PLP - Recuperatorio de Primer Parcial - 2^{do} cuatrimestre de 2006

Este examen se aprueba obteniendo al menos **70 puntos**. Poner nombre, apellido y número de órden en cada hoja, y numerarlas. Se puede utilizar todo lo definido en las prácticas y todo lo que se dió en clase, colocando referencias claras.

Ejercicio 1 - Programación funcional (40 puntos)

En ningún ítem de este ejercicio se puede hacer recursión explícita, salvo que se indique explícitamente lo contrario.

Este bien, mal o no resuelto un ítem, en los ítems subsiguientes se puede utilizar la función o funciones descriptas en el enunciado como si existieran. Se recomienda no abandonar todo el ejercicio si un ítem del medio no sale.

El objetivo final de este ejercicio es hacer un programa que busque las raices racionales de polinomios con coeficientes enteros.

Consideremos la siguiente representación de fracciones y polinomios, junto a los observadores para las fracciones [1]:

```
data Frac = F Int Int
data Poly = Var | Cst Int | Add Poly Poly | Mul Poly Poly | Neg Poly

numer,den :: Frac → Int
numer (F n d) = n
den (F n d) = d
```

Los polinomios entonces están definidos por inducción como una variable (el polinomio $P(x) = x$), una constante (el polinomio $P(x) = k, k \neq 0$) o la suma, producto o negación (opuesto) de polinomios. Notar que esta definición permite definir cualquier polinomio de coeficientes enteros y sólo éstos (algunos de más de una manera).[2]

a) **(5 puntos)** Escribir la función `divisores` que dado un entero devuelve la lista de sus divisores, en cualquier órden (el comportamiento está indefinido para el 0). Notar que el parámetro puede ser un entero negativo o positivo y que se deben devolver los divisores negativos y positivos del mismo. Se recomienda utilizar la función `rem` del `Prelude` que dados dos enteros devuelve el resto de dividir el primero por el segundo (da error si el segundo es cero).

```
divisores :: Int → [Int]

divisores 15 → [1,-1,3,-3,5,-5,15,-15]
```

b) **(5 puntos)** Escribir un esquema de recursión (*fold*) para el tipo `Poly`. Por supuesto, se puede hacer recursión explicita en este ejercicio.

```
foldPoly :: b → (Int → b) → (b → b → b) →
        (b → b → b) → (b → b) → Poly → b
```

[1] No nos interesa que la misma fracción pueda tener muchas representaciones.

[2] El polinomio nulo ($P(x) = 0$ ó `Cst 0`) no será considerado un polinomio en este ejercicio y no nos preocuparemos por él.

c) **(3 puntos)** Escribir las funciones `addF`, `mulF` y `negF` que representan la suma, multiplicación y negación de fracciones, respectivamente.

```
addF,mulF :: Frac → Frac → Frac
negF :: Frac → Frac
```

d) **(4 puntos)** Escribir la función `eval` que dado un polinomio P y una fracción f, devuelve el resultado de la evaluación $P(f)$.

```
eval :: Poly → Frac → Frac
```

e) **(4 puntos)** Escribir la función `grado` que dado un polinomio devuelve su grado. [3]

```
grado :: Poly → Int
```

f) **(4 puntos)** Escribir la función `coefIndepte` que dado un polinomio devuelve su coeficiente independiente. [4]

```
coefIndepte :: Poly → Int
```

g) **(5 puntos)** Escribir la función `coefPpal` que dado un polinomio devuelve su coeficiente principal. Nota: El coeficiente principal de la suma de 2 polinomios **no** es necesariamente igual a la suma de sus coeficientes principales. [5] Para esta función se puede utilizar recursión explícita.

```
coefIndepte :: Poly → Int
```

Según el lema de gauss, las raices racionales de un polinomio con coeficientes enteros son de la forma p/q donde p es un divisor del coeficiente independiente y q lo es del coeficiente principal.

h) **(5 puntos)** Escribir la función `posRaices` que dado un polinomio devuelve las posibles raices según el lema de Gauss. No importa si se devuelven múltiples representaciones de la misma fracción, mientras la lista sea finita, incluya todas las posibilidades y no incluya ninguna posibilidad falsa. Se puede asumir que el polinomio pasado como parámetro tiene coeficiente independiente distinto de 0.

```
posRaices :: Poly → [Frac]
```

i) **(5 puntos)** Escribir la función `raices` que dado un polinomio devuelve sus raices racionales. Se recomienda fuertemente utilizar el ejercicio anterior. No importa si se devuelven múltiples representaciones de la misma fracción, mientras la lista sea finita, incluya todas las raices y no incluya ninguna raiz falsa.

```
raices :: Poly → [Frac]
```

[3] El grado de un polinomio es el exponente mas grande al cual se eleva la variable cuando se lo escribe por sus monomios. El grado de un polinomio constante es 0.

