Práctica 5

Programación Funcional, UNQ

Propiedades y demostraciones

Aclaraciones:

- *Los ejercicios fueron pensados para ser resueltos en el orden en que son presentados. No se saltee ejercicios sin consultar antes a un docente.*
- *Recuerde que puede aprovechar en todo momento las funciones que ha definido, tanto las de esta misma práctica como las de prácticas anteriores.*
- *Pruebe todas sus implementaciones, al menos en una consola interactiva.*
- *Es sumamente aconsejable resolver los ejercicios utilizando primordialmente los conceptos y metodologías vistos en clase, dado que los exámenes de la materia evaluarán principalmente este aspecto. Si se encuentra utilizando formas alternativas al resolver los ejercicios consulte a los docentes.*
- *No dude en manifestar observaciones y críticas sobre los ejercicios de esta práctica, que con gusto serán recibidas por los docentes.*
- *Los ejercicios del anexo pueden obviarse, pero recuerde que aportan una compresión más profunda sobre los temas que aborda esta práctica. Considere resolverlos si se encuentra practicando para una instancia de evaluación y ya resolvió todos los anteriores.*

Esta práctica hace uso de funciones definidas en prácticas anteriores. En caso de que una función no haya sido requerida anteriormente, defínala aquí utilizando recursión. Considere dar una definición alternativa de las funciones si al hacerlo se simplifica la demostración.

1. Demostraciones simples

Indique si las siguientes afirmaciones son verdaderas o falsas y de una demostración:

a) `id . id` ≡ `id`

b) `swap . swap` ≡ `id`

c) `(\x y -> y)` ≡ `const id`

d) `curry . uncurry` ≡ `id`

e) `uncurry . curry` ≡ `id`

f) `curry snd` ≡ `curry (fst . swap)`

g) `flip (curry f)` ≡ `curry (f . swap)`

h) `fst` ≡ `uncurry const`

i) `snd` ≡ `uncurry (flip const)`

j) `swap` ≡ `uncurry (flip (,))`

k) `not (x && y)` ≡ `not x || not y`

l) `not (x || y)` ≡ `not x && not y`

m) `last [x]` ≡ `head [x]`

2. Inducción sobre los naturales

Indique claramente el esquema inductivo de los números naturales. Luego, considerando las funciones de suma, producto y comparación sobre los números naturales; demuestre las siguientes propiedades:

a) `(n + m) + k` ≡ `n + (m + k)`

b) `n * (m + k)` ≡ `n*m + n*k`

c) `n <= m && k <= p` ⇒ `n+k <= m+p`

d)* `gauss n` ≡ `div (n*(n+1)) 2`

```
   donde: gauss 0 = 0
          gauss n = n + gauss (n-1)
```

3. Inducción sobre listas

Indique claramente el esquema inductivo de las listas. Luego, demuestre las siguientes propiedades sobre listas:

a) `length (xs ++ ys)` ≡ `length xs ++ length ys`

b) `reverse (xs ++ ys)` ≡ `reverse ys ++ reverse xs`

c) `reverse . reverse` ≡ `id`

d) `length . reverse` ≡ `length`

e) `length . (map f)` ≡ `length`

f) `length` ≡ `sum . map (const 1)`

g) `elem` ≡ `any . (==)`

h) `(map f) . (map g)` ≡ `map (f . g)`

i) `map f (xs ++ ys)` ≡ `map f xs ++ map f ys`

j) `map f (concat xs)` ≡ `concat (map (map f) xs)`

4. Inducción sobre árboles

Indique claramente el esquema inductivo de los árboles binarios definidos en la práctica 4. Luego, demuestre las siguientes propiedades sobre árboles binarios:

a) `mirrorTip . mirrorTip` ≡ `id`

b) `heightTip` ≡ `heightTip . mirrorTip`

c) `walkover . mirrorTip` ≡ `reverse . walkover`

d) `walkover . (mapTip f)` ≡ `(map f) . walkover`

e) `leaves t` ≤ `nodes t + 1`

5. Inducción sobre polinomios

Considerando la representación de polinomios de la práctica 4, la función de evaluación allí definida y la siguiente operación de derivación:

```
derive :: Poly -> Poly
derive (Cte n) = Cte 0
derive X = Cte 1
derive (Add p q) = Add (derive p) (derive q)
derive (Mul p q) = Add (Mul (derive p) q) (Mul p (derive q))
```

Demuestre la siguiente propiedad para el caso de polinomios con coeficientes positivos:

a) `eval p 0 == 0` $\Rightarrow$ `eval p 1 <= eval (derive p) 1`

6. Inducción sobre fórmulas lógicas

Considerando la representación de fórmulas lógicas de la práctica 4, la función de evaluación allí definida y la siguiente operación que transforma una fórmula a forma normal negada (NNF):

```
nnf :: Logical -> Logical
nnf (Var x)      = Var x
nnf (And l1 l2) = And (nnf l1) (nnf l2)
nnf (Or  l1 l2) = Or (nnf l1) (nnf l2)
nnf (Not l)      = case l of
                     Var x -> Not (Var x)
                     Not l' -> nnf l'
                     And l1 l2 -> Or (nnf (Not l1)) (nnf (Not l2))
                     Or  l1 l2 -> And (nnf (Not l1)) (nnf (Not l2))
```

Demuestre la siguiente propiedad:

a) `eval l v` $\equiv$ `eval (nnf l) v`

Ayuda: Considere demostrar la "propiedad más fuerte" que valida este hecho para toda fórmula con una cantidad de constructores menor o igual a un n dado. Esto puede resultar particularmente útil al aplicar la hipótesis inductiva en el caso inductivo correspondiente al `Not`.