Esquema de Inducción

Grupo cNil

2025-09-16

Ejercicio 12

Descripción del problema de demostracion

Necesitamos demostrar que *toda la expresión tiene un literal más que su cantidad de operadores*. Los literales son las constantes y los rangos. Para esto se dispone de las siguientes definiciones.

```haskell
data Nat = Z | S Nat

suma :: Nat -> Nat -> Nat
suma Z m = m -- {S1}
suma (S n) m = S (suma n m) -- {S2}

cantLit :: Expr -> Nat
cantLit (Const _) = S Z                            -- {L1}
cantLit (Rango _ _) = S Z                          -- {L2}
cantLit (Suma a b) = suma (cantLit a) (cantLit b)   -- {L3}
cantLit (Resta a b) = suma (cantLit a) (cantLit b)  -- {L4}
cantLit (Mult a b) = suma (cantLit a) (cantLit b)   -- {L5}
cantLit (Div a b) = suma (cantLit a) (cantLit b)    -- {L6}

cantOp :: Expr -> Nat
cantOp (Const _) = Z                                -- {O1}
cantOp (Rango _ _) = Z                              -- {O2}
cantOp (Suma a b) = S (suma (cantOp a) (cantOp b))  -- {O3}
cantOp (Resta a b) = S (suma (cantOp a) (cantOp b)) -- {O4}
cantOp (Mult a b) = S (suma (cantOp a) (cantOp b))  -- {O5}
cantOp (Div a b) = S (suma (cantOp a) (cantOp b))   -- {O6}
```

Y del siguiente lema que podemos asumir como válido.
No hace falta demostrarlo:

$$\{CONMUT\} \quad \forall n,\ m :: Nat \ \cdot \ suma\ \ n\ \ m = suma\ \ m\ \ n$$

Dado que `cantList` y `cantOp` reciben un tipo de dato `Expr` haríamos bien en recordar cómo está compuesta su estructura.

Expr

```
data Expr = Const Float
          | Rango Float Float
          | Suma Expr Expr
          | Resta Expr Expr
          | Mult Expr Expr
          | Div Expr Expr
```

Propiedad a Demostrar

$\forall e :: Expr \ \cdot \ cantLit \ \ e = S \ \ (cantOp \ \ e)$

Demostración

a) Predicado Unario

Dado que la propiedad opera sobre expresiones `Expr` tiene sentido definir el *predicado unario* correspondiente a la demostracion por inducción estructural en una expresión `e :: Expr`. Queda definido como:

$$P(e) := \ cantList \ \ e = S \ \ (cantOp \ \ e)$$

b) Esquema formal de inducción estructural

Declaramos el principio de inducción estructural sobre *Expr*:

- Sea P un propiedad acerca de las expresiones de tipo *Expr* **tal que**

$$
\begin{aligned}
&(\forall x :: Float.\ P(Const\ x)\\
&\wedge \ \ \forall\ x :: Float.\ \forall\ y :: Float.\ P(Rango\ x\ y)\\
&\wedge \ \ \forall\ e1 :: Expr.\ \forall\ e2 :: Expr.\ ((P(e1)\ \wedge\ P(e2))\ \rightarrow\ P(Suma\ e1\ e2))\\
&\wedge \ \ \forall\ e1 :: Expr.\ \forall\ e2 :: Expr.\ ((P(e1)\ \wedge\ P(e2))\ \rightarrow\ P(Resta\ e1\ e2))\\
&\wedge \ \ \forall\ e1 :: Expr.\ \forall\ e2 :: Expr.\ ((P(e1)\ \wedge\ P(e2))\ \rightarrow\ P(Mult\ e1\ e2))\\
&\wedge \ \ \forall\ e1 :: Expr.\ \forall\ e2 :: Expr.\ ((P(e1)\ \wedge\ P(e2))\ \rightarrow\ P(Div\ e1\ e2)))
\end{aligned}
$$

entonces $\forall e :: Expr.\ P(e)$.

c) Demostración

Caso (base): Const Float

Queremos ver que:

$$\forall\ x :: Float.\ P(Const\ x) :=$$
$$cantList\ (Const\ x) = S\ (cantOp\ (Const\ x))$$

Sea x fijo veamos:

```
-- Por un lado
cantList (Const x) = S Z -- por {L1}

-- Por otro lado
S cantOp (Const x) = S Z -- por {O1}
```

Como ambos lados son iguales y x es fijo, el caso queda probado para todo x.

□

Caso (base): Rango Float Float

Queremos ver que:

$$\forall\ x :: Float.\ \forall\ y :: Float.\ P(Rango\ x\ y) :=$$
$$cantList\ (Rango\ x\ y) = S\ (cantOp\ (Rango\ x\ y))$$

Fijemos x e y y veamos:

```
-- Por un lado
cantList (Rango x y) = S Z -- por {L2}

-- Por otro lado
S cantOp (Rango x y) = S Z -- por {O2}
```

Como ambos lados son iguales y x e y son fijos, el caso general queda probado para todo x e y.

□

Caso (inductivo): Suma Expr Expr

Queremos ver que:

$$\forall\ e1 :: Expr.\ \forall\ e2 :: Expr.\ ((P(e1)\ \wedge\ P(e2))\ \rightarrow\ P(Suma\ e1\ e2))$$

Fijemos $e1$ y $e2$. Supongamos que vale la $HI : P(e1) \wedge P(e2)$, donde:

$$P(e1) := cantList\ e1 = S\ (cantOp\ e1)$$
$$P(e2) := cantList\ e2 = S\ (cantOp\ e2)$$

Ahora veamos que se cumple nuestra TI:

$$P(Suma\ e1\ e2) := cantList\ (Suma\ e1\ e2) = S\ (cantOp\ (Suma\ e1\ e2))$$

```
cantList (Suma e1 e2)
= suma (cantList e1) (cantList e2)        -- por {L3}
= suma (S (cantOp e1)) (cantOp e2)        -- por HI pues se cumple P(e1)
= suma (S (cantOp e1)) (S (cantOp e2))    -- por HI pues se cumple P(e2)
= S (suma (cantOp e1)) (S (cantOp e2))    -- por {S2} donde m = S (cantOp e2)
= S (suma (S (cantOp e2)) (cantOp e1))    -- por {CONMUT}
= S (S (suma (cantOp e2) (cantOp e1)))    -- por {S2} donde m = cantOp e1
= S (S (suma (cantOp e1) (cantOp e2)))    -- por {CONMUT}
= S (cantOp (Suma e1 e2))                 -- por {O3}
-- Como se quería probar.
```

Como $e1$ y $e2$ son fijos, el caso queda demostrado para todo $e1$ y $e2$.

□

Los demás casos inductivos son análogos a este último (cambiar “Suma” por “Mult”, “Resta” y “Div” usando {L4}, {L5}, {L6} y {O4}, {O5}, {O6} en vez de {L3} y {O3} respectivamente en cada caso).

Por lo tanto: $\forall\ e :: Expr.\ P(e)$.

■