Práctico 1: Expresiones Cuantificadas

Algoritmos y Estructuras de Datos I
2$^{\text{do}}$ cuatrimestre 2024

Objetivos

El objetivo de esta guía es trabajar sobre la semántica de cuantificadores generalizados y sus propiedades. Abordaremos ejercicios de cálculo de evaluación de expresiones cuantificadas y de aplicación de sus propiedades para lograr familiaridad en la manipulación simbólica de este tipo de expresiones. Conocer los axiomas y teoremas del cálculo y habilidad en las demostraciones es necesario para poder abordar la tarea de derivación y demostración de programas que encararemos de aquí en adelante.

Para el ***laboratorio*** *se evaluará la definición de funciones recursivas usando caso base y caso inductivo (a través de análisis por casos, o pattern-matching), la definición de funciones por composición, y el uso de las funciones provistas por el lenguaje (en el Preludio), para la definición de funciones polimórficas. En algunas de las funciones será necesario utilizar definiciones locales y también el uso de guardas para alternativas booleanas. En otros casos se deberá utilizar aplicación parcial de funciones y operadores binarios.*

Algunas consideraciones que debes tener en cuenta:

- *Definí el tipo de cada función que implementás.*
- *Hacé todo el práctico en un mismo archivo.*
- *Usá* `ghci`.
- *Comentá el código, indicando a qué ejercicio corresponden las funciones (por ejemplo:* `-- ejercicio n`*)*
- *Nombrá las distintas versiones de una misma función utilizando* `'` *(por ejemplo:* `f`, `f''`, `f'''`, *...)*
- ***IMPORTANTE****: En cada ejercicio pone como comentario, al menos dos ejemplos de haber ejecutado la función en* `ghci` *y su resultado.*

Cálculo proposicional

En el siguiente ejercicio se propone una revisión de algunos de los teoremas de la lógica proposicional que nos serán útiles para el cálculo utilizado en toda la materia. Se utiliza la misma notación utilizada en Introducción a los Algoritmos para las demostraciones, que además, se segurá utilizando para otras expresiones más complejas.

0. Demostrá algunos de los siguientes teoremas utilizando los axiomas y teoremas del cálculo proposicional listados en el digesto (y cualquier otro teorema que se te ocurra y demuestres, claro). Es importante que utilices la notación utilizada en clases.

   a) *Idempotencia de la conjunción*: $p \wedge p \equiv p$.

   b) *Neutro de la conjunción*: $p \wedge True \equiv p$.

   c) *Absorción de la conjunción*: $p \wedge (p \vee q) \equiv p$

   d) *Absorción de la disyunción*: $p \vee (p \wedge q) \equiv p$

   e) *Debilitamiento para* $\wedge$: $p \wedge q \Rightarrow p$.

   f) *Debilitamiento para* $\vee$: $p \Rightarrow p \vee q$.

   g) *Caracterización de* $\Rightarrow$: $p \Rightarrow q \equiv \neg p \vee q$.

   h) *Contrarrecíproca*: $p \Rightarrow q \equiv \neg q \Rightarrow \neg p$.

   i) *De Morgan para* $\wedge$: $\neg(p \wedge q) \equiv \neg p \vee \neg q$

   j) *De Morgan para* $\vee$: $\neg(p \vee q) \equiv \neg p \wedge \neg q$

   k) *Distributividad de* $\vee$ *con* $\wedge$: $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$

   l) *Distributividad de* $\wedge$ *con* $\vee$: $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$

   m) *Currificación* : $p \Rightarrow (q \Rightarrow r) \equiv p \wedge q \Rightarrow r$.

   n) *Distributividad de* $\Rightarrow$ *con* $\vee$: $p \vee q \Rightarrow r \equiv (p \Rightarrow r) \wedge (q \Rightarrow r)$.

ñ) Distributividad de $\Rightarrow$ con $\wedge$: $p \Rightarrow (q \wedge r) \equiv (p \Rightarrow q) \wedge (p \Rightarrow r)$.

