Andrea Romero
Isabel Paiz

RESOLUCIÓN

PRIMER EXAMEN PARCIAL

Pregunta 1

Demuestre que para todo *natural de peano* "**n"** se cumple la siguiente propiedad:

$$Succ\,0 + n = Succ\,n$$

Utilize la definicion de *suma* estudiada en clase como definicion del signo "+".

Reglas/Producciones de la Suma

Regla 1: n + 0 = m

Regla 2: 0 + m = m

Regla 3: n + Succ(m) = Succ(n + m)

Propiedad: *Succ 0 + n = Succ n*

Succ (0 + n) = Succ n

Succ (n) = Succ n

Succ n = Succ n

Pregunta 2

Provea una definicion inductiva para la propiedad "mayor que" (>) tal que:

$$a > b \begin{cases} Succ\,0 & \textit{si a es mayor que b} \\ 0 & \textit{de lo contrario} \end{cases}$$

En otras palabras, la propiedad "mayor que" es equivalente a **Succ 0** si el primer valor es mayor que el segundo o **0** de lo contrario. Puede utilizar el operador ">" en su definicion de la misma manera que se utiliza "+" en la definicion de suma.

El Axioma A5 hace referencia al Axioma de Inducción.

a ⊕ Succ b = Succ (a ⊕ b)

a > b = Succ (a > b)

a > b = Succ (0 > 0)

a > b = Succ 0 > b

Succ 0 > 0 = a > b

Andrea Romero
Isabel Paiz

Pregunta 3

Provea una definicion de las propiedades "esPar" e "esImpar" tal que:

$$esPar\ n \begin{cases} Succ\ 0 & si\ n\ es\ un\ numero\ par \\ 0 & de\ lo\ contrario \end{cases}$$

$$esImpar \begin{cases} Succ\ 0 & Si\ n\ es\ impar \\ 0 & de\ lo\ contrario \end{cases}$$

Se sugiere que para implementar estas propiedades, utilize la propiedad inversa en la definicion. En otras palabras un numero "n" es par cuando cierto otro numero es impar y vice versa.

esPar n = 0

esPar n = Succ 0 + 1

esPar n * 0 = 0

esPar n = 0 + Succ 1

esimpar n = Succ 0 + 0

esimpar Succ 0 + 0 = n

esimpar 0 + n = Succ 0

esimpar Succ 0 * n = 0

Pregunta 4

Utilize el lenguaje de programacion Haskell para definir la propiedad "predecesor". Esta propiedad debe aceptar un *numero de peano* y producir el predecesor del mismo. En el caso de *cero,* utilizar cero como su predecesor.

```
{-# LANGUAGE NoImplicitPrelude #-}

module Main where

import Prelude (IO, Show, undefined)

data Natural = Cero | Succ Natural deriving Show


--SERIE 4
Pred Cero = Cero
Pred (Succ Cero) = Cero
Pred n = Succ (Pred (n-1))
```