Tarea 2: Naturales

Puntaje máximo: 8 puntos

Entrega por Aulas: 5 de mayo de 2024 hasta las 21:00 hrs

IMPORTANTE: Deberá subirse a Aulas un único archivo de `Haskell` (`.hs`) con los ejercicios resueltos, aquellos que no son de programar funciones se entregará como código comentado (`{-- --}`). El archivo debe incluir el nombre y número de estudiante al principio del mismo.

Antes de comenzar la tarea definiremos al tipo de los naturales en `Haskell` de la siguiente manera: `type N = Integer`, y si bien lo que estamos haciendo en este caso es dandole un renombre a los enteros, a partir de ahora, podremos asumir que siempre que mis funciones reciban algo de tipo `N`, este será mayor o igual que cero.

? 1. Sea la función `suma_entre :: N -> N -> N`, que dados dos naturales `m` y `n`, calcula la sumatoria desde `m` hasta `n`, de la siguiente manera:

```
suma_entre ::  N -> N -> N
suma_entre m n
     | m > n = 0
     | otherwise = m + (suma_entre (m+1) n)
```

1. Analizándola, podemos concluir que funciona correctamente, a pesar de que en la llamada recursiva ninguno de los argumentos decrece. ¿Cuál es el resultado de aplicar `suma_entre 2 5` ?
2. ¿Por qué la definición precedente es bien fundada (da lugar siempre a computaciones finitas)? ¿Cuál es el "tamaño" del problema que decrece?
3. Definir la función `suma_entre' :: N -> N -> N`, que dados dos naturales `m` y `n`, calcule la sumatoria desde `n` hasta `m` y <u>no</u> desde `m` hasta `n` como la precedente.
4. Definir la función `suma_entre_f :: (N -> N) -> N -> N -> N`, que dada una función `f` y dos naturales `m` y `n`, calcula la sumatoria desde `n` hasta `m` aplicando la función `f` a cada término ($\sum_{i=m}^{n} f\, i$).
5. Haciendo uso de la función `suma_entre_f` definir la función `suma_i :: N -> N`, que dado un natural `n`, calcula la sumatoria desde `i=0` hasta `n` ($\sum_{i=0}^{n} i$).

? 2.

1. Definir la función `es_divisor :: N -> N -> Bool`, que dados dos naturales positivos `n` y `k`, retorna `True` si `k` es un divisor de `n` (o si `n` es un múltiplo de `k`).
   Ejemplos:
   `es_divisor 4 8 = False`, porque 8 no es divisor de 4.
   `es_divisor 10 5 = True`, porque 5 si divisor de 10.
2. Haciendo uso de la función `es_divisor`, definir la función `primer_divisor :: N -> N`, que dado un natural `n` mayor o igual a 2, retorna el primer divisor de `n` entre `[2..n]`.
   Ejemplos:
   `primer_divisor 6 = 2`
   `primer_divisor 7 = 7`
3. Usando las funciones `es_divisor` y `primer_divisor`, definir la función `es_primo :: N -> Bool`, que dado un natural `n`, indica si `n` es un número primo (que tiene exactamente 2 divisores, el 1 y él mismo).
   Ejemplo:
   `es_divisor 11 = True`, porque tiene exactamente 2 divisores el 1 y el 11.

? 3. Sea la función `minimo_acotado :: (N -> Bool) -> N -> N -> N`, que dado un predicado `p` y dos naturales `m` y `n`, se realiza una búsqueda lineal del primer elemento que cumpla el predicado `p` en el intervalo `[m..n]` retornándolo (si tal valor existe). Y se define de la siguiente manera:

```
minimo_acotado :: (N -> Bool) -> N -> N -> N
minimo_acotado p m n
    | m > n = m
    | m <= n && p m = m
    | m <= n && not (p m) = minimo_acotado p (m+1) n
```

1. Explicar la función `mínimo_acotado`. ¿Qué retorna en caso de que ningún valor en el intervalo considerado cumpla el predicado dado?
2. Utilizando las funciones `minimo_acotado` y `es_divisor`, definir la función `primer_divisor' :: N -> N`, que se comporta como la funcón `primer_divisor` definida en el ejercicio `2.2`.
3. Definir la función `maximo_acotado :: (N -> Bool) -> N -> N -> N`, que dado un predicado `p` y dos naturales `m` y `n`, retorna el máximo elemento que cumpla el predicado `p` en el intervalo `[m..n]`. Notar que es una función similar a `minimo_acotado`.
   Ejemplos:
   `maximo_acotado par 3 7 = 6`
   `maximo_acotado es_primo 2 11 = 11`
4. Sea la función `minimo_p :: (N -> Bool) -> N -> N`, la cual no toma recaudos para el caso en que no exista un valor que cumpla el predicado a considerar y, en consecuencia, puede entrar en loop infinito, definida de la siguiente manera:

   ```
   minimo_p :: (N -> Bool) -> N -> N
   minimo_p p n
       | p n = n
       | not (p n) = minimo_p p (n+1)
   ```

   ¿En qué condiciones termina una ejecución de la función precedente?

? 4.

1. Definir la función `cantidad_p :: (N -> Bool) -> N -> N -> N`, que dado un predicado `p`, y dos naturales `m` y `n`, retorna la cantidad de elementos que cumplen `p` en el intervalo `[m..n]`.
2. Definir la función `suma_p :: (N -> Bool) -> N -> N -> N`, que dado un predicado `p` y dos naturales `m` y `n`, retorna la suma de los elementos que cumplen `p` en el intervalo `[m..n]`.
3. Definir la fución `suma2_p :: (N -> Bool) -> N -> N -> N`, que dado un predicado `p` y dos naturales `m` y `n`, retorna la suma de los cuadrados de los elementos que cumplan `p` en el intervalo `[m..n]`.
4. Definir la función `sumaf_p :: (N -> Bool) -> (N -> N) -> N -> N -> N`, que dado un predicado `p`, una función `f` y dos naturales `m` y `n`, calcula la suma de aplicarle la función `f` a los elementos que cumplan `p` en el intervalo `[m..n]`.
5. Definir la funcón `todos_p :: (N -> Bool) -> N -> N -> Bool`, que dado un predicado `p` y dos naturales `m` y `n`, retorna `True` si todos los elementos en el intervalo `[m..n]` satisfacen `p`.
6. Definir la funcón `existe_p :: (N -> Bool) -> N -> N -> Bool`, que dado un predicado `p` y dos naturales `m` y `n`, retorna `True` si algún elemento en el intervalo `[m..n]` satisface `p`.