Por ejemplo:

$$
\begin{array}{ll}
 & \underline{p \Rightarrow q} \equiv \neg p \vee q \\
\equiv & \{ \text{ Definición de } \Rightarrow \} \\
 & p \vee q \equiv q \equiv \underline{\neg p} \vee q \\
\equiv & \{ \text{ Equivalencia y negación } \} \\
 & p \vee q \equiv q \equiv \underline{(p \equiv False)} \vee q \\
\equiv & \{ \text{ Distributividad de } \vee \text{ con } \equiv \} \\
 & p \vee q \equiv q \equiv p \vee q \equiv \underline{False \vee q} \\
\equiv & \{ \text{ Neutro de } \vee \ \} \\
 & p \vee q \equiv q \equiv p \vee q \equiv q \\
\equiv & \{ \text{ Reflexividad de } \equiv \ \} \\
 & True
\end{array}
$$

Semántica de los cuantificadores generalizados

*A los cuntificadores **universial** y **existencial** ya conocidos, se incorporan otros cuantificadores que serán de utilidad para expresar problemas más complejos.*

- *Cuantificador universal* ($\forall$)
- *Cuantificador existencial* ($\exists$)
- *Sumatoria* ($\sum$)
- *Productoria* ($\prod$)
- *Máximo (Max)*
- *Mínimo (Min)*
- *Intersección* ($\bigcap$)
- *Unión* ($\bigcup$)

El objetivo de los siguientes ejercicios es ejercitar la lectura y el uso de dichos cuantificadores generalizados para expresar problemas complejos de manera más simple. Los ejercicios de laboratorio requerirán definir funciones que impementen lo descripto en lenguaje formal.

A los fines de contar con estructuras de datos que nos permitan expresar propiedades sobre listas, utilizaremos el tipo de datos Figura. Una Figura es una tupla que contiene información sobre una figura geométrica, en particular: la forma, el color y el tamaño. Utilizaremos los predicados rojo, amarillo, azul, verde para determinar si la figura tiene un color particular; los predicados triangulo, cuadrado, rombo, circulo para determinar su forma; y utilizaremos la función tam para devolver el tamaño de la figura, que será un valor numérico.

Laboratorio 1 Ejercicio básico de repaso para comenzar a trabajar en el laboratorio. Programá las siguientes funciones:

1. `esCero :: Int -> Bool`, que verifica si un entero es igual a 0.
2. `esPositivo :: Int -> Bool`, que verifica si un entero es estrictamente mayor a 0.
3. `esVocal :: Char -> Bool`, que verifica si un caracter es una vocal en minúscula.
4. `valorAbsoluto :: Int -> Int`, que devuelve el valor absoluto de un entero ingresado.

1. Teniendo en cuenta que xs es una lista de *Figuras* como describimos anteriormente, describí el significado de las siguientes expresiones utilizando el lenguaje natural.

   a) $\langle \forall\, i \ : 0 \leq i < \#xs : \ rojo.(xs.i) \rangle$
   b) $\langle \exists\, i \ : 0 \leq i < \#xs : \ rombo.(xs.i) \rangle$
   c) $\langle \exists\, i \ : 0 \leq i < \#xs : \ \langle \exists\, j \ : 0 \leq j < \#xs : \ xs.i = xs.j \rangle \rangle$
   d) $\langle \exists\, i \ : 0 \leq i < \#xs : \ \langle \exists\, j \ : i < j < \#xs : \ xs.i = xs.j \rangle \rangle$
   e) $\langle \sum i \ : 0 \leq i < \#xs : \ tam.(xs.i) \rangle$

   Observación: La desigualdad de la forma $A \leq B < C$ es una abreviatura para $(A \leq B) \wedge (B < C)$ que utilizaremos de aquí en adelante.

   Pregunta: ¿Cuál es la diferencia entre el punto $1c$ e $1d$?

2. Escribí fórmulas para las siguientes expresiones en lenguaje natural.

a) Todas las figuras de xs son amarillas.

b) La suma de los tamaños de todas las figuras de xs es mayor a 10.

c) Ninguna figura de xs tiene tamaño menor a 7.

Observación: El último ejercicio se puede expresar con el cuantificador Min . Si no lo resolviste así, resolverlo también de esa manera.

3. Para cada una de las siguientes funciones escribí una expresión que las describa formalmente. Por otro lado, escribí un programa recursivo que compute la función.

   a) `paratodo :: [Bool] -> Bool`, que verifica que *todos* los elementos de una lista sean `True`.

   b) `sumatoria :: [Int] -> Int`, que calcula la suma de todos los elementos de una lista de enteros.

   c) `productoria :: [Int] -> Int`, que calcula el producto de todos los elementos de la lista de enteros.

   d) `factorial :: Int -> Int`, que toma un número n y calcula $n!$.

   e) Utilizá la función `sumatoria` para definir, `promedio :: [Int] -> Int`, que toma una lista de números no vacia y calcula el valor promedio (truncado, usando división entera).

