\beta\times 2$, `case`)
`False`	`not True`
	= (def. `not`, β, `case`)
	`False`

Ambas expresiones son iguales por reducir a la misma expresión.

ENTREGA:

Se deberá subir un único archivo a Aulas, que puede ser escrito en máquina o en papel y escaneado.
En caso de que sea lo segundo, pedimos que el documento sea *legible*. Si utilizan fotos, se recomienda utilizar alguna aplicación para escanearlas y generar archivos pdf.