PLP - Recuperatorio del Primer Parcial - 1er cuatrimestre de 2024

#Orden	Nro. Libreta	Apellido(s)	Nombre(s)
81		Fialkowski	Valentín.

Corregido por	Nota E1	Nota E2	Nota E3	Nota Final
PERLA	B-	B-	B-	A

Este examen se aprueba obteniendo al menos dos ejercicios bien menos (B-) y uno regular (R). Las notas para cada ejercicio son: -, I, R, B-, B. Entregar cada ejercicio en hojas separadas. Poner nombre, apellido y número de orden en todas las hojas, y numerarlas. Se puede utilizar todo lo definido en las prácticas y todo lo que se dio en clase, colocando referencias claras. El orden de los ejercicios es arbitrario. Recomendamos leer el parcial completo antes de empezar a resolverlo.

Ejercicio 1 - Programación funcional

Aclaración: en este ejercicio no está permitido utilizar recursión explícita, a menos que se indique lo contrario.

En este ejercicio vamos a modelar lógica proposicional en Haskell, de modo de poder construir fórmulas proposicionales y evaluarlas bajo distintas valuaciones.

```
data Prop = Var String | No Prop | Y Prop Prop | O Prop Prop | Imp Prop Prop

type Valuación = String -> Bool
```

Por ejemplo, la expresión: `Y (Var ''P'') (No (Imp (Var ''Q'') (Var ''R'')))` representa la proposición $P \wedge \neg(Q \Rightarrow R)$.

Las valuaciones se representan como funciones que a cada variable proposicional le asignan un valor booleano. Por ejemplo la valuación `\x -> x == ''P''` le asigna el valor verdadero a la variable P y falso a todas las otras variables proposicionales.

a) Dar el tipo y definir las funciones `foldProp` y `recProp`, que implementan respectivamente los esquemas de recursión estructural y primitiva para el tipo `Prop`. Solo en este inciso se permite usar recursión explícita.

b) Definir la función `variables :: Prop -> [String]`, que dada una fórmula devuelve la lista con todas sus variables proposicionales en algún orden, sin elementos repetidos.

Por ejemplo: `variables (O (Var ''P'') (No (Y (Var ''Q'') (Var ''P''))))` debería devolver la lista `[''P'',''Q'']` o la lista `[''Q'',''P'']`.

c) Definir la función `evaluar :: Valuación -> Prop -> Bool`, que indica si una fórmula es verdadera o falsa para una valuación dada.

d) Definir la función `estáEnFNN :: Prop -> Bool`, que indica si una fórmula está en Forma Normal Negada. Es decir, si no tiene implicaciones y la negación se aplica únicamente a variables y no a proposiciones más complejas.

Por ejemplo: `Y (Var ''P'') (No (Imp (Var ''Q'') (Var ''R'')))` no está en FNN, y en cambio `Y (Var ''P'') (Y (Var ''Q'') (No (Var ''R'')))` sí lo está.

Ejercicio 2 - Demostración e inferencia

Considerar las siguientes definiciones sobre árboles con información en las hojas[1]:

```
data AIH a = Hoja a | Bin (AIH a) (AIH a)
                                           der :: AIH a -> AIH a
     esHoja :: AIH a -> Bool           {D} der (Bin i d) = d
{E0} esHoja (Hoja x) = True
{E1} esHoja (Bin i d) = False              mismaEstructura :: AIH a -> AIH a -> Bool
                                      {M0} mismaEstructura (Hoja x) = esHoja
     izq :: AIH a -> AIH a            {M1} mismaEstructura (Bin i d) = \t -> not (esHoja t) &&
{I}  izq (Bin i d) = i                        mismaEstructura i (izq t) && mismaEstructura d (der t
```

a) Demostrar la siguiente propiedad:

$\forall$ `t::AIH a` . $\forall$ `u::AIH a` . mismaEstructura t u = mismaEstructura u t

Se recomienda hacer inducción en el primer árbol, utilizando extensionalidad en el segundo. Se permite definir macros (poner nombres a expresiones largas para no tener que repetirlas).

No es obligatorio reescribir los $\forall$ correspondientes en cada paso, pero es importante recordar que están presentes y escribir los que correspondan al platear la propiedad como predicado unario. Recordar también que los = de las definiciones pueden leerse en ambos sentidos.

Se consideran demostradas todas las propiedades conocidas sobre enteros y booleanos.

b) Usar el algoritmo **W** para inferir juicios de tipado válidos para las siguientes expresiones, o indicar por qué no es posible (recordar que en inferencia **no está permitido** renombrar variables):

i) $(\lambda x.x(\lambda x.\mathsf{Succ}(x)))(\lambda x.x)$

ii) $\lambda x.\mathsf{if}\ \mathsf{isZero}(x)\ \mathsf{then}\ x\ \mathsf{else}\ x\ \mathsf{zero}$

Ejercicio 3 - Cálculo Lambda Tipado

Se desea extender el cálculo lambda simplemente tipado para modelar **Árboles con información en las hojas**. Para eso se extienden los tipos y expresiones de la siguiente manera:

$\tau ::= \cdots \mid \mathtt{AIH}(\tau)$

$M ::= \cdots \mid \mathtt{Hoja}(M) \mid \mathtt{Bin}(M, M) \mid \mathsf{case}\ M\ \mathsf{of\ Hoja}\ x \leadsto M;\ \mathsf{Bin}(i, d) \leadsto M$

- $\mathtt{AIH}(\tau)$ es el tipo de los árboles con información en las hojas de tipo τ.
- $\mathtt{Hoja}(M)$ es un árbol compuesto por una única hoja con información M.
- $\mathtt{Bin}(M_1, M_2)$ es un árbol compuesto por dos subárboles M_1 y M_2.
- El observador $\mathsf{case}\ M_1\ \mathsf{of\ Hoja}\ x \leadsto M_2;\ \mathsf{Bin}(i, d) \leadsto M_3$ permite acceder al valor de un árbol que es hoja (el cual se ligará a la variable x que puede aparecer libre en M_2), y a los dos subárboles de un árbol que no es hoja (los cuales se ligarán a las variables i y d que pueden aparecer libres en M_3).

a. Introducir las reglas de tipado para la extensión propuesta.

b. Definir el conjunto de valores y las nuevas reglas de semántica operacional a pequeños pasos, tanto de congruencia como de cómputo.

c. Mostrar paso por paso cómo reduce la expresión:

$\mathsf{case}\ (\lambda n : \mathsf{Nat}.\ \mathtt{Hoja}(n))\ \mathsf{Succ}(\mathsf{zero})\ \mathsf{of\ Hoja}\ x \leadsto \mathsf{Succ}(\mathsf{Pred}(x));\ \mathsf{Bin}(i, d) \leadsto \mathsf{zero}$

d. Definir como macro la función esHoja_τ, que toma un $\mathtt{AIH}(\tau)$ y devuelve un booleano que indica si es una hoja.

[1] Escritas con recursión explícita para facilitar las demostraciones.