Laboratorio 2 Implementá en Haskell las funciones definidas en el ejercicio anterior.
A continuación mostramos algunos ejemplos del uso de las funciones en `ghci`:

```
$> paratodo [True, False, True]
False
$> paratodo [True, True]
True
$> sumatoria [1, 5, -4]
2
$> productoria [2, 4, 1]
8
```

4. Para cada una de las siguientes fórmulas, describí su significado utilizando el lenguaje natural.

   a) $\langle \forall\, i \;:\; 0 \leq i < \#xs :\; xs.i > 0 \rangle$

   b) $\langle \exists\, i \;:\; 0 \leq i < \#xs :\; xs.i = x \rangle$

   c) $\langle \forall\, i \;:\; 0 \leq i < \#xs :\; \langle \exists\, j \;:\; 0 \leq j < \#ys :\; xs.i = ys.j \rangle \rangle$

   d) $\langle \forall\, i \;:\; 0 \leq i < \#xs - 1 :\; xs.i = xs.(i+1) \rangle$

5. Para cada uno de los ítems del ejercicio anterior, evaluá la fórmula en las siguientes listas:

   a) $xs = [-5, -3, 4, 8]$

   b) $xs = [11, 2, 5, 8]$

   Para el ítem *b*), considerar $x = 5$. Para el ítem *c*), considerar $ys = [2, -3, 11, 5, 8]$.

   Ejemplo: Fórmula 4.*a*) aplicada a $xs = [-5, -3, 4, 8]$:

$$
\begin{array}{ll}
 & \langle \forall\, i \;:\; 0 \leq i < \#xs :\; xs.i > 0 \rangle \\
\equiv & \{ \text{ calculo rango sabiendo que } \#xs = 4 \ \} \\
 & \langle \forall\, i \;:\; i \in \{0,1,2,3\} :\; xs.i > 0 \rangle \\
\equiv & \{ \text{ aplico el término a cada elemento del rango } \} \\
 & (xs.0 > 0) \wedge (xs.1 > 0) \wedge (xs.2 > 0) \wedge (xs.3 > 0) \\
\equiv & \{ \text{ evalúo las indexaciones con } xs = [-5,-3,4,8] \ \} \\
 & (-5 > 0) \wedge (-3 > 0) \wedge (4 > 0) \wedge (8 > 0) \\
\equiv & \{ \text{ evalúo las desigualdades } \} \\
 & False \wedge False \wedge True \wedge True \\
\equiv & \{ \text{ resuelvo las conjunciones } \} \\
 & False
\end{array}
$$

Laboratorio 3 A partir de las expresiones de los ejercicios 4*a*, 4*b* y 4*d*

a) Identificá las variables libres de cada expresión y el tipo de cada una.

b) Definí funciones que tomen como argumento las variables libres identificadas y devuelvan el resultado de la expresión. **Atención:** Tené en cuenta que en algunos casos es necesario definir funciones auxiliares.

c) Evaluá las funciones tomando como argumento los valores señalados en 5.

6. Escribí fórmulas para las siguientes expresiones en lenguaje natural.

   a) Todos los elementos de xs e ys son iguales (*¡ojo! ¡sujeta a interpretación!*).
   b) Todos los elementos de xs ocurren en ys.
   c) Todos los elementos de xs ocurren en ys en la misma posición.

7. Para cada una de las siguientes fórmulas, describí su significado utilizando el lenguaje natural.

   a) $\langle \prod\ i\ : 1 \leq i \leq n :\ i \rangle$
   b) $\dfrac{\langle \sum i\ : 0 \leq i < \#xs :\ xs.i \rangle}{\#xs}$
   c) $\langle \text{Max}\ \ i\ : 0 \leq i < \#xs :\ xs.i \rangle < \langle \text{Min}\ \ i\ : 0 \leq i < \#ys :\ ys.i \rangle$
   d) $\langle\, \exists\, i, j\ : (2 \leq i < n) \wedge (2 \leq j < n) :\ i * j = n \,\rangle$

8. Para cada uno de los ítems del ejercicio anterior, evaluá respectivamente con los siguientes valores:

   a) $n = 5$.
   b) $xs = [6, 9, 3, 9, 8]$.
   c) $xs = [-3, 9, 8], ys = [6, 7, 8]$.
   d) $n = 5$.