[4] El coeficiente independiente es la constante que acompaña al monomio x^0.

[5] El coeficiente principal es la constante que acompaña al monomio x^k dónde $k = grado(P)$.

Ejercicio 2 - PCF (30 puntos)

Para un importante proyecto de computación distribuida se desea formalizar algunas ideas. Para dar un contexto a dichas formalizaciones se creará PCF-V, una extensión de PCF para dar soporte a computaciones vectoriales. Una computación vectorial es la computación de la misma función sobre muchos elementos al mismo tiempo. En una primera etapa del proyecto se extendieron los tipos y la sintaxis para soportar el nuevo tipo vectorial de la siguiente manera:

$$\sigma ::= ... \mid \sigma^k \qquad\qquad M ::= ... \mid (M_1, ..., M_k)$$

En esta notación σ^k representa un vector de k posiciones, cada una de tipo σ y $(M_1, ..., M_k)$ es la forma de construir un vector de k elementos. Las reglas de tipado y semántica de la extensión ya realizadas son:

$$\frac{\Gamma \rhd M_1 : \sigma, ..., \Gamma \rhd M_k : \sigma}{\Gamma \rhd (M_1, ..., M_k) : \sigma^k} \qquad\qquad \frac{M_i \rightarrow_{cbn} M_i'}{(M_1, ..., M_i, ..., M_k) \rightarrow_{cbn} (M_1, ..., M_i', ..., M_k)}$$

Lo que se desea, en este caso, es continuar la extensión con 2 construcciones mas: $++$ que permite agregar resultados previos (vectores) del mismo tipo en un vector nuevo "más grande" y $\forall$ que permite testear propiedades sobre los resultados.
La sintaxis para esta extensión es la siguiente:

$$\sigma ::= ... \mid \sigma^k \qquad\qquad M ::= ... \mid (M_1, ..., M_k) \mid M \ ++ \ N \mid \forall \ x \in M \ check \ N$$

La semántica de $++$ es la concatenación de 2 vectores y la de $\forall$ es chequear si la propiedad mencionada en N se hace verdadera reemplazando todas las apariciones **libres** de x en N por cada uno de los elementos del vector M. Ejemplos:

- $(1, 2, 3) + +(4, 5) \rightarrow (1, 2, 3, 4, 5)$
- $(1, 2, 3) + +(true, false)$ No está bien tipado
- $\forall \ x \in (1, 2, 3) \ Eq? \ x \ 1 \rightarrow false$
- $\forall \ x \in (1, 1, 1) \ Eq? \ x \ 1 \rightarrow true$
- $\forall \ x \in (1, 3, 5) \ (\lambda \ x.Eq? \ x \ 1)$ No está bien tipado

Se pide, para la extensión mencionada, definir:

a) **(16 puntos)** Reglas de tipado

b) **(4 puntos)** Semántica operacional *small-step* **solo para la construcción** $++$.

c) **(10 puntos)** Semántica operacional *big-step* **para ambas construcciones**.

Nota: El determinismo para las reglas semánticas es deseable pero no necesario. Se evaluará de una solución que sea correcta (o sea, que no permita reducir cosas indeseables) y completa (que permita reducir todo lo deseable) respecto del problema planteado.

Sugerencia: Para la semántica del $\forall$ puede resultar conveniente hacer 2 reglas, una para el caso en el cual se debe reducir a $false$ y otra para el caso en que se debe reducir a $true$.

Ejercicio 3 - Inferencia de tipos (30 puntos)

Utilizando el árbol de inferencia, inferir el tipo de las siguientes expresiones o demostrar que no son tipables. En caso de ser tipables, dar una cortísima descripción de lo que hace la expresión.

a) **(8 puntos)** $\lambda x.\ \lambda y.\ \lambda z.\ (z(x\ y))\ (y\ x)$

b) **(8 puntos)** $\lambda x.\ \lambda y.\ \lambda z.\ (x\ z)\ y$

c) **(14 puntos)** Extender el algoritmo de inferencia para soportar la inferencia de tipado de PCF-V (con vectores). En esta extensión del algoritmo solo se consideraran los constructores de los vectores.

La sintaxis de esta extensión es la siguiente:

$\sigma ::= ... \mid \sigma^k$

$M ::= ... \mid (M_1, ..., M_k)$

Y sus reglas de tipado, las siguientes:

$$\frac{\Gamma \rhd M_1 : \sigma, ..., \Gamma \rhd M_k : \sigma}{\Gamma \rhd (M_1, ..., M_k) : \sigma^k}$$

Una posible entrada (válida) para el algoritmo es la siguiente:

$$(\lambda x.(1, x, 5 + 5, x + 5, x + x))\ 5$$