Práctico 0: Repaso

Algoritmos y Estructuras de Datos I
2do cuatrimestre 2025

Objetivos

El objetivo de esta guía es recordar conceptos de Introducción a los Algoritmos que son importantes para esta materia.

1. 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:

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

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

a) `esCero :: Int -> Bool`, que verifica si un entero es igual a 0.

b) `esPositivo :: Int -> Bool`, que verifica si un entero es estrictamente mayor a 0.

c) `esVocal :: Char -> Bool`, que verifica si un caracter es una vocal en minúscula.

d) `valorAbsoluto :: Int -> Int`, que devuelve el valor absoluto de un entero ingresado.

2. 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) `todosPositivos :: [Int] -> Bool`, que verifica que todos los valores de una lista sean positivos.

   b) `hayPares :: [Int] -> Bool`, que da verdadero si alguno de los valores de la lista de enteros dada es par.

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

   d) `existe :: [Bool] -> Bool`, que verifica que *existe* algún elemento `True` en la lista.

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
$> hayPares [1, 5, -4]
True
$> todosPositivos [1, 5, -4]
False
$> todosPositivos [2, 4, 1]
True
```

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

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

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

   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 3*e* e 3*f*?

4. 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 3*a*, 3*b* y 3*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 4.

5. 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.

Lista de figuras

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.

En `Haskell` *lo definiremos así:*

```
data Color = Rojo | Amarillo | Azul | Verde
                        deriving (Show, Eq)

data Forma = Triangulo | Cuadrado | Rombo | Circulo
                        deriving (Show, Eq)

type Figura = (Forma, Color, Int)
```

Para Haskell es necesario, luego, definir los predicados y la función tam correspondientes.

Laboratorio 4 Definí en Haskell el tipo de datos `Figura` y los predicados asociados que caracterizan la forma y el color. También definí la función `tam`, que tome una `Figura` y devuelva su tamaño. Por ejemplo:

```
rojo :: Figura -> Bool
rojo (f,c,t) = c == Rojo
```

6. 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) $\langle \exists\, i \;:\; 0 \leq i < \#xs :\; rombo.(xs.i) \rangle$

Laboratorio 5 En el ejercicio 6 observamos que todas las expresiones tienen como única variable libre a xs. Es decir, el valor de la expresión, si la queremos evaluar, depende del valor de esa variable. Por lo tanto, para cada expresión podemos definir una función, que tome una lista de figuras y devuelva el valor de esa expresión para esa lista.

a) Definí funciones que calculen el valor de las expresiones en el ejercicio 6, para una lista dada, para los items a) y b) . Por ejemplo, para la expresión

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

definimos la función `todasRojas` de la siguiente manera

```
todasRojas :: [Figura] -> Bool
todasRojas [] = True
todasRojas (x:xs) = rojo x && todasRojas xs
```

b) Evaluá las funciones definidas para las listas:

```
[(Circulo, Azul, 40), (Rombo, Rojo, 10)]
[(Circulo, Rojo, 4), (Rombo, Rojo, 10)]
[ (Triangulo, Verdo, 10)]
```

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

   a) Todas las figuras de xs son amarillas.

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

   c) Todas las figuras de xs son de tamaño menor a 5.

   d) Todos los triángulos de xs son rojos.

   e) Existe un cuadrado verde en xs.

   f) Todos los círculos de xs son azules y de tamaño menor a 10.

   g) Ningún triángulo de xs es azul.

   h) En xs no hay círculos amarillos ni verdes.

   i) Existe (al menos) un cuadrado de tamaño menor a 5 en xs.

   j) Si hay círculos rojos en xs entonces hay cuadrados rojos.

Laboratorio 6 De la misma manera que en el ejercicio 6, en el ejercicio 7 observamos que todas las expresiones tienen como única variable libre a xs. Es decir, el valor de la expresión, si la queremos evaluar, depende del valor de esa variable. Por lo tanto, para cada expresión podemos definir una función, que tome una lista de figuras y devuelva el valor de esa expresión para esa lista.

Implementá en Haskell funciones recursivas para evaluar las propiedades del ejercicio anterior.