Laboratorio 4 A partir de las expresiones en el ejercicio 7

a) Identificá las variables libres de cada expresión y el tipo de cada una.

b) Definí funciones que tomen como argumento las variables libres identificadas y devuelvan el resultado de la expresión. **Atención:** Tené en cuenta que en algunos casos es necesario definir varias funciones.

c) Evaluá las funciones tomando como argumento los valores señalados en el ejercicio 8.

9. Suponiendo que $f : A \to Bool$ es una función fija cualquiera, y $xs : [A]$, caracterizá con una cuantificación la siguiente función recursiva:

$$algunof : [A] \to Bool$$

$$\begin{aligned} algunof.[\,] &= False \\ algunof.(x \triangleright xs) &= f.x \vee algunof.xs \end{aligned}$$

10. Definí recursivamente una función $todos : [Bool] \to Bool$ que verifica que todos los elementos de una lista son $True$, es decir, que satisface la siguiente especificación:

$$todos.xs \equiv \langle\, \forall\, i\ : 0 \leq i < \#xs :\ xs.i \,\rangle$$

Laboratorio 5 Implementá en Haskell la función que definiste en el ejercicio anterior.

11. Escribí fórmulas para describir formalmente las siguientes expresiones en lenguaje natural.

    Preguntas: ¿Qué tipo debe tener cada variable libre para que expresión tenga sentido? ¿Qué tipo tiene cada expresión?

    a) n es potencia de 2.

    b) n es el elemento más grande de xs.

    c) El producto de los elementos pares de xs.

    d) La suma de los elementos en posición par de xs.

> **Laboratorio 6** A partir de las expresiones del ejercicio anterior b) y c y d)
>
> a) Identificá las variables libres de cada expresión y el tipo de cada una.
>
> b) Definí funciones que tomen como argumento las variables libres identificadas y devuelvan el resultado de la expresión. **Atención:** Tené en cuenta que en algunos casos es necesario definir varias funciones.

12. **Calculá los rangos** de las siguientes cuantificaciones como conjuntos de posibles valores. Tomar $n = 10$, $xs = [-3, 9, 8, 9]$, $ys = [6, 9, 3]$. Usá tuplas cuando haya más de una variable cuantificada.

    a) $\langle \prod\, i \ : 1 \leq i \leq n \ \wedge \ i \bmod 3 = 1 : \ i \rangle$

    b) $\langle \sum i, j \ : 0 \leq i < \#xs \ \wedge \ 0 \leq j < \#ys : \ xs.i * ys.j \rangle$

    c) $\langle \forall\, i, j \ : 0 \leq i < j < \#xs : \ xs.i \neq xs.j \rangle$

    d) $\langle \text{Max} \ as, bs \ : xs = as +\!\!+ bs : \ sum.as \rangle$

    e) $\langle \sum i \ : 1 \leq i + 1 < \#xs + 1 : \ (x \triangleright xs).(i + 1) \rangle$

    Ejemplo: En el segundo ítem tenemos que i y j son independientes entre sí. Por lo tanto tenemos que ver todas las combinaciones posibles con $i \in \{0, 1, 2, 3\}$ y $j \in \{0, 1, 2\}$. En el tercer ítem hay una dependencia de j respecto de i; por lo tanto primero podés calcular los valores posibles de i y luego, para cada posible valor de i, los posibles valores de j.

Cálculo con cuantificadores generalizados

En esta sección comenzaremos a trabajar con las propiedades básicas de los cuantificadores.

13. Simplicá hasta obtener una expresión sin cuantificador las siguientes expresiones aplicando las propiedades de cuantificadores **rango vacío, rango unitario o término constante** y propiedades báscas de la aritmética.

    a) $\langle \exists\, i \ : i = 3 \wedge i \bmod 2 = 0 : \ 2 * i = 6 \rangle$

    b) $\langle \sum i \ : 5 \leq i \wedge i \leq 5 : \ -2 * i \rangle$

    c) $\langle \prod\, i \ : 0 < i < 1 : \ 34 \rangle$

    d) $\langle \text{Min} \ \ i \ : i \leq 0 \vee i > 10 : \ n * (i + 2) - n * i \rangle$

    e) $\langle \text{Max} \ \ a, as \ : a \triangleright as = [\,] : \ \#as \rangle$

14. Aplicá **partición de rango** si es que se puede, y si no se puede, explicá porqué.

    a) $\langle \sum i \ : i = 0 \ \vee \ 4 > i \geq 1 : \ n * (i + 1) \rangle$

    b) $\langle \forall\, i \ : 3 \leq |i| \leq 4 \ \vee \ 0 < i < 4 : \ \neg f.i \rangle$

    c) $\langle \sum i \ : |i| \leq 1 \ \vee \ 0 \leq 2 * i < 7 : \ i * n \rangle$

    d) $\langle \prod\, i \ : 0 \leq i < \#xs \ \wedge \ (i \bmod 3 = 0 \ \vee \ i \bmod 3 = 1) : \ 2 * xs.i + 1 \rangle$ (distribuir primero)

15. Evaluá las expresiones del ejercicio 14 tomando los siguientes valores de las variables libres: $n = 3$, $f.x = |x| < 4$, $xs = [-1, 1, 0, 3]$.

16. Descubrí el error en la siguiente prueba. Es decir, ¿la supuesta propiedad demostrada, vale para todos los valores posibles de xs? ¿Para qué valor o valores de xs falla?

$\langle \sum i \ : 0 \le i < \#xs : \ xs.i \rangle$
= { lógica }
$\langle \sum i \ : i = 0 \ \vee \ 1 \le i < \#xs : \ xs.i \rangle$
= { partición de rango disjunto }
$\langle \sum i \ : i = 0 : \ xs.i \rangle + \langle \sum i \ : 1 \le i < \#xs : \ xs.i \rangle$
= { rango unitario }
$xs.0 + \langle \sum i \ : 1 \le i < \#xs : \ xs.i \rangle$

17. Aplicá **distributividad**, si es que se puede.

    a) $\langle \sum i \ : i = 0 \ \vee \ 4 > i \ge 1 : \ n * (i + 1) \rangle$
    b) $\langle \prod i \ : 3 \le |i| \le 4 \ \vee \ 0 < i < 4 : \ n + i \rangle$
    c) $\langle \forall i \ : i = 0 \ \vee \ 4 > i \ge 1 : \ \neg(f.i \ \wedge \ f.n) \rangle$
    d) $\langle \text{Max} \ i \ : 0 \le i < \#xs : \ x + xs.i \rangle$

18. Calculá los resultados para todos los ítems del ejercicio anterior. Usá $n = 3$, $f.x = (x = 0)$, $x = -1$, $xs = [1, 0, 3]$.

19. Aplicá el **cambio de variable** indicado, si es que se puede. Explicá porqué puede o no puede aplicarse.

    a) $\langle \sum i \ : |i| < 5 : \ i \ div \ 2 \rangle$ con $i \to 2 * i$ (o sea $f.i = 2 * i$)
    b) $\langle \sum i \ : i \bmod 2 = 0 \wedge |i| < 5 : \ i \ div \ 2 \rangle$ con $i \to 2 * i$ (o sea $f.i = 2 * i$)
    c) $\langle \prod i \ : 0 < i \le \#(x \triangleright xs) : \ (x \triangleright xs).i \rangle$ con $i \to i + 1$ (o sea $f.i = i + 1$)
    d) $\langle \text{Max} \ as \ : as \neq [\,] : \ \#as \rangle$ con $(a, as) \to a \triangleright as$ (la función es $f.(a, as) = a \triangleright as$)

20. Simplificá el rango y aplicá alguna de las **reglas para la cuantificación de conteo**:

    a) $\langle N \, a, as \ : a \triangleright as = xs \wedge xs = [\,] : \ \#as = 1 \rangle$
    b) $\langle N \, i \ : i - n = 1 : \ i \bmod 2 = 0 \rangle$
    c) $\langle N \, i \ : i = 0 \vee 1 \le i < \#xs + 1 : \ ((x \triangleright xs).i) \bmod 2 = 0 \rangle$

21. Demostrá la siguiente propiedad:
    $\langle \sum i \ : 0 \le i < \#(x \triangleright xs) : \ T.((x \triangleright xs).i) \rangle = T.x + \langle \sum i \ : 0 \le i < \#xs : \ T.(xs.i) \rangle$

22. (*) **(Separación de un término)** Demostrá los siguientes teoremas útiles para la materia.
    Suponé que $\bigoplus$ es un cuantificador asociado a un operador genérico $\oplus$, que es conmutativo y asociativo (así como el $\forall$ es el cuantificador asociado a la conjunción $\wedge$) y $n : Nat$.

    a) $\langle \bigoplus i \ : 0 \le i < n + 1 : \ T.i \rangle = \langle \bigoplus i \ : 0 \le i < n : \ T.i \rangle \oplus T.n$
    b) $\langle \bigoplus i \ : 0 \le i < n + 1 : \ T.i \rangle = T.0 \oplus \langle \bigoplus i \ : 0 \le i < n : \ T.(i + 1) \rangle$

23. **(Rango unitario generalizado)** Sea $\oplus$ un cuantificador asociado a un operador conmutativo y asociativo. Probá la siguiente regla de *rango unitario generalizado* (Z no depende de i ni de j):

$$\langle \bigoplus i, j \ : i = Z \wedge R.i.j : \ T.i.j \rangle = \langle \bigoplus j \ : R.Z.j : \ T.Z.j \rangle \ .$$

24. Podemos definir un cuantificador de *conteo* N utilizando la sumatoria:

$$\langle \text{N} \, i \ : R.i : \ T.i \rangle \doteq \langle \sum i \ : R.i \wedge T.i : \ 1 \rangle$$

    Demostrá que $\langle \sum i \ : \ R.i \wedge T.i \ : \ k \rangle = \langle \text{N} i \ : \ R.i \ : \ T.i \rangle * k$

25. (*) Demostrá la siguiente relación entre los cuantificadores de máximo y mínimo cuando R es no vacío:

$$n = \langle \text{Min} \ i \ : \ R.i \ : \ -T.i \rangle \equiv n = -\langle \text{Max} \ i \ : \ R.i \ : \ T.i \rangle$$

26. (*) Demostrá los siguientes teoremas sobre $\forall$, utilizando los axiomas y teoremas del digesto:

    a) *Intercambio de* $\forall$ *(generalizada)*:
    $\langle \forall i \ : \ R.i \wedge S.i \ : \ T.i \rangle \equiv \langle \forall i \ : \ R.i \ : \ S.i \Rightarrow T.i \rangle$

b) *Instanciación de* $\forall$:
$\langle \forall i : : T.i \rangle \Rightarrow T.x$, cuando x no está cuantificada.
¿Como sería la regla de instanciación para $\exists$? Enunciala y demostrala.

Alto orden y ejercicios de laboratorio

El objetivo de estos ejercicios es ejercitar más definiciones de funciones y comenzar a introducir algunos conceptos de polimorfismo y funciones de alto orden. Esta guía se abordará principalmente en el laboratorio.

27. Definí recursivamente las siguientes funciones.

a) La función *paratodo*, que dada una lista de valores $xs : [A]$ y un predicado $T : A \to Bool$, determina si todos los elementos en xs hacen verdadero el predicado T, es decir:

$$paratodo : [A] \to (A \to Bool) \to Bool$$

$$paratodo.xs.T \equiv \langle\, \forall\, i \ : 0 \leq i < \#xs : \ T.(xs.i) \,\rangle$$

Puede ser de ayuda recordar la función del ejercicio 10.

b) La función *existe*, que dada una lista de valores $xs : [A]$ y un predicado $T : A \to Bool$, determina si algún elemento en xs hace verdadero el predicado T, es decir:

$$existe : [A] \to (A \to Bool) \to Bool$$

$$existe.xs.T \equiv \langle\, \exists\, i \ : 0 \leq i < \#xs : \ T.(xs.i) \,\rangle$$

Puede ser de ayuda recordar la función del ejercicio 9.

c) La función *sumatoria*, que dada una lista de valores $xs : [A]$ y una función $T : A \to Num$ (toma elementos de A y devuelve números), calcula la suma de la aplicación de T a los elementos en xs es decir:

$$sumatoria : [A] \to (A \to Num) \to Num$$

$$sumatoria.xs.T = \langle \sum i \ : 0 \leq i < \#xs : \ T.(xs.i) \,\rangle$$

d) La función *productoria*, que dada una lista de valores $xs : [A]$ y una función $T : A \to Num$, calcula el producto de la aplicación de T a los elementos de xs, es decir:

$$productoria : [A] \to (A \to Num) \to Num$$

$$productoria.xs.T = \langle \prod i \ : 0 \leq i < \#xs : \ T.(xs.i) \,\rangle$$

Laboratorio 7 Programá las funciones definidas en el ejercicio 27. Para definir los tipos en haskell lo podés hacer de la siguiente manera:

a) `paratodo' :: [a] -> (a -> Bool) -> Bool`

b) `existe' :: [a] -> (a -> Bool) -> Bool`

c) `sumatoria' :: [a] -> (a -> Int) -> Int`

d) `productoria' :: [a] -> (a -> Int) -> Int`

Laboratorio 8 Teniendo en cuenta las funciones definidas en el Laboratorio 1 y en el Laboratorio 7, evaluá las expresiones como se muestra en el ejemplo.

Ejemplos en `ghci`:

```
$> paratodo' [0,0,0,0] esCero
True
$> paratodo' [0,0,1,0] esCero
False
```

```
$> paratodo' "hola" esVocal
False
$> existe' [0,0,1,0] esCero
True
$> existe' "hola" esVocal
True
$> existe' "tnt" esVocal
False
```

Laboratorio 9

Utilizando las funciones del Laboratorio 7, programá las siguientes funciones por composición, sin usar recursión ni análisis por casos. Tenés en cuenta que puede ser necesario definir funciones auxiliares para que las definiciones sean más claras.

a) `todosPares :: [Int] -> Bool` verifica que todos los números de una lista sean pares.

b) `hayMultiplo :: Int -> [Int] -> Bool` verifica si existe algún número dentro del segundo parámetro que sea múltiplo del primer parámetro.

c) `sumaCuadrados :: Int -> Int`, dado un número no negativo n, calcula la suma de los primeros n cuadrados, es decir $\langle \sum i : 0 \leq i < n : i^2 \rangle$. **Ayuda:** En Haskell se puede escribir la lista que contiene el rango de números entre n y m como `[n..m]`.

d) Programar la función `existeDivisor::Int-> [Int] -> Bool`, que dado en entero `n` y una lista `ls` , devuelve `True` si y solo si, existe algún elemento en `ls` que divida a `n`a.

e) Utilizando la función del apartado anterior, definí la función `esPrimo:: Int -> Bool`, que dado un entero `n`, devuelve `True` si y solo si `n` es primo. **Ayuda:** En Haskell se puede escribir la lista que contiene el rango de números entre n y m como `[n..m]`.

f) ¿Se te ocurre como redefinir `factorial` (ej. 3*d*) para evitar usar recursión?

g) Programar la función `multiplicaPrimos :: [Int] -> Int` que calcula el producto de todos los números primos de una lista.

h) Programar la función `esFib :: Int -> Bool`, que dado un entero `n`, devuelve `True` si y sólo si `n` está en la sucesión de Fibonacci. **Ayuda:** Realizar una función auxiliar `fib :: Int -> Int` que dado un `n` devuelva el *n*-ésimo elemento de la sucesión.

i) Utilizando la función del apartado anterior, definí la función `todosFib :: [Int] -> Bool` que dada una lista `xs` de enteros, devuelva si todos los elementos de la lista pertenecen (o no) a la sucesión de Fibonacci.

28. Indagá en Hoogle sobre las funciones `map` y `filter`. También podes consultar su tipo en `ghci` con el comando `:t`.

    - ¿Qué hacen estas funciones?
    - ¿A qué equivale la expresión `map succ [1, -4, 6, 2, -8]`, donde `succ n = n+1`?
    - ¿Y la expresión `filter esPositivo [1, -4, 6, 2, -8]`?

29. Programá una función que dada una lista de números xs, devuelve la lista que resulta de duplicar cada valor de xs.

    a) Definila usando recursión.

    b) Definila utilizando la función `map`.

Laboratorio 10 Implementá en Haskell las funciones definidas en el ejercicio anterior.

Laboratorio 11 Programá una función que dada una lista de números xs, calcula una lista que tiene como elementos aquellos números de xs que son primos.

a) Definila usando recursión.

b) Definila utilizando la función `filter`.

c) Revisá tu definición del ejercicio g. ¿Cómo podes mejorarla?

Laboratorio 12 La función `primIgualesA` toma un valor y una lista, y calcula el tramo inicial más largo de la lista cuyos elementos son iguales a ese valor. Por ejemplo:

```
primIgualesA 3 [3,3,4,1] = [3,3]
primIgualesA 3 [4,3,3,4,1] = []
primIgualesA 3 [] = []
primIgualesA 'a' "aaadaa" = "aaa"
```

a) Programá `primIgualesA` por recursión.

b) Programá nuevamente la función utilizando `takeWhile`.

Laboratorio 13 La función `primIguales` toma una lista y devuelve el mayor tramo inicial de la lista cuyos elementos son todos iguales entre sí. Por ejemplo:

```
primIguales [3,3,4,1] = [3,3]
primIguales [4,3,3,4,1] = [4]
primIguales [] = []
primIguales "aaadaa" = "aaa"
```

a) Programá `primIguales` por recursión.

b) Usá cualquier versión de `primIgualesA` para programar `primIguales`. Está permitido dividir en casos, pero no usar recursión.

30. (*) Todas las funciones del ejercicio 7 son similares entre sí: cada una aplica la función término t a todos los elementos de una lista, y luego aplica algún operador entre todos ellos, obteniéndose así el resultado final. Para el caso de la lista vacía, se devuelve el elemento neutro. De esa manera cada una de ellas computa una cuantificación sobre los elementos de la lista transformados por t:

$$\begin{aligned} \texttt{paratodo'}.xs.t &= \langle \forall\, i : 0 \le i < \#xs : t.xs.i \rangle \\ \texttt{existe'}.xs.t &= \langle \exists\, i : 0 \le i < \#xs : t.xs.i \rangle \\ \texttt{sumatoria'}.xs.t &= \langle \Sigma\, i : 0 \le i < \#xs : t.xs.i \rangle \\ \texttt{productoria'}.xs.t &= \langle \Pi\, i : 0 \le i < \#xs : t.xs.i \rangle \end{aligned}$$

Por ejemplo, para `sumatoria'` el operador asociado al cuantificador Σ es la suma $(+)$, por lo que

```
sumatoria' [1,2,3] t = (t 1) + (t 2) + (t 3) + 0
```

donde el cálculo consistió en aplicar t a cada elemento, combinándolos con el operador $(+)$ hasta llegar a la lista vacía donde se devuelve el neutro de la suma (0). Guiándote por las observaciones anteriores, definí de manera recursiva la función *cuantGen* (denota la cuantificación generalizada):

```
cuantGen :: (b -> b -> b) -> b -> [a] -> (a -> b) -> b
cuantGen op z xs t = ...
```

que tomando como argumento un operador `op`, su elemento neutro `z`, una lista de elementos `xs` y una función término `t`, aplica el operador a los elementos de la lista, transformados por la función término. En otras palabras, sea $\bigoplus$ un cuantificador cualquiera y $\oplus$ su operador asociado,

$$cuantGen.\oplus.z.xs.t = \langle \bigoplus\, i : 0 \le i < \#xs : t.(xs\,!\,i) \rangle$$

Laboratorio 14 (*) A partir de lo realizado en el ejercicio anterior

a) Definí en Haskell la función `cuantGen :: (b -> b -> b) -> b -> [a] -> (a -> b) -> b`.

b) Reescribí todas las funciones del laboratorio 7 utilizando el cuantificador generalizado (sin usar inducción y en una línea por función).

Laboratorio 15 (*) Definí una función primeros que cumplen, `primQueCumplen::[a]->(a->Bool)->[a]`, tal que, dada una lista `ls` y un predicado `p`, devuelve el tramo inicial de `ls` que cumple `p`.

Laboratorio 16 (*) Para cada uno de los siguientes patrones, decidí si están bien tipados, y en tal caso dá los tipos de cada subexpresión. En caso de estar bien tipado, ¿el patrón cubre todos los casos de definición?

a)
```
f :: (a, b) -> ...
f (x , y) = ...
```

b)
```
f :: [(a, b)] -> ...
f (a , b) = ...
```

c)
```
f :: [(a, b)] -> ...
f (x:xs) = ...
```

d)
```
f :: [(a, b)] -> ...
f ((x, y) : ((a, b) : xs)) = ...
```

e)
```
f :: [(Int, a)] -> ...
f [(0, a)] = ...
```

f)
```
f :: [(Int, a)] -> ...
f ((x, 1) : xs) = ...
```

g)
```
f :: (Int -> Int) -> Int -> ...
f a b = ...
```

h)
```
f :: (Int -> Int) -> Int -> ...
f a 3 = ...
```

i)
```
f :: (Int -> Int) -> Int -> ...
f 0 1 2 = ...
```

Laboratorio 17 (*) Para las siguientes declaraciones de funciones, da al menos una definición cuando sea posible. ¿Podés dar alguna otra definición alternativa a la que diste en cada caso?

Por ejemplo, si la declaración es `f :: (a, b) -> a`,
la respuesta es: `f (x,y) = x`

a) `f :: (a, b) -> b`

b) `f :: (a, b) -> c`

c) `f :: (a -> b) -> a -> b`

d) `f :: (a -> b) -> [a] -> [b]`